Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik

Towards Efficient Human Activity Recognition

Okan Köpüklü

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr.-Ing. Alexander W. Koch

Prüfende der Dissertation:
1. Prof. Dr.-Ing. Gerhard Rigoll
2. Prof. Dr. Daniel Rückert

Die Dissertation wurde am 17.11.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 24.05.2022 angenommen.

Acknowledgements

This thesis would not have been possible without the support of many people. First of all, I would like to thank my PhD supervisor Prof. Gerhard Rigoll for giving me the opportunity to work at the Institute for Human-Machine Communication at TUM and for supporting me with his advice during my PhD project. I thank Prof. Daniel Rückert for being a member of the thesis committee and Prof. Alexander Koch for chairing the examination.

I would like to thank my institute colleagues Neslihan, Michael, Fabian, Stefan, Martin, Patrick, Tobias, Ludwig, Lujun, Mikhail, Ioannis, Maryam, Torben, Maximilian, Johannes and others that I may have forgotten for the beautiful time I spent in MMK.

Munich has a special place in my heart. Not because it is the city that I lived most in my life, but because it is the city that witnessed many remarkable memories. Those memories would not be possible without my good-hearted friends Halil, Tuğçe, Cemil, Murat, M. Hilal, Mahmut, Ebru, Umur, Özge, Serhan, Çağatay, Erdem and Selin. I thank them all.

Most importantly, I would like to thank my beloved wife Hilal for all her support and encouragement. I could not imagine how I would finish this thesis if it were not for her constant love and faith in me.

Last but not least, I would like to thank my family for supporting me all the time. I thank my father Bilal Köpüklü and my mother Özgul Köpüklü for being the source of my motivation all the time. I give special thanks to my brother Hakan for being the best brother ever, and his wife Meltem for encouraging me all the time.

Munich, August 2022 Okan Köpüklü

Abstract

The amount of generated video data grows at ever-increasing rates dominating the majority of internet traffic. Therefore, the ability to automatically analyze video data effectively and efficiently is of great importance for numerous applications. The main goal of this thesis is to automatically capture visual and audio information from videos by using deep learning algorithms and keeping efficiency as a primary concern. Specifically, this thesis focuses on the high-level task of human activity recognition spanning the tasks of action recognition, hand gesture recognition, spatiotemporal action localization and audio-visual active speaker detection. Following the historical evolution of human activity recognition research, the contributions in this thesis are presented in three chapters.

Firstly, we present video analysis with frame-level features. We initially compare different spatiotemporal modeling techniques operating on frame-level features extracted by 2D CNNs. Next, in order to incorporate motion information to frame-level features, we present a data level fusion strategy, Motion Fused Frames, and demonstrate its advantages on hand gesture recognition task.

Secondly, we present video analysis with clip-level features. We initially propose a unified CNN architecture, You Only Watch Once (YOWO), benefiting clip-level features extracted by 3D CNNs for real-time spatiotemporal action localization task. However, 3D CNNs contain significantly more parameters and computational complexity compared to 2D CNNs. To address this drawback, we present families of resource efficient 3D CNN architectures for efficient video processing. Afterwards, for video-based Human Computer Interaction (HCI) applications that remain idle most of the time, we present a two-level hierarchical architecture, where a lightweight detector activates a heavyweight classifier only when it detects a hand gesture. Such an approach provides considerable savings on power and memory budget. Lastly, we address the challenges of operating 3D CNN architectures on video streaming applications without redundant computations by proposing Dissected 3D CNN architecture.

Thirdly, we present video analysis with audio-visual features. Audio and visual modalities contain complementary information for various video analysis applications. We focus on the audio-visual active speaker detection task and introduce several practical guidelines that result in the three-stage ASDNet architecture.

The content in this thesis covers the last decade of human activity recognition research that intensely benefited from the progress of deep learning algorithms, specifically CNNs. In order to enable further research on the field, we also publicly release three new datasets on gesture and action recognition.

Zusammenfassung

Die Menge an generierten Videodaten wächst mit höheren Raten und dominiert den Großteil des Internetverkehrs. Daher ist es für viele Anwendungen sehr wichtig, Videodaten effektiv und effizient automatisch zu analysieren. Das Hauptziel dieser Arbeit ist die automatische Erfassung visueller und akustischer Informationen aus Videos, mithilfe möglichst effizienter Deep-Learning-Algorithmen. Insbesondere konzentriert sich diese Arbeit auf die übergeordnete Aufgabe der menschlichen Aktivitätserkennung, welche Aktionserkennung, Handgestenerkennung, raumzeitliche Aktionslokalisierung und audiovisuelle Aktivsprechererkennung umfasst. Nach der historischen Entwicklung der Forschung zur Erkennung menschlicher Aktivitäten werden die Beiträge dieser Arbeit in drei Kapiteln präsentiert.

Zuerst stellen wir die Videoanalyse mit Frame-Level-Features vor. Wir vergleichen zunächst verschiedene raumzeitliche Modellierungstechniken, die auf Frame-Level-Fea-tures arbeiten, die mithilfe von 2D-CNNs extrahiert wurden. Um Bewegungsinformationen in Frame-Level-Features zu integrieren, präsentie-ren wir als Nächstes eine Fusionsstrategie auf Datenebene, Motion Fused Frames, und demonstrieren deren Vorteile bei der Handgestenerkennung.

Zweitens präsentieren wir die Videoanalyse mit Clip-Level-Features. Wir schlagen zunächst eine vereinheitlichte CNN-Architektur vor, You Only Watch Once (YOWO), die von 3D-CNNs extrahierte Clip-Level-Features für raumzeitliche Aktionslokalisierung in Echtzeit nutzt. 3D-CNNs enthalten jedoch im Vergleich zu 2D-CNNs deutlich mehr Parameter und Rechenkomplexität. Um diesen Nachteil zu beheben, präsentieren wir Familien von ressourceneffizienten 3D-CNN-Architekturen für eine effiziente Videoverarbeitung. Anschließend präsentieren wir für videobasierte Human Computer Interaction (HCI)-Anwen-dungen, die die meiste Zeit im Leerlauf bleiben, eine zweistufige hierarchische Architektur, bei der ein schlanker Detektor einen aufwendigen Klassifikator nur dann aktiviert, wenn er eine Handbewegung erkennt. Ein solcher Ansatz bietet beträchtliche Einsparungen bei Energie und Speicherbudget. Zuletzt stellen wir die Dissected 3D-CNN-Architektur vor, die noch effizienter ist, indem sie die beim Betrieb von 3D-CNN-Architekturen in Video-Streaming-Anwendungen redundanten Berechnungen vermeiden.

Drittens präsentieren wir Videoanalysen mit audiovisuellen Features. Akustische und visuelle Modalitäten enthalten sich gegenseitig ergänzende Informationen für verschiedene Videoanalyseanwendungen. Wir konzentrieren uns auf die Aufgabe der audiovisuellen aktiven Sprechererkennung und stellen mehrere praktische Richtlinien vor, die zur dreistufigen ASDNet-Architektur führen.

Diese Dissertation umfasst das letzte Jahrzehnt der Forschung zur Erkennung menschlicher Aktivitäten, welche stark vom Fortschritt des maschinellen Lernens,

insbesondere von CNNs in tiefen neuralen Netzen, profitiert hat. Um weitere Forschungen auf diesem Gebiet zu ermöglichen, veröffentlichen wir außerdem drei neue Datensätze zur Gesten- und Aktionserkennung.

Contents

Chapter 1

Introduction

As the saying goes, a picture is worth a thousand words. By nature, humans are visual creatures. Since the beginning of time, people have communicated with cave carvings to tell stories and record history, pictograms to transmit ideas, paintings to express feelings. How about videos? Considering the fact that a one-minute video contains more than a thousand pictures, a video can convey a massive amount of information.

Research on computer vision has been critical considering the increasingly growing amount of visual data. Video data specifically is being generated at ever-increasing rates dominating the internet traffic. Cisco predicts that internet video traffic will constitute 82% of all internet traffic (both consumer and business) by 2022, up from 75% in 2017 [1]. It is also predicted that close to 1 million minutes of video will traverse the internet at each second for various applications such as gaming, surveillance, Virtual Reality (VR), Augmented Reality (AR), Video-on-Demand (VoD), etc. The majority of created video content is centered around humans. Human-centric video perception is all about analyzing video content and understanding all possible human-related information within it, such as estimating human pose, estimating 3D body shape, recognizing a hand gesture, localization of an action, re-identification of a person from walking style, recognizing a face, etc.

With the vast amount of video data available, there is a continual need for powerful video perception algorithms. However, video applications run on a diverse set of devices having limitations on power, compute and memory. In addition, video data needs to be processed in real-time for most of these applications. Therefore, developed video perception algorithms need to be efficient in order to be applicable to a broader set of hardware.

This thesis addresses deep learning based human activity recognition by keeping efficiency as a primary concern. We investigate human activity recognition by specifically focusing on video analysis with frame-level features (Chapter 3), video analysis with clip-level features (Chapter 4) and audio-visual video analysis (Chapter 5). Fundamentals of deep learning such as how convolution operation works, optimizers, or loss functions are excluded from the scope of this thesis. Curious readers can refer to [1] for an introduction to the fundamentals of deep learning.

This chapter is organized as follows: Section 1.1 presents the goals of this thesis. Section 1.2 introduces the tackled challenges. Section 1.3 presents the contributions produced during this thesis. Finally, Section 1.4 explains the organization of the thesis.

[1] Cisco Visual Networking Index: Forecast and Trends, 2017–2022, White Paper.

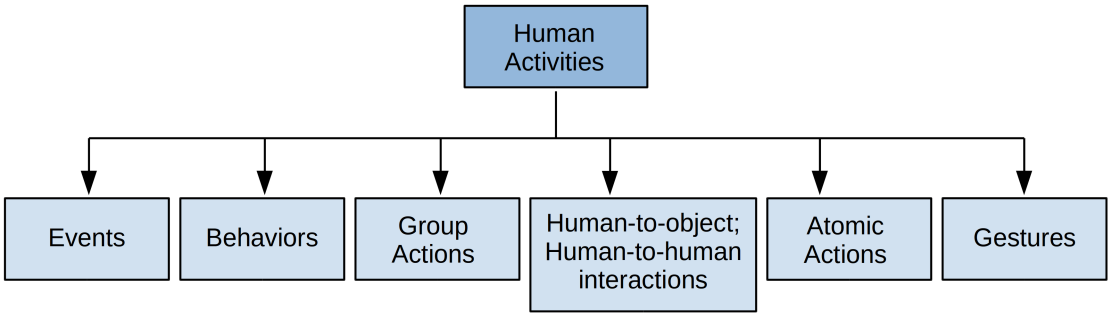


Figure 1.1: Decomposition of human activities depending on their complexity level. Adapted from [2].

1.1 Goals

The main goal of this thesis is to automatically capture audio and visual information from videos by using deep learning algorithms and keeping efficiency as a primary concern. Specifically, this thesis focuses on the high-level task of human action recognition. According to [2], human activities are grouped under six categories depending on their complexity level: (i) gestures, (ii) atomic actions, (iii) human-to-object or human-to-human interactions, (iv) group actions, (v) behaviors, and (vi) events. These categories are visualized in Fig. 1.1. We must note that these categories are not mutually exclusive. For example, human-human and human-object interactions can also be atomic actions, as in the case of the AVA dataset [3]. In this thesis, we span the majority of these categories by mostly working on action recognition, hand gesture recognition, spatiotemporal action localization, and audio-visual active speaker detection tasks. Fig. 1.2 shows some examples of human activities from the datasets which are used within the scope of this thesis.

Following the historical evolution of human activity recognition research, we have grouped our contributions in three chapters. Firstly, we focus on video analysis with frame-level features. There are in general two drawbacks of this approach: (i) Each extracted feature represents only the corresponding frame, hence cannot contain the pixel-level motion information between consecutive frames, which is critical for motion intensive actions and gestures. (ii) There needs to be a spatiotemporal modeling mechanism to make reasoning on the extracted frame-level features. We aim to compare various spatiotemporal modeling techniques in Section 3.2. Moreover, Section 3.3 aims to incorporate motion information to frame-level features. It must be noted that frame-level feature extraction is efficient since extracted features can be cached and used for future timestamps in the applied spatiotemporal modeling technique. Moreover, for the reason that the consecutive frames mostly contain very similar information, sparse sampling can be applied to reduce computational complexity even further.

Secondly, we focus on video analysis with clip-level features. Our initial goal is to show how clip-level features extracted by 3D CNNs can be used for real-time

Figure 1.2: Example activities with different complexity levels. Samples for atomic actions and human-object interactions are taken from AVA dataset [3]. Samples for gestures are from Jester dataset [4]. Samples for behaviors, human-human interactions are from Kinetics dataset [5]. Samples for sport events are from UCF101 dataset [6].

spatiotemporal action localization task, which is explained in Section 4.2. However, the main drawback of extracting clip-level features is that it requires a lot more parameters and computation for the used deep learning models compared to frame-level feature extraction. Motivated with this, our second goal has been creating resource efficient 3D Convolutional Neural Network (CNN) architectures in Section 4.3. Although these architectures reduce complexity and the number of parameters considerably, the complexity can still be further reduced by deactivating these architectures when they are not needed. With this motivation, we aim to investigate a two-level hierarchical framework that consists of a lightweight detector and heavyweight classifier for the task of hand gesture recognition as described in Section 4.4. The architecture proposed in Section 4.4 is also designed with the motivation to address the challenges of online recognition of hand gestures. However, it cannot be used for the tasks where the heavyweight classifier needs to be always kept active. For such cases, many works opt to extract clip-level features with a sliding window approach, either with a small temporal stride or larger stride. In the former case, there is severe resource waste due to reprocessing frames in the overlapping regions, which are already processed in the previous timestamps. In the latter case, there is an information loss since relations between some of the frames are not exploited. Motivated by these drawbacks, Section 4.5 addresses these drawbacks and aims to reduce computation by proposing a new 3D CNN architecture for online video processing applications.

Thirdly, we focus on audio-visual video analysis. Audio is often overlooked for video understanding tasks. However, it can provide complementary and discriminative information to the visually captured features. For the tasks of speech recognition, emotion recognition, active speaker detection, visual and auditory cues are equally

Figure 1.3: Example variations in video data. The top row illustrates variations in terms of human subjects and camera views. The bottom row illustrates variations in terms of video modality (from left to right: RGB, joint, depth, infrared). Examples are taken from the NTU RGB+D dataset [7].

important and should be taken into account for the creation of an effective architecture. Section 5.2 aims to address the task of Audio-Visual (AV) active speaker detection in the wild with the objective to provide several guidelines to create an effective detection architecture.

1.2 Challenges

Humans recognize actions in videos effortlessly, even in very complex scenarios. However, this is still a very difficult task for machines. Videos are perceived by machines as incoming video frames each consisting of a set of pixels and corresponding audio signal represented by changing level of electrical voltage. Therefore, recognizing human activities from videos requires translation of this data into meaningful representations. In recent years, deep learning models, particularly CNNs, have achieved astounding success on this task. Nonetheless, human activity recognition from videos is a popular research area since the following challenges still exist.

1.2.1 Data Challenges

Scarcity of annotated data. Excellent performance of CNNs in object detection and classification tasks has created an increasing trend to apply them also other computer vision areas including video analysis. This, however, comes at an additional cost of acquiring sufficient annotated data, which is indispensable for the successful training of CNN models. Despite the recent availability of large-scale video datasets such as Sports-1M [8], ActivityNet [9], Kinetics [5], AVA [3], MiT [10], CNN architectures still shows improvement with additional new data. However, the manual annotation of very large-scale datasets requires a great amount of time investment, sometimes also domain

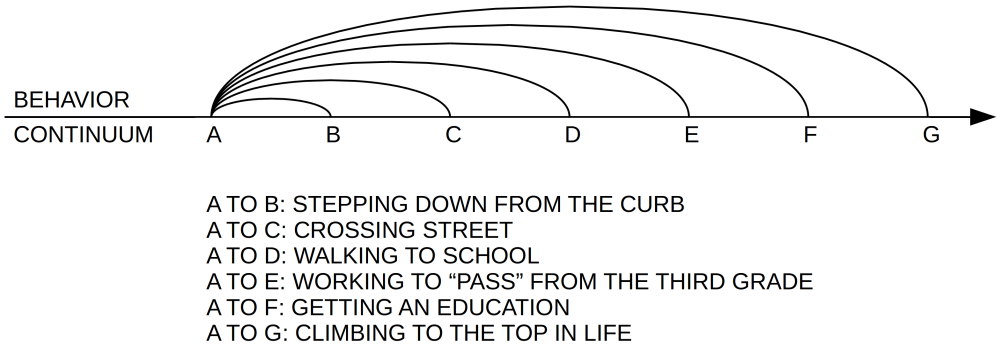


Figure 1.4: Hierarchical nature of an activity in terms of temporal granularity. Adapted from [11].

expertise. Moreover, manual annotation will always be prone to human errors. These challenges created a motivation in the research community to move towards active learning, semi-supervised learning and unsupervised learning.

Variation in video data. A common challenge in all deep learning applications is that the trained model needs to generalize to all variations of data. By nature, visual data contain several variations such as illumination conditions, different viewpoints, occlusion, deformations, difficult background conditions, and intra-class variations. Moreover, visual data might be captured via various sensors and be provided in different modalities such as RGB, depth, infrared and thermal. For example, Fig. 1.3 illustrates variations of different subjects and viewpoints in the top row and different modalities in the bottom row. Regarding human activities, the temporal dimension of the data also creates variations such as performing an action with different speeds. Similarly, the audio component of videos also contains several variations such as sampling frequency, different reverberation characteristics of the recording environment, and different accents or talking styles of actors. Since CNNs are data-driven architectures and their performance is highly correlated with the quality and quantity of the used dataset, the data curation for the given task needs to be exercised correctly without being biased to any specific variation. Although data augmentation techniques provide some variations on data, a proper dataset should contain data with all possible variations.

Taxonomy of human actions. Human actions can be defined based on the different temporal granularity and context. For the first, a person's action at a time can be defined differently according to the used temporal level. For example, the hierarchical nature of an activity in terms of temporal granularity is shown in Fig. 1.4. At the duration of 'A to B' in Fig. 1.4, the action of a person can be attributed to any six activities. For the latter, an action defined with a verb or phrase might lead to different human movements, poses and interactions. Fig. 1.5 shows the example of *riding* action under

riding in car | riding a motorbike | riding a horse | riding a bicycle

Figure 1.5: Examples of riding action in different contexts are highlighted with purple rectangles. Samples are taken from AVA dataset [3].

four different contexts. Therefore, the definition of a taxonomy for human actions is a challenge that researchers are still trying to address [12, 13].

1.2.2 Implementation Challenges

Large and complex neural network architectures. In order to improve accuracy, the trend in the computer vision community is to build larger and more complex Deep Neural Network (DNN) architectures. When the AlexNet [14] won the ImageNet challenge in 2012, it only contained 60 million parameters and achieved 63.3% top-1 classification accuracy. The best performing architecture on ImageNet dataset is currently CoAtNet-7 architecture [15] with 2.44 billion parameters and achieves 90.88% top-1 accuracy. Such large architectures require bigger Graphical Processing Unit (GPU)s to fit in that very few people have possession of. Moreover, as the number of parameters increases, there is the risk of overfitting the architecture to the training set of the used dataset.

Platform-related constraints. Trained DNN architectures need to be deployed to a platform in order to be used. Therefore, the DNN architecture needs to be designed considering the memory, compute power, thermal properties of the target platform. Most of the time, additional pruning and quantization of the model are also required for the deployment.

1.2.3 Challenges at Online Operation

Video processing DNN architectures are mostly designed for offline applications in contrast to the fact that most of the video applications require online operation. Therefore, before designing a DNN architecture, the following challenges of online operation need to be taken into consideration.

Recognition of dynamic actions. The main challenge of the recognition of dynamic actions, specifically gestures, is that there is not any direct signal, as in audio, stating the start and the end of dynamic actions. Moreover, dynamic gestures mostly contain very similar preparation and retraction stages and differ only at their nucleus stages [16]. Therefore, it is very critical to capture the nucleus part of the gestures in order to recognize them successfully.

Single-time recognition. For video applications such as Human-Computer Interaction (HCI) systems, the single-time recognition of actions and gestures is crucial. For example, for a gesture-based infotainment system in a car, *swipe right* gesture should be recognized only once by the system to switch to the next song and not to the second next song.

Fast reaction time. For several video processing tasks requiring online operation, fast reaction time is critical in order to provide more engaging and effective user interaction. According to studies, the range of acceptable variation of delay between performing an action and receiving a visual feedback should be no more than 0.1 to 0.2 seconds [17]. Therefore, for these kinds of applications, DNN architectures should be designed to have low complexity for faster inference and dynamically update their predictions with each new coming video frame.

1.3 Contributions

In this section, we present our contributions as publications, software and dataset releases that have been created during the course of this PhD project. Seven of our publications are detailed in several sections of the main body of this thesis, whereas three publications will be detailed in the appendix.

1.3.1 Publications

The work during this PhD led to the following publications:

- **O. Köpüklü**, F. Herzog, and G. Rigoll. Comparative analysis of CNN-based spatiotemporal reasoning in videos. In *International Conference on Pattern Recognition*, 2021. [18] (Section 3.2)
- **O. Köpüklü**, N. Köse, and G. Rigoll. Motion fused frames: Data level fusion strategy for hand gesture recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018. [19] (Section 3.3)
- **O. Köpüklü**, X. Wei, and G. Rigoll. You only watch once: A unified CNN architecture for real-time spatiotemporal action localization. In *arXiv*, 2021. [20] (Section 4.2)
- **O. Köpüklü**, N. Köse, A. Gunduz, and G. Rigoll. Resource efficient 3d convolutional neural networks. In *IEEE International Conference on Computer Vision Workshops*, 2019. [21] (Section 4.3)
- **O. Köpüklü**, A. Gunduz, N. Köse, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *IEEE International Conference on Automatic Face and Gesture Recognition*, 2019. [22] (Section 4.4)

- **O. Köpüklü**, S. Hörmann, F. Herzog, H. Cevikalp, and G. Rigoll. Dissected 3D CNNs: Temporal skip connections for efficient online video processing. In *Computer Vision and Image Understanding*, 2022. [23] (Section 4.5)
- **O. Köpüklü**, M. Taseska, and G. Rigoll. How to design a three-stage architecture for audio-visual active speaker detection in the wild." In *IEEE International Conference on Computer Vision*, 2021. [24] (Section 5.2)
- **O. Köpüklü**, Y. Rong, and G. Rigoll. Talking with your hands: Scaling hand gestures and recognition with CNNs. In *IEEE International Conference on Computer Vision Workshops*, 2019. [25] (Appendix A)
- **O. Köpüklü**, T. Ledwon, Y. Rong, N. Köse, and G. Rigoll. Drivermhg: A multi-modal dataset for dynamic recognition of driver micro hand gestures and a real-time recognition framework. In *IEEE International Conference on Automatic Face and Gesture Recognition*, 2020. [26] (Appendix B)
- **O. Köpüklü**, J. Zheng, H. Xu, and G. Rigoll. Driver anomaly detection: A dataset and contrastive learning approach. In *IEEE Winter Conference on Applications of Computer Vision*, 2021. [27] (Appendix C)
- **O. Köpüklü**, A. Gunduz, N. Köse, and G. Rigoll. Online dynamic hand gesture recognition including efficiency analysis. In *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2020. [28]
- **O. Köpüklü**, M. Babaee, S. Hörmann, and G. Rigoll. Convolutional neural networks with layer reuse. In *IEEE International Conference on Image Processing*, 2019. [29]
- **O. Köpüklü**, and G. Rigoll. Analysis on temporal dimension of inputs for 3d convolutional neural networks. In *IEEE International Conference on Image Processing, Applications and Systems*, 2018. [30]
- H. Cevikalp, B. Uzun, **O. Köpüklü**, and G. Ozturk. Deep compact polyhedral conic classifier for open and closed set recognition. In *Pattern Recognition*, 2021. [31]
- N. Köse, **O. Köpüklü**, A. Unnervik, and G. Rigoll. Real-time driver state monitoring using a CNN based spatio-temporal approach. In *IEEE International Conference on Intelligent Transportation Systems*, 2019. [32]
- M. Babaee, Y. Zhu, **O. Köpüklü**, S. Hörmann, and G. Rigoll. Gait energy image restoration using generative adversarial networks. In *IEEE International Conference on Image Processing*, 2019. [33]
- S. Hörmann, M. Knoche, M. Babaee, **O. Köpüklü**, and G. Rigoll. Outlier-robust neural aggregation network for video face identification. In *IEEE International Conference on Image Processing*, 2019. [34]

- M. Kayhan, **O. Köpüklü**, M. H. Sarhan, M. Yigitsoy, A. Eslami, and G. Rigoll. Deep attention based semi-supervised 2d-pose estimation for surgical instruments. In *International Conference on Pattern Recognition*, 2021. [35]
- F. Kälber, **O. Köpüklü**, N. Lehment, and G. Rigoll. U-net based zero-hour defect inspection of electronic components and semiconductors. In *International Conference on Computer Vision Theory and Applications*, 2021. [36]
- H. Saribas, H. Cevikalp, **O. Köpüklü**, and B. Uzun. TRAT: Tracking by attention using spatio-temporal features. In *Neurocomputing*, 2022. [37]
- Y. Feng, S. Wu, **O. Köpüklü**, X. Kang, and F. Tombari. Unsupervised monocular depth prediction for indoor continuous video streams. In *arXiv*, 2019. [38].

1.3.2 Software and Dataset Contributions

Software. The code for the seven sections and three appendixes of this thesis are publicly released:

- STModeling: The code and pretrained models of various spatiotemporal modeling mechanisms for video analysis are released as a part of this theses as described in [18] (Section 3.2).
  https://github.com/fubel/stmodeling
- Motion Fused Frames: The code and pretrained models of a data level fusion strategy for hand gesture recognition are released as a part of this theses as described in [19] (Section 3.3).
  https://github.com/okankop/MFF-pytorch
- YOWO: The code and pretrained models of a unified CNN architecture for real-time spatiotemporal action localization are released as a part of this theses as described in [20] (Section 4.2).
  https://github.com/wei-tim/YOWO
- Efficient-3DCNNs: The code and pretrained models of various resource efficient 3D convolutional neural network architectures are released as a part of this theses as described in [21] (Section 4.3).
  https://github.com/okankop/Efficient-3DCNNs
- Real-time-GesRec: The code and pretrained models of a two-level hierarchical convolutional neural network architecture for real-time hand gesture detection and classification are released as a part of this theses as described in [22] (Section 4.4).
  https://github.com/ahmetgunduz/Real-time-GesRec
- Dissected-3D-CNNs: The code and pretrained models of Dissected 3D CNN architecture are released as a part of this theses as described in [23] (Section 4.5).
  https://github.com/okankop/Dissected-3D-CNNs

- ASDNet: The code and pretrained models of a three-stage architecture for audio-visual active speaker detection in the wild are released as a part of this theses as described in [24] (Section 5.2).
  https://github.com/okankop/ASDNet
- SHGD: The code for scaling hand gestures and recognition with CNNs is released as a part of this theses as described in [25] (Appendix A).
  https://github.com/yaorong0921/GeScale
- DriverMHG: The code of a real-time recognition framework for dynamic recognition of driver micro hand gestures is released as a part of this theses as described in [26] (Appendix B).
  https://www.ei.tum.de/mmk/drivermhg/
- DAD: The code and pretrained models for driver anomaly detection are released as a part of this theses as described in [27] (Appendix C).
  https://github.com/okankop/Driver-Anomaly-Detection

In addition, the code for the following publications that are completed during this PhD but whose details are not covered within this thesis are also publicly released:

- LRUNet: The code and pretrained models of a layer reusing approach for CNNs are released, whose details are described in [29].
  https://github.com/okankop/CNN-layer-reuse
- DC-EPCC: The code of deep compact polyhedral conic classifier for open and closed set recognition is released, whose details are described in [31].
  https://github.com/bdrhn9/dc-epcc
- SSL-2D-Pose: The code of a deep attention based semi-supervised 2d-pose estimation for surgical instruments is released, whose details are described in [35].
  https://github.com/mertkayhan/SSL-2D-Pose

Datasets. Three datasets are publicly released during this PhD:

Scaled Hand Gesture Dataset

Scaled Hand Gesture Dataset (SHGD) is released with the publication of [25] (Appendix A) and publicly available at https://www.ei.tum.de/mmk/shgd/. SHGD contains 15 single hand gestures, each recorded for infrared (IR) and depth modalities. Each recording contains 15 gesture samples (one sample per class). There are in total 324 recordings from 27 distinct subjects in the dataset. Recordings of 8 subjects are reserved for testing, which makes 30% of the dataset. Every subject makes 12 video recordings using two hands under 6 different environments, which are designed for increasing the network robustness against different lighting conditions and background disturbances. Subjects perform gestures while observing a computer screen, where the gestures were prompted in random order. Videos are recorded at 45 frames per second

(fps) with a spatial resolution of 352×287 pixels. Each recording lasts around 33 seconds. Single static gestures in SHGD are referred as gesture-phonemes. In [25], we propose a methodology to scale hand gestures by forming them with predefined gesture-phonemes, and a CNN based framework to recognize hand gestures by learning only their constituents of gesture-phonemes.

Driver Micro Hand Gesture Dataset

The Driver Micro Hand Gesture (DriverMHG) dataset is released with the publication of [26] (Appendix B) and publicly available at https://www.ei.tum.de/mmk/drivermhg/. The driver micro gestures are performed within very short time intervals at spatially constrained areas. The dataset is recorded with the help of 25 volunteers (13 males and 12 females) using a simulator, which consists of a monitor, a Creative Blaster Senz3D camera featuring Intel RealSense SR300 technology, a Logitech G27 racing controller, whose wheel is replaced with a truck steering wheel and the OpenDS driving simulator software. The dataset is recorded in synchronized RGB, infrared and depth modalities with the resolution of 320 × 240 pixels and the frame rate of 30 fps. For each subject, there are in total 5 recordings each containing 42 gestures for 5 different gestures together with *other* and *none* gestures for each hand. Each recording of a subject was recorded under different lighting conditions. We randomly shuffled the order of the subjects and split the dataset by subject into training (72%) and testing (28%) sets.

Driver Anomaly Detection Dataset

The Anomaly Detection (DAD) dataset is released with the publication of [27] (Appendix C) and publicly available at https://www.ei.tum.de/mmk/dad/. For the dataset recording, 31 subjects are asked to drive in a computer game performing either *normal driving* or *anomalous driving*. Each subject belongs either to the training or to the test set. The training set contains recordings of 25 subjects and each subject has 6 normal driving and 8 anomalous driving video recordings. Each normal driving video lasts about 3.5 minutes and each anomalous driving video lasts about 30 seconds containing a different distracting action. In total, there are around 550 minutes recording for normal driving and 100 minutes recording of anomalous driving in the training set. The test set contains 6 subjects and each subject has 6 video recordings lasting around 3.5 minutes. Anomalous actions occur randomly during the test video recordings. Most importantly, there are 16 distracting actions in the test set that are not available in the training set. Because of these additional distracting actions, the networks need to be trained according to open set recognition task and distinguish normal driving no matter what the distracting action is. The complete test set consists of 88 minutes of recording for normal driving and 45 minutes of recording of anomalous driving. The test set constitutes the 17% of the complete DAD dataset. The size of the DAD dataset is around 95 GB.

1.4 Organization

This thesis contains six chapters including this introduction chapter and an appendix at the end. The rest of the thesis is organized as follows:

Chapter 2 introduces the literature review focusing on action recognition, gesture recognition, spatiotemporal action localization, audio-visual active speaker detection tasks, and corresponding available datasets.

Chapter 3 presents video analysis with frame-level features. First, motivations for using frame-level features are mentioned in Section 3.1 and possible drawbacks and advantages are elaborated. Then, in Section 3.2, different spatiotemporal modeling techniques, which are necessary in order to create reasoning on the extracted features, are analyzed and compared in terms of resource efficiency and performance on action and gesture recognition tasks. Lastly, we provide a data level fusion strategy in Section 3.3 for the hand gesture recognition task in order to incorporate motion information into the extracted frame-level features.

Chapter 4 presents video analysis with clip-level features. Similar to the previous chapter, we start with providing motivations for using clip-level features in Section 4.1 and elaborating on possible drawbacks and advantages. Then, Section 4.2 presents a unified spatiotemporal action localization architecture that makes use of clip-level features. To reduce the cost of extracting clip-level features, resource efficient 3D CNN architectures are introduced in Section 4.3. These architectures are inflated versions of some famous resource efficient 2D CNN architectures and evaluated for their capacities to learn complex classes, abilities to capture motion patterns, and applicability of transfer learning. Afterwards, in order to save computations for vision-based applications that the system remains idle for most of the time such as HCI systems, a two-level hierarchical framework is presented in Section 4.4. Lastly, in order to address the drawbacks of applying 3D CNNs at online video streaming applications, a new 3D CNN architecture is proposed in Section 4.5.

Chapter 5 presents video analysis with audio-visual features. It first provides motivations for using audio-visual features in Section 5.2.1 and elaborates on popular tasks that leverages audio and visual features jointly. Then, the task of Audio-Visual Active Speaker Detection (AV-ASD) is presented in Section 5.2.

Chapter 6 concludes our work and points out open problems and future work.

In the appendix, we provide our dataset contributions on action and gesture recognition tasks. Specifically, Appendix A introduces Scaled Hand Gesture Dataset (SHGD) where a scaling mechanism for hand gestures is explained and a recognition mechanism with CNNs is proposed. Appendix B introduces Driver Micro Hand Gesture (DriverMHG) dataset for dynamic recognition of driver micro hand gestures. Lastly, Appendix C introduces Driver Anomaly Detection (DAD) dataset for the recognition of safe and anomalous driving patterns of the drivers.

Chapter 2

Literature Review

This chapter provides a literature review on action recognition, gesture recognition, spatiotemporal action localization, audio-visual active speaker detection tasks, and popular datasets, which are the focus of this thesis.

2.1 Action Recognition

In this section, we survey the research on video-based action recognition. We first briefly present the representation based approaches before the deep learning era. Then we provide a comprehensive review of deep learning based methods on action recognition.

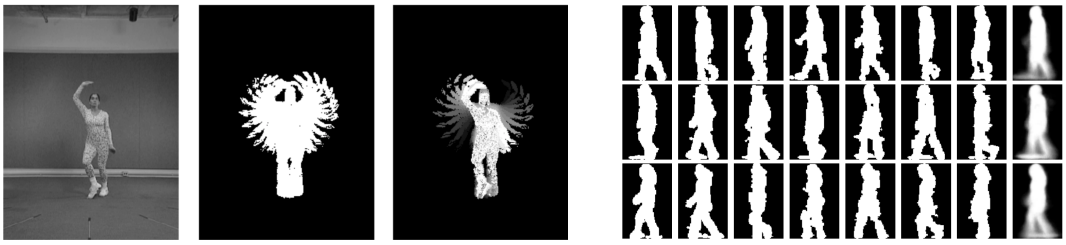
2.1.1 Hand-Crafted Features Based Methods

Representation of human actions in videos is a challenging problem since actions in videos can vary depending on the camera view, actor's pose, speed of motion, etc. A successful action representation method should take an input video and create a discriminative and representative feature vector. This section briefly reviews the hand-crafted feature representations under the categories of holistic and local representations; and action classifiers that turn these representations into class labels. We refer readers to [39, 40] for detailed surveys covering hand-crafted feature representations based action recognition.

2.1.1.1 Holistic Representations

Holistic representations refer to feature vectors that contain the information of an entire human subject including human body pose and movements. In [41], in order to encode dynamic human movements into static images, Motion Energy Image (MEI) and Motion History Image (MHI) are proposed. MEI is a binary image sequence indicating where the motion is occurring, whereas MHI shows how the motion is happening by allocating higher intensities to more recent movements. These two methods are applied to silhouette images as shown in Fig. 2.1 (left). A similar approach is applied for gait recognition in [42] by proposing Gait Energy Image (GEI). Different from MEI and MHI, GEI is calculated by averaging the binary silhouette sequence of a complete gait cycle of a person as shown in Fig. 2.1 (right).

In order to create view invariant and more robust representations, MEI and MHI are extended to 3D volumes by [43] and [44], respectively. [43] represent actions using



key frame, MEI and MHI gait cycle and GEI

Figure 2.1: (left) Examples of a key frame in a video, corresponding motion energy image (MEI) and motion history image (MHI) [41]. (right) Examples of gait cycles and corresponding gair enery image (GEI) at the end [42].

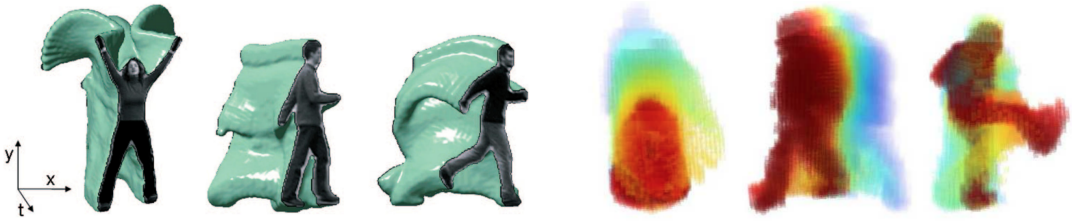


Figure 2.2: (left) Examples of spatiotemporal evolutions of jumping-jack, walk and run actions [43]. (right) Examples of motion history volume (MHV) for sit down, walk and kick actions [44].

silhouette images to construct space-time volumes as shown in Fig. 2.2 (left). [44] uses motion history volume (MHV) for representing actions as shown in Fig. 2.2 (right).

As an alternative approach to global action representation, motion information can also be computed using optical flow algorithms [45, 46, 47, 48]. Optical flow represents the apparent motion of objects, surfaces, and edges between two consecutive frames of a video. In [49], the horizontal and vertical axes of optical flow are half-wave rectified into four non-negative channels, which are then used in a nearest-neighbor querying framework to classify actions.

Although holistic representations have been popular between the middle of 1990s and early 2000s, there has been a shift towards local representations based action recognition, which will be explained in the next section.

2.1.1.2 Local Representations

Local representations refer to feature vectors that contain the salient motion information of local regions of a human subject. The works based on local representations widely used either Space-Time Interest Points (STIPs) [50, 51, 52] or motion trajectories [53, 54, 55, 56].

STIPs [50] has been the pioneering work on local representations based action recognition. In [50], Harris corner detector [57] has been extended to 3D Harris detector, which identifies points in space-time dimensions with large spatial variations

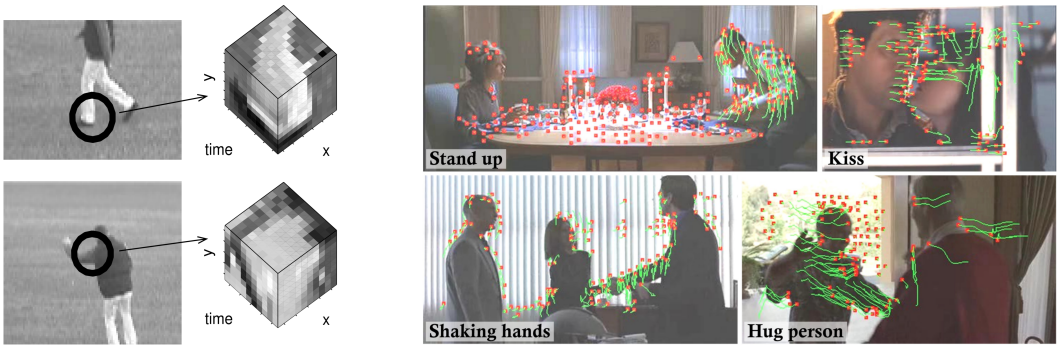


Figure 2.3: (left) Examples of local space-time neighborhoods for corresponding space-time interest points [63]. (right) Example trajectories of the SIFT salient points for four actions. [64].

and erratic motions. Cuboids are used from pixels around STIPs, as shown in Fig. 2.3 (left) Similarly, in [58], the Hessian detector is extended to its 3D version for action recognition. There have been other STIPs detectors that have extended their 2D counterparts to histograms of 3D gradient orientations (HOG3D) [59] and local trinary patterns [60]. In [52], Gabor filters are used to detect STIPs. In order to deal with the cases where STIPs are too rare in the video, [51] proposes to separate spatial and temporal filtering. Optical flow and gradients are also used to describe motion and apperance information. For instance, for a set of found interest points between two frames, computed optical flow is aggregated in histograms, named histograms of flow (HOF) in [61] that is later combined with histograms of oriented gradient (HOG) [62] to recognize complex human actions.

The major drawback of STIPs is that they cannot capture long-term temporal information. Tracking these interest points over long time duration, called feature trajectories, is the straightforward solution to this problem [65, 53, 64]. In [53], trajectories are extracted from interest points with KLT tracker [45]. [64] proposes to use SIFT descriptors [66] around interest points and use Markov chaining to determine feature trajectories, as shown in Fig. 2.3 (right). For the creation of trajectories, multiple features can be aggregated as in [56, 55, 67] that use Trajectory [68], HOF [61], HOG [62] and motion boundary histogram (MBH) [69] features.

2.1.1.3 Classification

Once the representations for the actions are computed, there remains only the classification problem. In other words, correct action labels need to be assigned to a given video input based on its extracted representation. Direct approaches have been actively used to recognize actions using off-the-shelf classifiers such as k-nearest neighbor (k-NN) [43, 63] or support vector machines (SVMs) [64, 56, 55, 67]. However, these approaches accept fixed-size input feature vectors and cannot function with sets of varying size feature vectors. Therefore, an aggregation mechanism is required to transform local descriptors into fixed-size, discriminative descriptors. Aggregation

methods of Bag-of-Visual Words (BoV) [70] and Fisher Vector (FV) encoding [71] have been actively used to aggregate features as in [72, 61, 43] and [73, 74], respectively. There are also sequential approaches that aim to capture temporal evolution of actions such as hidden Markov models (HMMs) [75] and conditional random fields (CRFs) [76] as in [77] and [78], respectively.

Although research on action recognition was dominated by hand-crafted features based approaches till the beginning of 2010s, recent research is overwhelmingly dominated by deep learning based approaches that will be discussed in the next section.

2.1.2 Deep Learning Based Methods

Ever since AlexNet [14] won the ImageNet Challenge (ILSVRC 2012 [79]), CNNs have dominated the majority of the computer vision tasks such as super-resolution [80], image denoising [81], and classification [82]. The success of AlexNet was a breakthrough for vision-based recognition tasks and is the point in history where the interest in deep learning increased rapidly [83]. Today, vision-based recognition approaches are dominated by the use of CNNs. After their success on image recognition, they have been explored also for video analysis tasks. A lot of work has proven that deep CNNs are capable to handle action recognition [84, 85, 86]. In this section, we review deep learning based action recognition methods using frame-level features and clip-level features.

2.1.2.1 Methods Based on Frame-Level Features

Due to the success of 2D CNNs in static images, video analysis approaches initially applied also 2D CNNs. There were two main motivations for this: (i) There were plenty of 2D CNN architectures [14, 87, 88, 89, 90], and these architectures could be pretrained using the very large-scale ImageNet dataset [82]. (ii) There were not large enough action recognition dataset that can be used to train a deep CNN architecture from scratch. Accordingly, initial approaches opt to extract features with ImageNet pretrained 2D CNN architectures and apply a spatiotemporal (ST) modeling mechanism afterwards for action recognition.

The simple application of MLP for the input of concatenation of frame-level features already produces competitive results [19, 91]. In [92], Temporal Segment Network (TSN) is proposed that divides a video into uniformly divided segments and extracts frame-level features from a randomly selected frame within each segment. Then, these extracted features are fed to an MLP to modify their dimension to class number. A consensus method is applied over these features such as evenly averaging, maximum, and weighted averaging to get final class scores. Similarly, Temporal Relation Network (TRN) is proposed in [91], which keeps the order of the extracted features intact and tries to discover possible temporal relations at multiple time scales.

The major drawback of working with frame-level features is that the extracted features cannot capture motion information within consecutive video frames. In order to address this problem, a two-stream approach is proposed in [93]. In context stream, a single video

frame is fed to a 2D CNN, whereas on the temporal stream stacked optical flow frames are fed to a 2D CNN by modifying its initial convolutional layer. Extracted features are passed through MLP and softmax layers and averaged to get final class scores. This fusion scheme is called late fusion. In [94], different fusion schemes are investigated for the two-stream architectures. These methods rely on separately processing the spatial and temporal components of the video, which can be a disadvantage. To address the problem of capturing motion information, we propose Motion Fused Frames (MFFs) [19] that applies data level fusion strategy of optical flow and RGB modalities such that extracted frame-level features also contain motion information. These features are fed to a two-layer MLP to get final class scores. This approach has been explained in detail in Section 3.3. To facilitate information exchange among neighboring frames, a temporal shift module (TSM) is introduced in [95], which can be inserted into 2D CNNs to shift part of the channels along the temporal dimension. TSM is a dynamic architecture and a causal version of it can be created by applying only single-sided shifting operation.

Recurrent neural networks are a natural choice for processing varying length video sequences, and several modern architectures have been proposed for action recognition in videos. Long Short-Term Memory (LSTM) [96] has been used in various video understanding tasks. In [97], LSTM is employed after CNN-based feature extraction on the individual frames to learn spatiotemporal components and apply the architecture on the UCF101 dataset [6]. In [98], vanilla LSTM architecture is modified to learn spatiotemporal domains for action recognition using 3D skeleton data. Gated Recurrent Unit (GRU) [99] is a popular variant of LSTM architecture which is actively used in video recognition tasks such as [100]. There have been many other variants of LSTM architecture, which are summarized in [101]. Another recurrent method is the Differentiable RNN [102] generated by salient motion patterns in consecutive video frames.

Although LSTM structure is proven to be stable and powerful in modeling long range temporal relations in various studies [103], [104], it handles spatiotemporal data using only full connections where no spatial information is encoded. Convolutional LSTM (ConvLSTM) [105] addresses this problem by using convolutional structures in both the input-to-state and state-to-state transitions. ConvLSTM is first introduced for precipitation nowcasting task [105], and later used for many other applications such as video saliency prediction [106] and medical segmentation [107].

Fully Convolutional Networks (FCN) are another approach to use for spatiotemporal modeling of frame-level features. FCN is first proposed for the image segmentation task [108] and currently the majority of segmentation architectures are based on FCNs. Later on, FCN architectures have been used at many other tasks such as object detection [109], [110]. As the name implies, FCN is a neural network that only performs convolution operations that can be applied on top of frame-level features for action recognition.

Recently, Transformer architectures [111] are producing astounding results on natural language processing tasks, making them the de-facto choice. This success motivated vision community to use them for their applications such as image classification [112, 113], object detection [114] and video instance segmentation [115]. For action recognition, several Transformer-based architectures are proposed. In [116]

proposes to use a Transformer-based architecture on skeleton data for action recognition. [117] uses a Transformer architecture to model person-specific contextual cues for action recognition and localization. TimeSFormer [118] and ViViT [119] are also Transformer-based architectures that treats video as a sequence of patches extracted from patches. Multiscale Vision Transformers (MViT) is proposed in [120], which aims to connect the of multiscale feature hierarchies with the Transformer model. Most relevant to action recognition based on frame-level features is the Video Transformer Network (VTN) architecture that uses a Transformer encoder, specifically Longformer [121], in order to learn temporal relationships between frame-level features [122].

In Section 3.2, we analyze and compare various techniques for ST modeling of the features extracted by a 2D CNN from sparsely sampled frames of action and gesture videos. Although frame-level processing of videos provides several advantages, their performance lags behind approaches that apply clip-level video processing, specifically using 3D CNN architectures, which will be reviewed next.

2.1.2.2 Methods Based on Clip-Level Features

Capturing motion information with optical flow modality is computationally expensive. A video is a 3D tensor with two spatial and one temporal dimensions, and motion information lies between the video frames over the temporal dimension. Therefore, a natural choice to extract motion information is to operate convolutions on space and time, also known as 3D convolutions. 3D CNN architectures are proposed for the first time in [123]. However, the proposed architecture was shallow containing only two 3D convolution layers.

3D CNNs contain significantly more parameters compared to their 2D counterparts making them more challenging to train and prone to overfitting. With the availability of large-scale datasets such as Sports-1M [8], ActivityNet [9], Kinetics [5, 124, 125], AVA [3], MiT [10], HACS [126], etc., the overfitting problem has been resolved. Ever since then, there have been plenty of 3D CNN architectures to achieve better accuracies at video classification tasks such as C3D [85], I3D [127], R(2+1)D [128], P3D [129], SlowFast [130], etc. The effect of dataset size is investigated in [131] together with the performance of the 3D versions of widely-used architectures such as ResNet [88], DenseNet [89], ResNext [90]. In Section 4.3, we investigate the 3D versions of popular resource efficient architectures for video classification tasks [21]. The SlowFast architecture in [130] explores the trade-offs of different spatial, temporal, and channel resolutions in the Slow and Fast pathway of the architecture. X3D is proposed in [132], which is a spatiotemporal architecture expanded from a tiny spatial network by multiple axes in space, time, width and depth in order to ensure good computation and accuracy trade-off.

The spatiotemporal modeling mechanism reviewed in the previous part can still be applied on top of clip-level features in order to capture long-term temporal information. For example, [133] proposes an architecture that uses a 3D CNN to extract clip features followed by an LSTM for online gesture recognition.

It must be noted that all the 3D CNN architectures mentioned above are designed for offline operation as they operate with a fixed number of input frames. Moreover, the number of floating point operations (FLOPs) is in the order of 10s-100s GFLOPs at inference time, which is too costly for online operation. To address this problem, in Section 4.5, we propose Dissected 3D CNNs [23], where the intermediate volumes of the network are dissected and propagated over depth (time) dimension for future calculations, substantially reducing the number of computations at online operation. Recently, [134] proposes Continual 3D CNNs that process videos frame-wise rather than clip-wise while benefiting from pretrained weights of popular 3D CNNs.

3D CNNs have dominated the research on video action recognition providing state-of-the-art results until recently. The current trend in action recognition is applying Transformer-based architectures [119, 120, 135], which achieve new state-of-the-art results.

2.2 Gesture Recognition

The requirements for video gesture recognition are very similar to action recognition. Therefore, literature review about action recognition in Section 2.1 is also valid for gesture recognition. In this section, we briefly review deep learning based gesture recognition.

Similar to action recognition, there has been several hand-crafted features based methods [136, 137] followed by deep learning based methods [133, 138, 139, 140, 22] for gesture recognition. In [141], an architecture with twelve streams focusing on hands is proposed. [133] focuses on online gesture recognition problems and proposes an architecture that is trained with connectionist temporal classification (CTC) [142] loss.

Video-based hand gesture recognition is mostly applied to HCI systems, where most of the time the system remains idle. To address this problem, in Section 4.4, we propose a two-level hierarchical architecture consisting of a lightweight gesture detector and a heavyweight gesture classifier [22]. Detector always remains active checking the availability of gestures, and if a gesture is detected the classifier gets activated to classify the performed gesture. We also address other challenges of online gesture recognition such as early and sing-time recognition

2.3 Spatiotemporal Action Localization

Spatiotemporal action localization task can be divided into object detection and action recognition subtasks. Literature review on action recognition is already provided in Section 2.1. Here, we will review deep learning based detection architectures and spatiotemporal action localization architectures.

For object detection in images, R-CNN series extract region proposals using selective search [143] or Region Proposal Network (RPN) [109] in the first stage and classify the objects in these potential regions in the second stage. Although Faster R-CNN [109]

achieves state-of-the-art results in object detection, it is hard to implement it for real-time tasks due to its time-consuming two-stage architecture. Meanwhile, YOLO [110] and SSD [144] aim to simplify this process to one stage and have outstanding real-time performance. For action localization in videos, due to the success of R-CNN series, most of the research approaches propose first detecting the humans in each frame and then linking these bounding boxes reasonably as action tubes [145, 146, 147]. Two-stream detectors introduce an additional stream on the base of the original classifier for optical flow modality [146, 148, 149]. Some other works produce clip tube proposals with 3D CNNs and achieve regression as well as classification on the corresponding 3D features [147, 148], thus region proposal is necessary for them. In a recent work [150], authors propose a 3D capsule network for video action detection which can jointly perform pixel-wise action segmentation along with action classification. However, it is too expensive in terms of computational complexity and number of parameters since it is a U-Net [151] based 3D CNN architecture.

In Section 4.2, we present YOWO [20], a unified CNN architecture for real-time spatiotemporal action localization in video streams. YOWO is a single-stage architecture with two branches to extract temporal and spatial information concurrently and predict bounding boxes and action probabilities directly from video clips in one evaluation.

2.4 Audio-Visual Active Speaker Detection

In this section, we present the literature review on beep learning based AV-ASD task in two parts: (i) audio-visual feature extraction in various applications, and (ii) contributions that address active speaker detection in the wild and its challenges.

2.4.1 Audio-visual feature extraction

Video. The literature review for deep learning based video feature extraction is provided in Section 2.1. Application of 3D CNNs [85, 127, 129, 128, 130, 131, 21, 132] for visual ASD task is favorable since 3D convolutions can inherently capture pixel-wise motion information within frames, which is critical to recognize mouth activity of active speakers.

Audio. A common approach to extract features in speech and audio research in different applications, is to use CNN and Recurrent Neural Network (RNN) with log-Mel or Short-Time Fourier Transform (STFT) spectrograms as inputs [152]. The popularity of these fixed transforms is due to their success in traditional speech and audio processing and the fact that they extract relevant information from first principles. Furthermore, the image-like configuration of the spectrograms allows employing network architectures well-known from computer vision applications. Particularly, in AV-ASD, this allows to use similar audio and video backbone architectures [153, 154].

Based on the interpretation of CNN as a data-driven filterbank, researchers have applied CNN directly on the audio waveforms to capture discriminative information for the task at hand [155, 156]. Such an approach in the context of AV-ASD has been used

for an audio backbone in [157]. However, these approaches need much more data and computational resources than the ones exploiting spectrograms. With the goal to exploit the best from both worlds, researchers have come up with learnable, but yet constrained transformations of raw audio data. Examples include Harmonic CNN used for music tagging, and the SincNet architecture proposed in [158]. The latter was successfully used in several audio applications [159, 160, 161].

Fusion. The extracted modality-specific features can be combined at data level [19], feature level [162] or decision level [93]. Data level fusion is not an option since audio and visual modalities come from different domains. Feature level fusion stands as the best option since fused features still need to be processed for context and temporal modeling afterwards.

2.4.2 Active speaker detection in the wild

Audio-visual active speaker detection is a specific case of source separation [163, 164], where audio and visual signals are leveraged jointly to assign a speech segment to its speaker. For this task, initial approaches [165, 166] use datasets collected in controlled environments. With the availability of AVA-ActiveSpeaker dataset [167], the research community was able to shift towards active speaker detection in the wild.

Audio-visual feature extraction is the first step in top-performing frameworks for active speaker detection task [167, 154, 153, 168, 169]. A two-backbone approach has established itself as a standard architecture due to its versatility [93]. With a good audio-visual feature extraction and RNN-based temporal modeling, the authors in [168] achieved competitive performance on the AVA-ActiveSpeaker dataset. Temporal modeling constitutes an integral part of recent active speaker detection pipelines [167, 154, 168, 169]. Often neglected is the context information that can be obtained by modeling inter-speaker relationships. Researchers have only recently proposed methods to exploit the context information[154, 153].

In Section 5.2, we present a new architecture called ASDNet [24] based on a series of controlled experiments, which would act as a practical guideline for audio-visual active speaker detection.

2.5 Datasets

Properly annotated, large-scale datasets are indispensable for effective utilization of deep CNN architectures. The effect of dataset size on the performance of 3D CNN architectures is investigated on [131].

UCF101 [6] and HMDB-51 [173] datasets are two popular video action recognition benchmarks, which are actively used at early 2010s. However, neither of them is large enough to train a deep 3D CNN architecture from scratch without overfitting. With the availability of Sports-1M dataset [8], deep 3D CNN architectures such as C3D [85] can be trained from scratch. Later on, several large-scale datasets [9, 5, 124, 125, 3, 10, 126] have been publicly made available. Kinetics dataset [5] has been specifically popular among

Dataset	Task	# Videos	# Classes	Frame Rate
Jester [4]	Gesture Rec.	148092	27	12 fps
nvGesture [133]	Gesture Rec.	1532	25	30 fps
ChaLearn LAP IsoGD [170]	Gesture Rec.	47933	249	10 fps
EgoGesture [171]	Gesture Rec.	24161	83	30 fps
Something-Something-V2 [172]	Action Rec.	220847	174	–
Kinetics-600 [124]	Action Rec.	495547	600	–
UCF101 [6]	Action Rec.	13320	101	25 fps
ActivityNet [9]	Action Rec.	19994	203	–
UCF101-24 [6]	ST Loc.	3207	24	25 fps
J-HMDB-21 [173]	ST Loc.	928	21	30 fps
AVA [3]	ST Loc.	430	80	25-30 fps
AVA-ActiveSpeaker [167]	AV-ASD	262	2	25-30 fps

Table 2.1: Summary of datasets that have been used in this thesis. The number of segmented gesture clips is reported for EgoGesture dataset.

Jester Dataset nvGesture Dataset

Figure 2.4: Examples from the Jester [4] and nvGesture [133] datasets.

action recognition researchers that most of the new architectures are benchmarked on this dataset.

In this thesis, several datasets are used for the tasks of action recognition, gesture recognition, spatiotemporal action localization, and audio-visual active speaker detection. The summary of the used datasets is shown in Table 2.1. For the rest of this section, we will give the details of each used dataset.

Jester.

Jester dataset is currently the largest available dataset, which has recently been available [4]. It is a large collection of densely labeled video clips that shows humans performing

ChaLearn LAP IsoGD Dataset EgoGesture Dataset

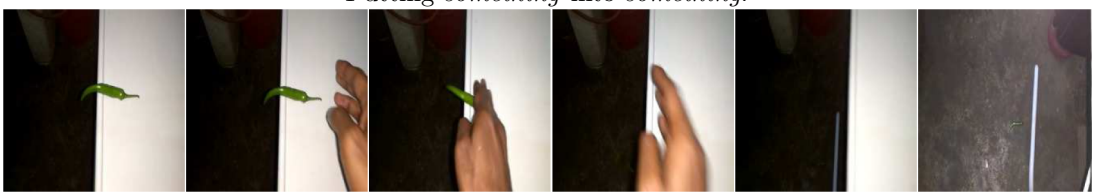
Figure 2.5: Examples from the ChaLearn LAP IsoGD [170] and EgoGesture [171] datasets.

predefined hand gestures in front of a laptop camera or webcam. The videos are recorded at 12 frames per second with the resolution of 100 pixels height and variable width. There are in total 148,092 gesture videos under 27 classes performed by a large number of crowd workers. The dataset is divided into three subsets: training set (118562 videos), validation set (14787 videos), and test set (14743 videos). Some examples from the Jester dataset are shown in Fig. 2.4. In this thesis, the Jester dataset is used in Sections 3.2, 3.3 and 4.3.

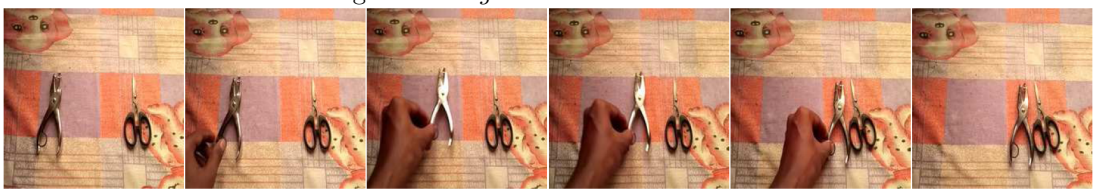
NVIDIA Dynamic Hand Gesture Dataset (nvGesture).
The nvGesture dataset [133] contains 25 hand gesture classes, each intended for human-computer interfaces and recorded by multiple sensors and viewpoints. There are 1532 weakly segmented videos in total, which are performed by 20 subjects at an indoor car simulator with both bright and dim artificial lighting. The dataset is randomly split by subjects into training (70%) and test (30%) sets, resulting in 1050 training and 482 test videos. Videos are captured by a SoftKinetic DS325 sensor with a frame rate of 30 fps. Since the gesture videos are weakly segmented, some parts of the videos do not contain any gesture. Some examples from the nvGesture dataset are shown in Fig. 2.4. In this thesis, the nvGesture dataset is used in Sections 3.3 and 4.4.

ChaLearn LAP IsoGD.
ChaLearn LAP IsoGD dataset [170] includes 47933 presegmented RGB-D gesture videos each representing one gesture only. There are 249 gesture classes performed by 21 different individuals. The dataset has been divided into three sub-datasets having 35878, 5784 and 6271 videos for training, validation and testing, respectively. Videos are captured by a Kinect device with a frame rate of 10 fps. Some examples from the ChaLearn dataset are shown in Fig. 2.5. In this thesis, the ChaLearn dataset is used in Section 3.3.

Putting *something* into *something*.



Pushing *something* so that it falls off the table.

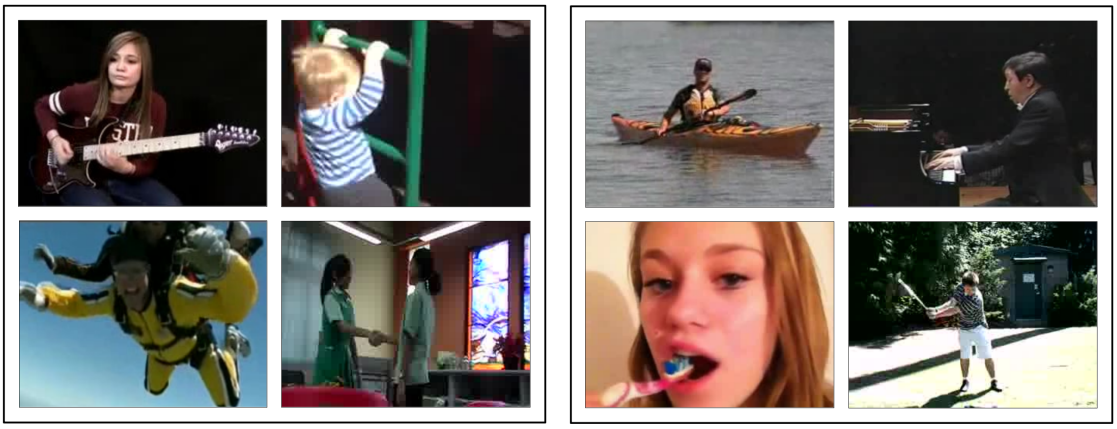


Moving *something* closer to *something*.

Figure 2.6: Examples from the Something-Something-V2 [172] dataset.

EgoGesture.
EgoGesture dataset is a recent multi-modal large-scale dataset for egocentric hand gesture recognition [171]. This dataset is created not only for segmented hand gesture classification, but also for online recognition of hand gestures in continuous data. There are 83 classes of static and dynamic gestures collected from 6 diverse indoor and outdoor scenes and from 50 distinct subjects. Dataset videos are recorded with the resolution of 640 $\times$ 480 and with the frame rate of 30 fps. The dataset splits are created by distinct subjects with a ratio of 3:1:1 resulting in 1239 training, 411 validation and 431 testing videos, having 14416, 4768 and 4977 gesture samples, respectively. Some examples from the EgoGesture dataset are shown in Fig. 2.5. In this thesis, the EgoGesture dataset is used in Section 4.4.

Something-Something-V2.
The Something-Something-V2 dataset is a collection of segmented video clips that show humans performing pre-defined basic actions with everyday objects [172]. It allows researchers to develop machine learning models capturing a fine-grained understanding of basic actions. The dataset consists of 220847 video clips under 174 classes, which is split into training, validation and test sets containing 168913, 24777 and 27157 videos, respectively. Some examples from the Something-Something-V2 dataset are shown in Fig. 2.6. In this thesis, the Something-Something-V2 dataset is used in Section 3.2.



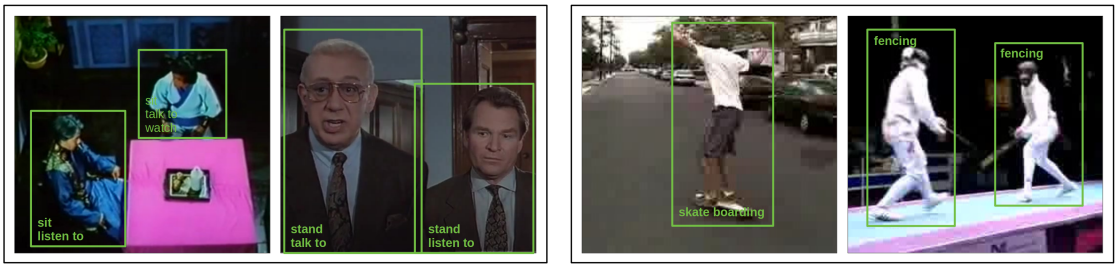
Kinetics Dataset UCF101 Dataset

Figure 2.7: Examples from the Kinetics [5] and UCF101 [6] datasets.

Kinetics.
Kinetics dataset is first introduced in [5] containing 400 human action classes, with at least 400 video clips for each action. Video clips are collected from YouTube videos and each lasts around 10 seconds with a variable resolution and frame rate. The actions in the Kinetics dataset are human-focused covering a wide range of classes including sports activities, playing musical instruments, human-object interactions and human-human interactions. The initial version of the dataset is referred as Kinetics-400 since it contains 400 distinct classes. Later on, versions of Kinetics-600 [124] and Kinetics-700 [125] are released containing 600 and 700 classes, respectively. Kinetics-400, Kinetics-600 and Kinetics-700 contain 306245, 495547 and 650317 video clips, respectively. Some examples from the Kinetics dataset are shown in Fig. 2.7. In this thesis, the Kinetics-600 dataset is used in Sections 4.3 and 4.5.

UCF101.
UCF101 [6] is a widely-used dataset for human action recognition consisting of 101 action classes. There are in total 13320 video clips each belonging to one of the five class categories: human-object interaction, body-motion only, human-human interaction, playing musical instruments and sports. Video clips are collected from YouTube and have the resolution of 320 $\times$ 240 and frame rate of 25 fps. Some examples from the UCF101 dataset are shown in Fig. 2.7. In this thesis, the UCF101 dataset is used in Section 4.3.

ActivityNet.
The ActivityNet dataset [9] is a large-scale video benchmark for human activity understanding. It contains 203 activity classes with an average of 137 untrimmed videos per class and 1.41 activity instances per video that are annotated with temporal boundaries. The videos are collected from YouTube and in total 849 hours long. Version 1.3 of the dataset, which is the one used in this thesis, contains 19994 videos in



AVA Dataset UCF101-24 Dataset

Figure 2.8: Examples from the AVA [3] and UCF101-24 [6] datasets.

total and divided into training, validation and test with the ratio of 2:1:1. The ground-truth annotations for the test set videos are not publicly available. In this thesis, the ActivityNet dataset is used in Section 4.5.

UCF101-24.
UCF101-24 is a subset of UCF101 dataset [6]. Out of 101 classes of UCF101 dataset, 24 action classes are annotated with spatiotemporal bounding boxes for 3,207 videos. Each video clip belongs to only a single action category. However, there might be multiple action instances in each video, which have the same class label but different spatial and temporal boundaries. Videos are not densely annotated. On average, each video contains 1.5 action instances covering around 70% of video duration. Some examples from the UCF101-24 dataset are shown in Fig. 2.8. In this thesis, the UCF101-24 dataset is used in Section 4.2.

J-HMDB-21.
J-HMDB-21 is a subset of the HMDB-51 dataset [173] and consists of 928 short videos with 21 action categories in daily life. Each video is well trimmed and has a single action instance across all the frames. We report our experimental results on the first split. In this thesis, the J-HMDB-21 dataset is used in Section 4.2.

AVA.
AVA is a video dataset of spatiotemporally localized Atomic Visual Actions (AVA). The AVA dataset contains 80 atomic visual actions, which are densely annotated for 430 15-minute video clips, where actions are localized in space and time. The annotations are person-centric and provided at the sampling frequency of 1 Hz. Every person in the sampled frame is annotated with person's bounding box and labeled with an action corresponding to person's pose, with person-object interaction actions (if any) person-person interaction classes (if any). This results in 1.58M action labels. AVA is a heavily imbalanced dataset having long-tailed distribution. According to ActivityNet evaluation guidelines, only the most frequent 60 action classes of the AVA dataset are used at evaluations. Some examples from the AVA dataset are shown in Fig. 2.8. In this thesis, the AVA dataset is used in Section 4.2.

Figure 2.9: Examples from the AVA-ActiveSpeaker dataset [167]. Green and red bounding boxes imply *speaking* and *not speaking* classes, respectively.

AVA-ActiveSpeaker.
The AVA-ActiveSpeaker dataset [167] is the first audio-visual active speaker dataset collected in the wild. It contains 262 15-minute videos from Hollywood movies, recorded at 25-30 fps, 120 of which are used for training, 33 for validation, and 109 for testing. The videos consist of 3.65 million human-labeled frames, where face crops belonging to the same speaker are aggregated to create face tracks, and each face crop is annotated with *speaking* or *not-speaking* label. This results in 38.5 hours of face tracks with the corresponding audio signal. The number of speakers in the videos is time-varying, and a significant portion of face crops has a resolution less than 100 pixels, making the dataset considerably challenging. Some examples from the AVA-ActiveSpeaker dataset are shown in Fig. 2.9. In this thesis, the AVA-ActiveSpeaker dataset is used in Section 5.2.

Chapter 3

Video Analysis With Frame-Level Features

This chapter presents video analysis with frame-level features. Specifically, we work on action and gesture recognition tasks on video streams that require temporal reasoning of the spatial content from different time instants, i.e., spatiotemporal (ST) modeling. In Section 3.2, we make a comparative analysis of different ST modeling techniques for action and gesture recognition tasks. Since CNNs are proved to be an effective tool as a feature extractor for static images, we apply ST modeling techniques on the features of static images from different time instants extracted by 2D CNNs. All techniques are trained end-to-end together with the CNN feature extraction part and evaluated on two publicly available benchmarks: The Jester and the Something-Something datasets. The Jester dataset contains various dynamic and static hand gestures, whereas the Something-Something dataset contains actions of human-object interactions. The common characteristic of these two benchmarks is that the designed architectures need to capture the full temporal content of videos in order to correctly classify actions and gestures.

Extraction of features from individual frames with 2D CNNs cannot capture motion information within consecutive video frames. Therefore, in Section 3.3, we present Motion Fused Frames (MFFs), a data level fusion strategy that is designed to fuse motion information into static images to represent the ST states of actions better.

3.1 Introduction

Deep learning has dominated computer vision research, spanning tasks such as image classification, object detection, action recognition, etc. Currently, nearly all state-of-the-art solutions for these tasks employ deep learning based architectures, specifically CNNs. Due to lack of large-scale video datasets, 2D CNNs that are pretrained on ImageNet dataset [82] have been actively used for video understanding tasks. With the availability of large-scale datasets, specifically Kinetics dataset [5], 3D CNNs have replaced 2D CNNs for video understanding tasks since 3D CNNs can capture the ST patterns in videos inherently without requiring additional mechanisms. However, their drawback is that the input size should always remain the same for 3D CNNs such as 16 or 32 frames, which makes them not suitable for capturing temporally varying actions. This is not a problem for activity recognition tasks for Kinetics [127] or UCF101 [6] datasets, as videos can be successfully classified using even very small snippets of the complete video. However, there are tasks where the designed architectures need to observe the complete

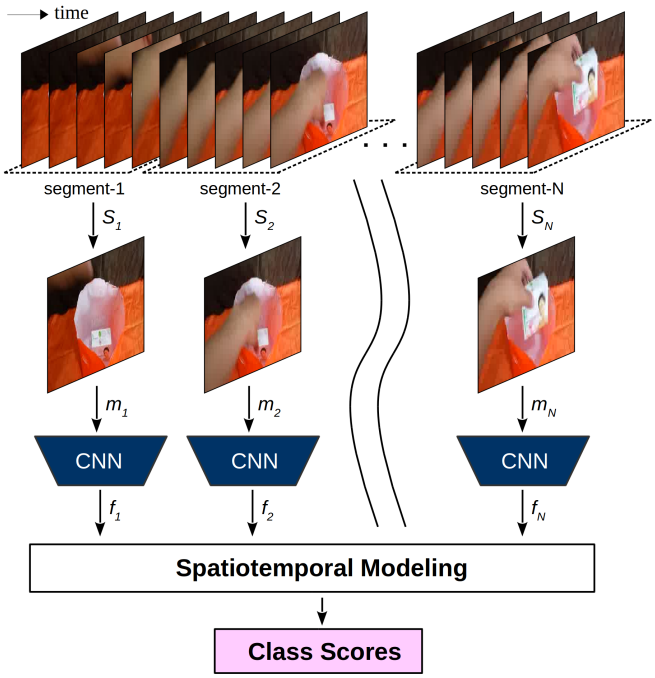


Figure 3.1: Frame-level video analysis architecture. One input video containing an action or gesture is divided into N segments. Afterwards, equidistant frames (m_1,m_2, .. m_N) are selected from the segments and fed to a 2D CNN for feature extraction. Extracted features are fed to a ST modeling block, which produces the final class score of the input video. In this example, action of *taking something from somewhere* is depicted, which is taken from the Something-Something-V2 dataset [172].

video at once in order to make correct predictions. For these tasks, 2D CNN based architectures are still useful as the complete videos can be sparsely sampled with the desired number of segments, and features of the selected frames can be extracted. Still, these architectures need an extra mechanism to provide ST modeling of the extracted features.

Section 3.2 aims to analyze and compare various techniques for ST modeling of the features extracted by a 2D CNN from sparsely sampled frames of action videos. Fig. 3.1 depicts the used ST modeling architecture. A complete action video is divided into a predefined number of segments. From each segment, a frame is selected (randomly in training and equidistant in testing) and fed into the 2D CNN to extract its features. In order to understand which type of action is performed, a ST modeling technique is used. Although analyzed techniques have been used in several works in the literature, there has not been any comparative analysis to highlight the advantages of each ST modeling technique. Section 3.2 tries to fill this gap by comparing each technique in terms of efficiency (i.e. number of parameters and floating point operations) and classification accuracy. Section 3.2 is based on our publication *Comparative Analysis of CNN-based Spatiotemporal Reasoning in Video* [18].

The main advantage of applying a video analysis method based on frame-level features, as in Fig. 3.1, is resource efficiency. Computed frame-level features can be cached in a queue to be used at the deployed ST modeling technique for online operation. Since the information within consecutive frames is nearly the same, sparse sampling can be applied. Moreover, resource efficient 2D CNN architectures [87, 174, 175, 176, 177] can be used for feature extraction. On the other hand, the major drawback of such an architecture is that it lacks capturing motion information within frames since every frame-level feature is extracted independently. Section 3.3 tries to resolve this setback by proposing a data level fusion strategy of optical flow and RGB modalities. Section 3.3 is based on our publication *Motion Fused Frames: Data Level Fusion Strategy for Hand Gesture Recognition* [19].

3.2 Spatiotemporal Modeling Mechanisms

In this section, we have analyzed several ST modeling techniques: Multi-Layer Perceptron (MLP) based techniques such as simple MLP, Temporal Relational Network (TRN) and Temporal Segment Network (TSN), Recurrent Neural Network (RNN) based techniques such as vanilla RNN, gated recurrent unit (GRU), long short-term memory (LSTM), bidirectional LSTM (B-LSTM) and convolutional LSTM (ConvLSTM) techniques, Transformer based techniques, and finally fully convolutional network (FCN) techniques. Fig. 3.1 illustrates the applied frame-level video analysis architecture. First, a video clip V that contains a complete action is divided into N segments. Each segment is represented as $S_n \in \mathbb{R}^{w \times h \times c \times m}$ of $m \geq 1$ sequential frames with 224×224 spatial resolution and $c = 3$ channels. RGB modality is used in all the trainings. Afterward, within segments, equidistant frames are selected and passed to a 2D CNN model for feature extraction. Extracted features are first pooled and transformed to a fixed size of 256 (except for TSN where features are transformed to *number-of-classes*) via a one-layer Multi-layer Perceptron (MLP) except for ConvLSTM and 3D-FCN techniques. For these two techniques, no pooling is applied at the feature extraction and the number of channels is transformed to 256 by using a 1×1 2D convolution layer.

For feature extraction, two different CNN models are used: (i) SqueezeNet [87] with simple bypass and (ii) Inception with Batch Normalization (BNInception) [178]. The reason to choose these models is that the performance of the investigated ST modeling techniques can be evaluated with a lightweight CNN feature extractor (SqueezeNet) and a relatively more complex and heavyweight CNN feature extractor (BNInception). In this way, *CNN-model-agnostic* performance of evaluated techniques can be observed.

Extracted features are finally fed to a ST modeling block, which produces the final class scores of the input video clip. Next, we are going to investigate different ST modeling techniques in detail that are used in this block.

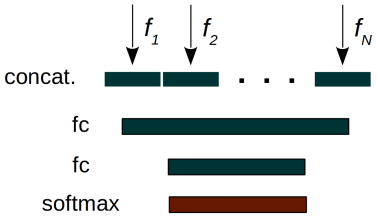


Figure 3.2: Simple MLP technique. Extracted features are concatenated keeping their order same to form N dimensional vector. This vector is fed to a 2-layer MLP to get final class scores.

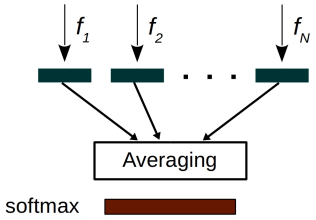


Figure 3.3: Temporal Segment Network (TSN) architecture. Extracted frame features are transformed to *Number-of-classes* dimension and averaged to get class conditional scores.

3.2.1 Multilayer Perceptron (MLP) Based Techniques

MLP based ST modeling techniques are simple but effective to incorporate temporal information. These techniques make use of MLPs once or multiple times. Extracted features are then fed to these MLP based ST modeling blocks keeping their order intact. The intuition is that MLPs can capture the temporal information of the sequence inherently without knowing that it is a sequence at all.

3.2.1.1 Simple MLP

As illustrated in Fig. 3.2, extracted features are concatenated preserving their order. Then, the concatenated single $N \times 256$ dimensional vector is fed to a 2-layer MLP with 512 and *Number-of-classes* dimensions. Finally, the output is fed to a softmax layer to get class conditional scores. This is a simple but effective approach. Combined with other modalities such as optical flow, infrared and depth, competitive results can be achieved [19].

3.2.1.2 Temporal Segment Network (TSN)

TSN aims to achieve long-range temporal structure modeling using sparse sampling strategy [92]. When the original paper was written, TSN achieved state-of-the-art performance on two activity recognition datasets, namely the UCF101 [6] dataset and the HMDB [173] dataset.

The original TSN architecture uses the optical flow and RGB modalities, as well as different consensus methods such as evenly averaging, maximum, and weighted averaging. Among them, evenly averaging achieved the best results in the original experiments. Therefore, we have also experimented with evenly averaging for RGB modality only.

The corresponding TSN approach is depicted in Fig. 3.3. Unlike other ST modeling techniques, the extracted frame features are transformed into a fixed size of *number-*

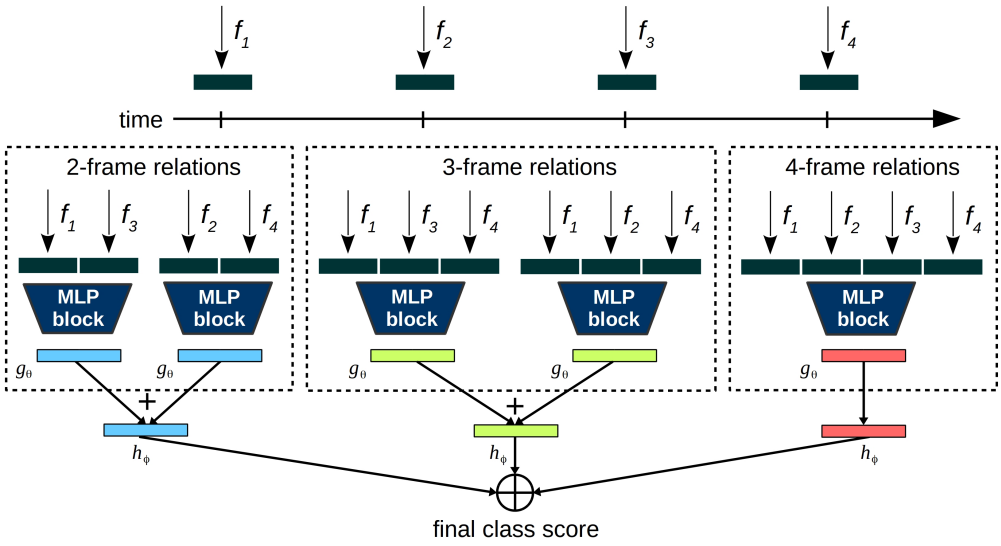


Figure 3.4: Illustration of Temporal Relation Networks. Features extracted from different segments of a video by a 2D CNN are fed into different frame relation modules. Only a subset of the 2-frame, 3-frame, and 4-frame relations are shown in this example (4 segments), as there are higher frame relations included according to the segment size.

of-classes instead of 256. Afterward, all extracted features are averaged and fed to a softmax layer to get class conditional scores.

Although TSN achieved state-of-the-art performance on UCF101 and HMDB benchmarks at the time, it achieves inferior performance in the Jester and Something-Something benchmarks. The reason is that averaging causes loss of temporal information. This does not create a huge problem for the UCF101 and HMDB benchmarks as temporal order is not critical for these. Correct classification can even be achieved using only one frame of the complete video. However, the Jester and Something-Something datasets require the incorporation of the complete video in order to infer correct class scores.

3.2.1.3 Temporal Relation Network (TRN)

TRNs [91] aim to discover possible temporal relations between observations at multiple time scales. The main inspiration for this work comes from the relational reasoning module for visual question answering [179]. The pairwise temporal relations (2-frame relations) on the observations of the video V are defined as follows:

$$T_2(V) = h_\phi \left(\sum_{i<j} g_\theta(f_i, f_j) \right), \tag{3.1}$$

where the input is the features of the n selected frames of the video $V = \{f_1, f_1, ..., f_n\}$, in which f_i represents the feature of the i^{th} frame segment extracted by a 2D CNN.

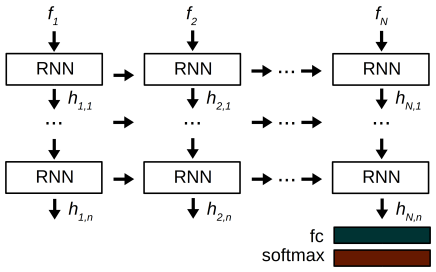


Figure 3.5: M-layered architecture of Recurrent Neural Networks.

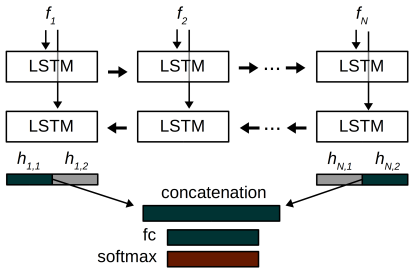


Figure 3.6: The data flow for Bidirectional LSTM architecture. .

Here, g_θ and h_ϕ represent the feature fusing functions, which are MLPs with parameters θ of size 256 and ϕ of size *Number-of-classes*, respectively. These two-frame temporal relations functions are further extended to higher frame relations, but the order of the segments should always be kept same in order to learn temporal relations inherently. Finally, all frame relations can be incorporated in order to get a single final output $MT_N(V) = T_2(V) + T_3(V) + ... + T_N(V)$, which is referred as multiscale TRN, where each T_d captures temporal relationships between features of d ordered frames. Fig. 3.4 depicts the overall TRN architecture.

3.2.2 Recurrent Neural Networks (RNN) Based Techniques

Recurrent neural networks (RNNs) consist of recurrently connected hidden layers which are capable of capturing temporal information. Furthermore, they allow the input and output sequences to vary in size. It is important to note that the hidden layer parameters do not depend on the time step but are shared across all RNN slices. The ability to keep information from previous time steps makes the hidden layer work like a memory. General M-layered RNN architecture is depicted in Fig. 3.5.

In our experiments, we use two different vanilla RNNs, based on the hyperbolic tangent activation function, and the rectified linear unit (ReLU) activation function, respectively. Vanilla RNN with hyperbolic tangent activation function can be described by following equations:

$$h_t = tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right), \tag{3.2}$$

$$y_t = W_{hy}h_t. \tag{3.3}$$

Generally, we feed the output of the last node to a fully connected layer to obtain a vector size of the number of classes in the dataset. We also proceed in the same manner for all other RNN types except for the *Bidirectional LSTM*.

The output gate decides what the next hidden state should be

3.2.2.1 Long Short-Term Memory (LSTM)

LSTMs [96] are recurrent neural networks consisting of an input gate, a forget gate, an output gate, a cell state, and a hidden state. The input gate i_t decides how much

the current x_t contributes to the overall output. The cell state c_t is responsible for remembering the previous state information, and also uses the results of the forget gate f_t, which decides how much of the previous cell state c_{t-1} flows into the current cell. As the name suggests, the forget gate can completely erase the previous state if necessary. Finally, the output gate o_t determines the next hidden state h_t using the current cell state c_t and previous hidden state h_{t-1}. Following the formulation in [104], the LSTM cell can be described by the following equations:

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i\right), \tag{3.4}$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f\right), \tag{3.5}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right), \tag{3.6}$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o\right), \tag{3.7}$$
$$h_t = o_t \circ tanh\left(c_t\right), \tag{3.8}$$

where '$\circ$' denotes the Hadamard product; W and b refer to weight matrices and bias vectors, respectively; σ and $tanh$ refer to sigmoid and hyperbolic tangent functions, respectively.

3.2.2.2 Gated Recurrent Units (GRU)

GRUs [99] are very similar to LSTMs and consist of two gates: an update gate z_t and a reset gate r_t. However, unlike LSTMs, GRUs do not have their own memory control mechanism. Instead, the entire hidden layer information is directed to the next time step. The advantage of GRUs compared to LSTMs is their simplicity in structure, which significantly reduces the number of parameters to be learned. GRU can be described by the following equations:

$$z_t = \sigma\left(W_{xz}x_t + W_{hz}h_{t-1} + b_z\right), \tag{3.9}$$
$$r_t = \sigma\left(W_{xr}x_t + W_{hr}h_{t-1} + b_r\right), \tag{3.10}$$
$$\tilde{h_t} = tanh\left(W_{xh}x_t + W_{hh}(r_t \circ h_{t-1}) + b_h\right), \tag{3.11}$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t}, \tag{3.12}$$

where '$\circ$' denotes the Hadamard product; $\tilde{h_t}$ and h_t represent the intermediate memory and output, respectively.

3.2.2.3 Bidirectional LSTM (BLSTM)

BLSTMs [180] are a special form of LSTMs, but are trained in both directions. The fully connected layer is obtained by concatenating two halved outputs $h_{1,1}$ and $h_{m,2}$, namely the first output of the positive time direction and the last output of the negative time direction. The data flow for BLSTM architecture is depicted in Fig. 3.6. We also investigate the effect of the hidden size by reducing it to half of the hidden size value we used for the other RNN-structures. This allows us to make meaningful comparisons with the latter. The reduction of the hidden layer size means that the vector size remains

unchanged before the last fully connected layer. Consequently, the same number of output neurons is used for the classification.

3.2.2.4 Convolutional LSTM (ConvLSTM)

The main drawback of conventional LSTM (also GRU and vanilla RNN) in handling spatiotemporal data is that input-to-state and state-to-state transitions are made by full connections, where no spatial information is encoded. To overcome this drawback, convolutional LSTM proposes to use convolutional structures for the mentioned transitions. The main equations of ConvLSTM are given as follows:

$$i_t = \sigma\left(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \circ c_{t-1} + b_i\right), \tag{3.13}$$
$$f_t = \sigma\left(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \circ c_{t-1} + b_f\right), \tag{3.14}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ tanh\left(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c\right), \tag{3.15}$$
$$o_t = \sigma\left(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \circ c_t + b_o\right), \tag{3.16}$$
$$h_t = o_t \circ tanh\left(c_t\right), \tag{3.17}$$

where '$*$' and '$\circ$' denote the convolution operator and Hadamard product, respectively. In order to make use of ConvLSTM technique for ST modeling, we have made some modifications. First, to keep spatial resolution, we have removed the final pooling layer of our feature extractor 2D CNN. Then, the output features are concatenated in time dimension forming a $D \times W \times H$ tensor. The last output of ConvLSTM is average pooled and fed to a final fully connected layer and a softmax layer to get class conditional scores.

3.2.3 Transformer Based Techniques

Transformer [111] architectures have dominated the natural language processing tasks due to their strength in capturing temporal relations with a self-attention mechanism. Motivated by the recent application of Transformer based architectures to vision tasks [112, 115, 116, 119, 122], we have also used a Transformer based ST modeling technique.

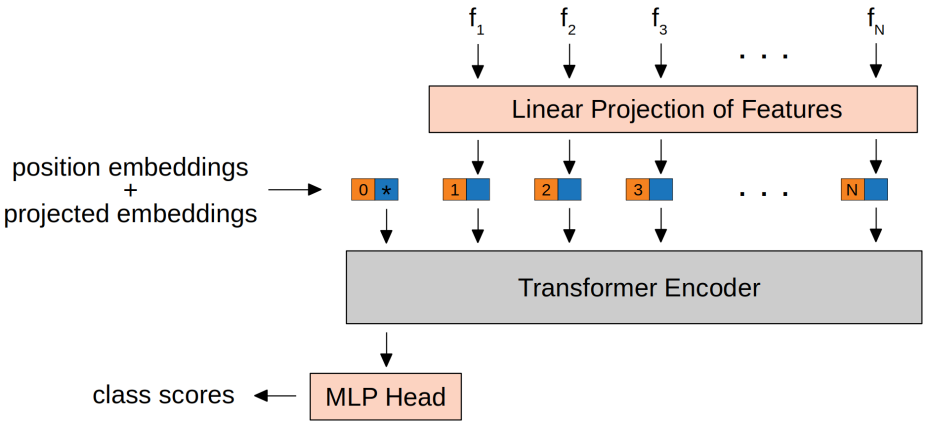


Figure 3.7: Illustration of the applied Transformer based ST modeling technique.

Layer/Stride	**Filter size**	**Output size**
Input		1×N×256
Conv1/s(1,2)	3×3	64×N×128
Conv2/s(1,2)	3×3	64×N×64
Conv3/s(1,2)	3×3	128×N×32
Conv4/s(1,2)	3×3	128×N×16
Conv5/s(1,2)	3×3	256×N×8
Conv6/s(1,1)	1×1	*NumCls*×N×8
AvgPool/s(1,1)	N×8	*NumCls*

Table 3.1: Details of 2D fully convolutional ST modeling architecture.

Fig. 3.7 illustrates the used Transformer based ST modeling approach. We first apply a linear projection layer, which is a simple linear layer, to project each frame-level feature from 256 to 768 dimensions. Following popular architectures [112, 122], we prepend a class token, which is a learnable embedding, in front of the sequence of the projected features. Afterwards, we add position embeddings to the projected features to provide positional information. Position embeddings are learnable 1D embeddings with the size of 1024. In the Transformer encoder, we have used the famous BERT [181] architecture with only 3 successive transformer layers, each having 12 attention heads, layer normalization (LN) [182], and MLP blocks. At the position where class embeddings are inserted, the output of the Transformer encoder is fed to a 2-layer MLP with 1024 and *Number-of-classes* dimensions.

3.2.4 Fully Convolutional Network (FCN) Based Techniques

As the name implies, all of the layers of a fully convolutional network are convolutional layers. FCNs do not contain any linear (fully connected) layers at the end, which is the typical use for the classification task. In order to utilize FCNs as a ST modeling technique, output features coming from the 2D CNNs can be concatenated over a dimension such that convolution operation can be performed over the concatenated tensor. If the features are pooled, concatenated features form a 2D tensor over which 2D convolutional layers can operate. If pooling is not applied at the feature extraction stage, concatenated features form a 2D tensor over which 3D convolutional layers can operate.

3.2.4.1 2D-FCN

The inputs to 2D-FCN are the concatenated feature vectors of each segment resulting in $N \times 256$ such that each row represents features from a segment. The input volume enters a series of 2D convolutions with stride $(1, 2)$, which keeps the temporal dimension (i.e. the number of segments) intact throughout convolution operations. The kernel size is set to 3×3 with the same padding for all convolutions. After applying five convolutions,

Layer/Stride	**Filter size**	**Output size**
Input		256×D×W×H
Conv1/s(2,1,1)	3×3×3	64×D/2×W×H
Conv2/s(2,1,1)	3×3×3	128×D/4×W×H
Conv3/s(2,1,1)	3×3×3	256×D/8×W×H
Conv4/s(1,1,1)	1×1×1	*NumCls*×D/8×W×H
AvgPool/s(1,1,1)	D/8×W×H	*NumCls*

Table 3.2: Details of 3D fully convolutional ST modeling architecture.

2D convolution with 1×1 kernel is applied where the number of channels equals the number of classes. Finally, average pooling with $N \times 8$ is applied to get class conditional scores. After each convolution, batch normalization and ReLU are applied. The details of the used 2D-FCN are given in Table 3.1.

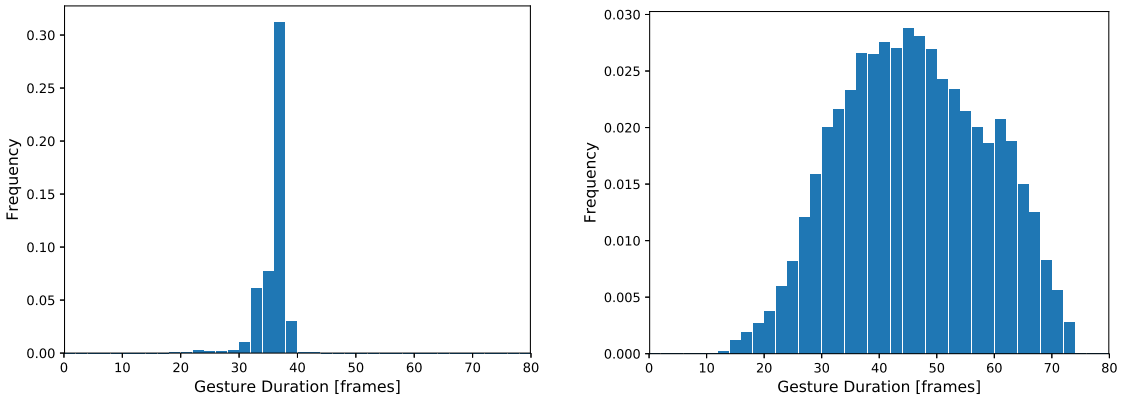
3.2.4.2 3D-FCN

In order to make use of spatial information, similar to ConvLSTM, we have not used a pooling layer at the end of feature extractor 2D CNN, and output features are concatenated in the depth dimension to create $D \times W \times H$ tensor. This tensor enters a series of 3D convolutions with stride $(2, 1, 1)$ in order to keep the spatial resolution the same. The kernel size is set to $3 \times 3 \times 3$ with the same padding for all convolutions. After applying three convolutions, 3D convolution with $1 \times 1 \times 1$ kernel is applied to reduce the number of channels to the number of classes, which is pooled later to get class scores. After each convolution, batch normalization and ReLU are applied. The details of the used 3D-FCN are given in Table 3.2.

The proposed approach is in fact very similar to rMCx models in [128] except for that 3D convolutions are applied at the very end. It is again similar to ECO architecture [183], but 2D features in ECO architecture are extracted again at an early stage of the 2D CNN feature extractor. Although 3D CNN architectures can be used for varying video lengths, we can use 3D convolution layers as a ST modeling technique since we are using a fixed segment size for all the input videos.

3.2.5 Comparative Analysis

The proposed ST modeling techniques are evaluated on two publicly available benchmarks: (i) The Jester dataset that contains dynamic and static hand gesture videos, (ii) the Something-Something dataset that contains videos of various human-object interactions. The common aspect of both these videos is that the proposed recognition architectures need to analyze the full content of the video in order to make a successful recognition, which makes them perfect benchmarks for analyzing ST modeling techniques.



Jester dataset Something-Something-V2 dataset

Figure 3.8: Histograms of the duration of video clips in Jester [4] (left) and Something-Something-V2 [172] (right) datasets.

3.2.5.1 Datasets

The details of the Jester [4] and Something-Something-V2 [172] datasets are already provided in Section 2.5. In addition, we provide the histograms for the duration of video clips in Jester and Something-Something-V2 datasets in Fig. 3.8. The duration of gesture clips in the Jester dataset is concentrated between 30 - 40 frames. However, the Something-Something dataset has videos with relatively varying temporal durations between 20 and 70 frames, which is the reason why 3D CNN architectures accepting fixed-size inputs are not suitable for this benchmark. In order to recognize video clips correctly, the used architectures should incorporate information coming from all parts of the videos.

3.2.5.2 Training Details

Given the ST modeling architecture in Fig. 3.1, the CNN architecture used to extract frame features plays a critical role in the performance of the overall architecture. In order to get *CNN-model-agnostic* performance of the applied ST modeling techniques, the SqueezeNet and BNInception models are used. For both models, the input frames with resolution of 224×224 are used and the features are transformed to 256-dimensional vectors (*Number-of-classes*-dimensional vectors for only TSN) via an MLP after the global pooling layer except for ConvLSTM and FCN3D. For these two ST modeling approaches, spatial resolution (13×13 and 7×7 for SqueezeNet and BNInception) is preserved by removing the final pooling layer, and 1×1 convolution layer is used to transform the number of channels to 256. For recurrent architectures, always single hidden layer is used with the hidden state dimension of 256. The only exception is for Bidirectional LSTM, where we used a hidden state dimension of 128 in order to have the same dimension before the final fully connected layer. For all experiments, CNN models that are used for frame-level feature extraction are initialized with ImageNet pretraining weights and all models are trained end-to-end.

Learning: Stochastic gradient descent (SGD) with standard categorical cross-entropy loss is applied. For momentum and weight decay, 9×10^{-1} and 5×10^{-4} are used, respectively. The learning rate is initialized with 1×10^{-3} and reduced twice with a factor of 10^{-1} after validation loss converges.

Regularization: Several regularization techniques are applied in order to reduce over-fitting and achieve a better generalization. Weight decay of $\gamma = 5 \times 10^{-4}$ is applied to all parameters of the architecture. A dropout layer is applied after the global pooling layer of 2D CNN architectures with a ratio of 0.3. Moreover, data augmentation of multiscale random cropping is applied for both datasets and random horizontal flip is applied for the Something-Something dataset. Random horizontal flipping is not performed for the training of the Jester dataset since this changes the annotations of some classes.

Implementation: The complete ST modeling architecture is implemented and trained end-to-end in PyTorch. We make our code publicly available at https://github.com/fubel/stmodeling for the reproducibility of the results.

3.2.5.3 Resource Efficiency Analysis

For real-time systems, the resource efficiency of the applied ST modeling techniques is as essential as the achieved classification accuracy. Therefore, we have investigated the number of parameters and floating point operations (FLOPs) of each technique, which can be found in Table 3.3. For all calculations, we have used 8-segment for the Jester dataset.

Out of all ST modeling techniques, TSN comes for free since it requires no parameters and there is only averaging operation. However, temporal information is lost due to averaging, which results in inferior performance compared to simple-MLP or TRN techniques.

Transformer based approach is the most costly technique in terms of the number of parameters, although it achieves the best performance on the Something-Something dataset. It contains around 48 M parameters and requires 194 MFLOPs. Its performance is worse than ConvLSTM and 3D-FCN on the Jester dataset since the latter ones keep the spatial resolution of the extracted features, which is critical to capture motion patterns.

ConvLSTM and 3D-FCN do not employ pooling at the end of the feature extraction, hence require relatively high number of FLOPs. Specifically, ConvLSTM is the most costly technique in terms of the number of FLOPs. The number of FLOPs for SqueezeNet is $13^2/7^2 = 3.48$ times higher than BNInception due to the output resolution of feature maps. In terms of the number of parameters, ConvLSTM and 3D-FCN contain 4.73 M and 1.56 M parameters, respectively.

On the other hand, the resource efficiency of the feature extractors (i.e. 2D CNNs) is also important. The BNInception architecture contains 10.27 M parameters and requires 2049 MFLOPs to extract features of a frame with resolution 224×224. On the other hand, the SqueezeNet architecture contains only 0.72 M parameters and requires 269 MFLOPs to extract the features of a same-resolution frame.

Model	MFLOPs	Params	Accuracy (%)					
			Jester (8 seg.)		Something (8 seg.)		Something (16 seg.)	
			Squeez.	BNIncep.	Squeez.	BNIncep.	Squeez.	BNIncep.
Simple-MLP	1.07	1.06M	87.28	92.80	31.89	46.35	33.96	47.01
TSN	0.001	0.00M	72.84	82.74	20.91	37.28	22.15	36.22
TRN-multiscale	6.00	2.34M	88.39	93.20	33.73	46.91	34.38	47.73
RNN_tanh	1.06	0.14M	70.51	79.53	16.12	25.17	14.48	21.64
RNN_ReLU	1.06	0.14M	78.33	88.15	21.40	36.01	15.84	24.88
LSTM	4.24	0.53M	84.28	90.80	25.24	39.04	28.25	42.83
GRU	3.18	0.40M	83.10	90.86	25.40	40.69	30.24	43.31
B-LSTM	3.19	0.40M	84.87	91.12	25.04	39.35	27.88	42.41
ConvLSTM	1850.10/6380.93	4.73M	89.57	93.38	31.31	46.40	32.86	46.64
2D-FCN	77.19	0.56M	88.11	93.64	27.72	39.17	29.95	40.56
3D-FCN	152.32/525.36	1.56M	**90.19**	**94.07**	**37.10**	46.66	37.59	47.37
Transformer	194.10	47.87M	88.52	92.86	36.48	**48.25**	**38.75**	**49.21**

Table 3.3: Comparison of different ST modeling techniques over classification accuracy, number of parameters and computation complexity (i.e., number of Floating Point Operations - FLOPs). Methods are evaluated using 8 and 16 segments on validation sets of Jester-V1 and Something-Something-V2 datasets. The number of parameters and FLOPs are calculated for only ST modeling blocks excluding CNN feature extractors for Jester dataset using 8 segments. FLOPs values of ConvLSTM and 3D-FCN are reported separately for BNInception (left) and SqueezeNet (right) since their spatial resolution is 7×7 and 13×13, respectively.

3.2.5.4 Results Using Jester Dataset

For the Jester dataset, the spatial content for all classes is the same: A hand in front of a camera performing a gesture. Therefore, a designed architecture should capture the form, position, and motion of the hand in order to recognize the correct class.

Comparative results of different ST-modeling techniques for the Jester dataset can be found in Table 3.3. Inspired from [19], we have used eight segments for this benchmark as it achieves the best performance for MFF architecture. Compared to BNInception, architectures with SqueezeNet have 5-10% inferior classification accuracy for the same ST modeling technique. However, the technique-wise comparison remains similar within the same 2D CNN backbone.

Out of all ST modeling techniques, TRN-multiscale, 3D-FCN and ConvLSTM stand out for classification accuracy. Considering the resource efficiency, the simple-MLP model can also be preferred over TRN-multiscale. Surprisingly, RNN based methods except for ConvLSTM, which come to mind first for modeling sequences, perform worse than these techniques.

The superiority of ConvLSTM over other RNN based techniques and superiority of 3D-FCN over 2D-FCN validates the importance of the spatial content. Transformer based approach, which is recently the most popular sequence modeling approach, also performs worse than the ConvLSTM and 3D-FCN. On the Jester dataset, the best performing

technique is 3D-FCN in terms of accuracy. However, preserving the spatial resolution brings the burden of increased computation and number of parameters. As expected, TSN yields the lowest classification accuracy as the averaging operation causes a loss of temporal information.

3.2.5.5 Results Using Something-Something Dataset

Compared to the Jester dataset, the Something-Something dataset contains much more classes with more complex spatial content. In order to identify the correct class label, the designed architectures need to extract the spatial content and temporally link this content successfully. Therefore, the frame feature extractors (i.e., 2D CNNs) are critical for the overall performance.

Comparative results of different ST-modeling techniques for the Something-Something dataset can be found in Table 3.3. Besides the 8-segments architectures, we have also made experiments for 16-segments architectures as the spatial complexity of the dataset is higher compared to Jester. Due to this complexity, architectures with SqueezeNet have 10% to 15% inferior classification accuracy compared to architectures with BNInception.

Compared to 8-segments, 16-segments architectures perform better. However, performance improvement is not as drastic as the effect of feature extractors. This shows that the main complexity of this task comes from the complexity of scenes, not the complexity of finer temporal details. In order to get better performance on the Something-Something dataset, more complex architectures with deeper and wider structures can be preferred.

Out of all ST modeling techniques, Transformer based approach achieves the best classification accuracy for the 16-segments case. However, its computational complexity is significantly higher compared to RNN based and MLP based approaches. 3D-FCN and ConvLSTM also achieve close to Transformer based approach in terms of accuracy, but require significantly more number of FLOPs. TRN-multiscale also stands out for its balance between resource efficiency and accuracy. All RNN based techniques achieve 8-15% worse compared to Transformer based approach. Similar to the Jester dataset, Vanilla RNN and TSN yield the lowest accuracies showing the importance of temporal information for this task.

3.2.6 Summary

In this section, we have analyzed various techniques for CNN-based spatiotemporal modeling techniques and compared them based on a consistent 2D CNN feature extraction of sparsely sampled frames. The individual methods were then evaluated on the Jester and Something-Something datasets. It has been shown that the results heavily depend on the CNN models that are used for feature extraction and the number of used frames. For the Jester dataset, the 3D-FCN technique achieves the best results using both SqueezeNet and BNInception. On the Something-Something dataset, Transformer based approach outperforms all other techniques. It has also been

shown that simple vanilla RNNs are unable to understand the complex spatiotemporal relationships of the data. All the more complex RNNs tested perform very similarly.

Interestingly, the TSN model, which showed state-of-the-art performance on the UCF101 and HMDB benchmarks, performs rather poorly in our experiments, which shows the importance of maintaining the temporal information. Among all techniques, Transformer based approach contains the highest number of parameters and requires the highest FLOPs. Moreover, ConvLSTM and 3D-FCN require a relatively high number of parameters and FLOPs compared to RNN and MLP based techniques, since they do not employ pooling at the feature extractor and preserve spatial resolution. While some models like Transformer, TRN, LSTM, GRU, and B-LSTM can benefit from an increase in the number of segments, Vanilla RNNs and the TSN model can suffer from overfitting. One possibility for future research would be to develop resource efficient ST modeling techniques that preserve the spatial resolution of the extracted features.

3.3 Incorporation of Motion Information to Frame-Level Features

This section addresses the problem of capturing motion information for video analysis methods with frame-level features.

Acquiring spatiotemporal states of actions is the most crucial step for action classification. In this section, we propose a data level fusion strategy, Motion Fused Frames (MFFs), designed to fuse motion information into static images as better representatives of spatiotemporal states of actions. MFFs can be used as input to any deep learning architecture with very little modification on the network. We evaluate MFFs on hand gesture recognition tasks using three video datasets - Jester, ChaLearn LAP IsoGD and NVIDIA Dynamic Hand Gesture Datasets - which require capturing long-term temporal relations of hand movements. Our approach obtains very competitive performance on Jester and ChaLearn benchmarks with the classification accuracies of 96.28% and 57.4%, respectively, while achieving state-of-the-art performance with 84.7% accuracy on the NVIDIA benchmark.

This section is based on our publication *Motion Fused Frames: Data Level Fusion Strategy for Hand Gesture Recognition* [19].

3.3.1 Motivation

Action and gesture recognition have become very popular topics within the computer vision field in the last few years, especially after the application of deep learning in this domain. Similar to other areas of computer vision, the recent work on action and gesture recognition is mainly based on Convolutional Neural Networks (CNNs).

The temporal information in videos has been analyzed using different modalities such as RGB, depth, infrared, and flow images as input. The results show that although each of these modalities provides good recognition performances alone, the fusion analysis of these modalities further increases the recognition performance [94, 93, 133].

The applied fusion strategy plays a critical role in the performance of multimodal gesture recognition. Different modalities can be fused either on the data level, feature level, or decision level. Feature and decision level fusions are the most popular fusion strategies that most of the CNNs currently apply [94, 93]. Although they perform pretty well on action and gesture recognition tasks, they have some drawbacks: (i) Usually a separate network must be trained for each modality, which means the number of trainable parameters is multiple times of a single network; (ii) at most of the time, pixel-wise correspondences between different modalities cannot be established since fusion is only on the classification scores or final fully connected layers; (iii) applied fusion scheme might require complex modifications on the network to obtain good results.

The data level fusion is the most cumbersome one since it requires frame registration, which is a difficult task if the multimodal data is captured by different hardware. However, the drawbacks arising at the feature and decision level fusion methods disappear inherently. Firstly, a single network training is sufficient, which reduces the number of parameters multiple times. Secondly, since different modalities

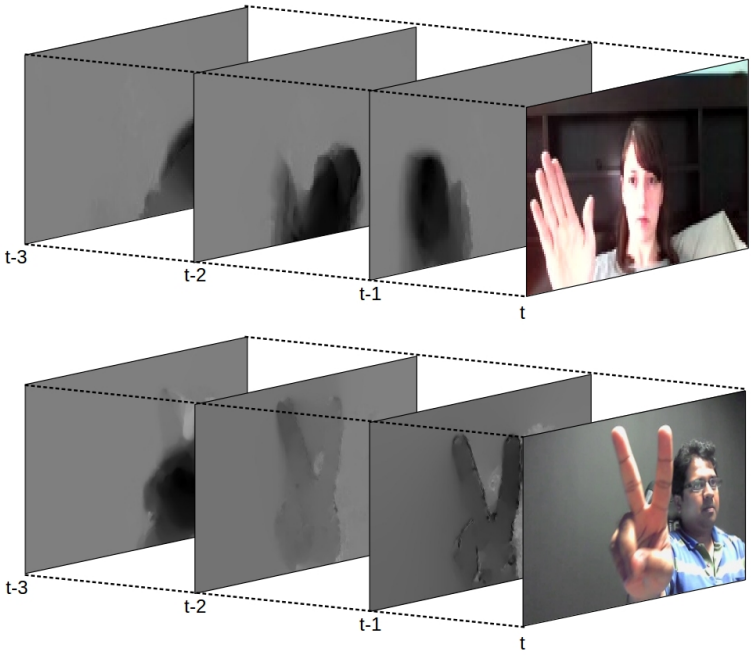


Figure 3.9: Motion Fused Frames (MFFs): Data level fusion of optical flow and color modalities. Appending optical flow frames to static images makes spatial content aware of which part of the image is in motion and how the motion is performed. Top: 'Swipe-right' gesture. Bottom: 'Showing two fingers' gesture.

are fused at the data level, pixel-wise correspondences are automatically established. Lastly, any CNN architecture can be adopted with very little modification.

In this section, we propose a data level fusion strategy, Motion Fused Frames (MFFs), using color and optical flow modalities for hand gesture recognition. MFFs are designed to fuse motion information into static images as better representatives of spatiotemporal states of actions. This makes them favorable since hand gestures are composed of sequentially related action states, and slight changes in these states form new hand gestures. To the best of our knowledge, it is the first time that data level fusion is applied for deep learning based action and gesture recognition.

An MFF is generated by appending optical flow frames to a static image as extra channels. The appended optical flow frames are calculated from the consecutive previous frames of the selected static image. Fig. 3.9 shows two MFF examples: 'Swipe-right' gesture (top) and 'showing two fingers' gesture (bottom). In the top example of Fig. 3.9, by looking at only the static image, one can infer the information of a lady holding her hand upward. However, incorporating optical flow frames into the static images brings extra motion information, which shows that the hand is actually moving from left to right making it a swipe-right gesture.

We evaluated MFFs on three publicly available datasets, which are Jester Dataset [4], ChaLearn LAP IsoGD Dataset (ChaLearn) [170] and NVIDIA Dynamic Hand

Gesture Dataset (nvGesture) [133]. Our approach obtains very competitive performance on Jester and ChaLearn datasets with the classification accuracies of 96.28% (2^{nd} place in the leaderboard) and 57.4%, respectively, while achieving state-of-the-art performance with 84.7% accuracy on the nvGesture dataset.

3.3.2 Related Work

The literature review on action and gesture recognition is already provided in Section 2.1 and Section 2.2, respectively. Here, we will provide related work on the fusion of multiple modalities for action and gesture recognition.

Fusion of information from different modalities is also a common approach in CNNs to increase recognition performance. There are three main variants for information fusion in deep learning models: data level, feature level and decision level fusions. Within each fusion strategy, still different approaches exist. For instance, for decision level fusion, averaging [93, 133], concatenating [91] or consensus voting can be applied on the scores of different modalities trained on separate networks. For the feature level fusion case, features from different layers of the CNNs can be fused at different levels [94], or different schemes can be proposed as in [162], which proposes a canonical correlation analysis based fusion scheme.

Out of all fusion strategies, the data level fusion is the least used one so far since data preparation requires effort especially when different hardware is used for different modalities. However, it has very critical advantages over feature and decision level fusions like training only a single network, or automatically establishing the pixel-wise correspondence between different modalities. Thus, we propose a data level fusion strategy, Motion Fused Frames, to draw attention to these advantages.

3.3.3 Methodology

In this section, we describe Motion Fused Frames and the network architecture used for hand gesture recognition. Particularly, we first define MFFs and explain how to form them. Then, we introduce the network architecture which takes advantage of this data fusion strategy. Finally, we describe the training details on experimented datasets.

3.3.3.1 Motion Fused Frames

A single RGB image usually contains static appearance information at a specific time instant and lacks contextual and temporal information about previous and next frames. As a result, single video frames cannot represent the actual states of actions completely.

Motion fused frames are inspired to be better representatives of action states. As the name implies, they are composed by fusing motion information into static RGB images. Motion information is simply optical flow frames that are calculated from consecutive previous frames, and the fusion method is achieved by appending optical flow frames to RGB frames as extra channels, as illustrated in Fig. 3.9. Therefore, an MFF contains (i) *spatial* content contained in RGB channels, and (ii) *motion* content contained in $optical flow$ channels. Since optical flow images are computed from RGB images, the

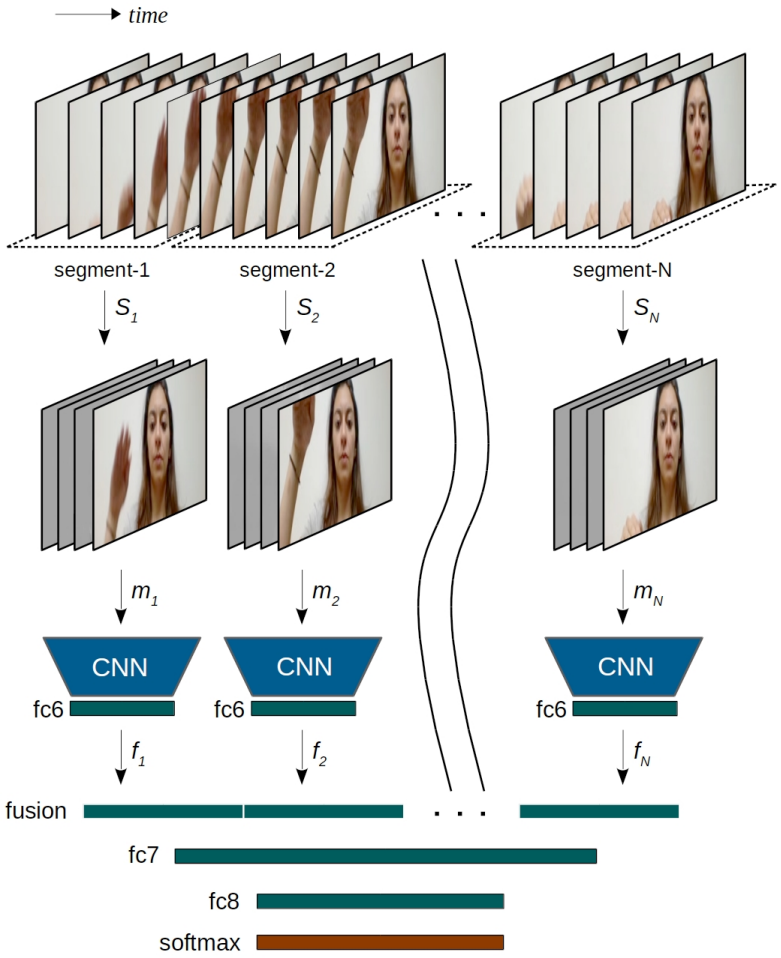


Figure 3.10: Network Architecture for N-segment 3-motion-1-frame MFF (N-MFFs-3f1c): One input video is divided into N segments, and equidistant frames are selected from the created segments. 3 optical flow frames calculated from previous frames are appended to RGB frames as extra channels, which forms the Motion Fused Frames (MFFs). Each MFF is fed into a CNN to extract a feature vector representing the spatiotemporal state of the segment. Extracted features are concatenated at the fusion layer and passed to fully connected layers to get class scores.

approach needs in fact only the RGB modality, which avoids the need for enhanced sensors providing several modalities like depth and infrared images.

Blending motion information into contextual information, as in MFFs, ensures pixel-wise correspondence automatically. In other words, spatial content is aware of which part of the image is in motion and how the motion is performed.

The quality of motion estimation techniques plays a critical role in the performance of gesture recognition. In [184], the performance of several optical flow estimation techniques are tested to investigate the dependency of action recognition on the quality

of motion estimation. It has been experimentally proved that Brox flow [48] performs better compared to MPEG flow [185] and Farneback [47] techniques. Therefore, we have computed horizontal and vertical components of optical flow frames using the Brox technique. We have scaled optical flow frames according to the maximum of the absolute values of horizontal and vertical components and mapped discretely into the interval [0, 255]. Using this step, the range of optical flow frames becomes the same as RGB images.

3.3.3.2 Network Architecture

In this study, we use a deep convolutional neural network architecture applied on segmented video clips, as illustrated in Fig. 3.10. The architecture consists of 4 parts: The formation of MFFs, a deep CNN to extract spatiotemporal features from MFFs, the fusion of features from different segments and fully connected layers for global temporal modeling, and finally a softmax layer for predicting class-conditional gesture probabilities.

We first divide entire video clip $\mathcal{V}$ into N segments. Each video segment is represented as $S_n \in \mathbb{R}^{w \times h \times c_{rgb} \times m}$ of $m \geq 1$ sequential frames of size $w \times h$ pixels with $c_{rgb} = 3$ channels. Then, within segments, equidistant color frames are selected randomly. Each segment is transformed with $\mathcal{M}$ into an MFF $\mathbf{m}_n$ by appending precomputed optical flow frames to the selected color frames:

$$\begin{aligned} &\mathcal{M} : \mathbb{R}^{w \times h \times c_{rgb} \times m} \rightarrow \mathbb{R}^{w \times h \times c_{mff}}, \\ &where\ m_n = \mathcal{M}(S_n). \end{aligned} \tag{3.18}$$

The number of channels in an MFF can be calculated with $c_{mff} = c_{rgb} + n.c_{flow}$, where n is the number of optical flow frames appended for each segment, and c_{flow} is the number of channels in flow frames that is equal to 2 containing horizontal and vertical components. For instance, an MFF containing 3 flow and 1 color frames, as in Fig. 3.9, has $c_{mff} = 3 + 3.2 = 9$ channels. Each MFF $\mathbf{m}_n$ is then transformed into a feature representation $\mathbf{f}_n$ by a CNN $\mathcal{F}$:

$$\mathcal{F} : \mathbb{R}^{w \times h \times c_{mff}} \rightarrow \mathbb{R}^q,\ where\ f_n = \mathcal{F}(m_n). \tag{3.19}$$

After extracting feature representations $\mathbf{f}_n$ of each MFF, we concatenate them by keeping their order intact:

$$(f_1 \oplus f_2 \oplus ... \oplus f_N) \in \mathbb{R}^{N \times q}, \tag{3.20}$$

where $\oplus$ refers to concatenation. We pass this vector into a two-layer multilayer perceptron (MLP). The intuition behind this is that the MLP will be able to infer the temporal features from the sequence inherently, without having to know that it is a sequence at all. Finally, a softmax layer is applied to get the class-conditional probabilities of each class. ReLU nonlinearity is applied between all convolutional and

fully connected layers except for the final fully connected layer which has no nonlinearity.

A network architecture dividing gesture videos into N segments and transforming each segment into an MFF by appending n optical flow frames to 1 color image is referred as N-MFFs-nf1c.

3.3.3.3 Training Details

The CNN architecture used to extract features from MFFs is critical for the performance of the overall network, and it has been experimented that deeper architectures like ResNet [88] perform slightly better results. However, we aim to evaluate the effectiveness of the proposed data level fusion strategy, Motion Fused Frames, in hand gesture recognition. Therefore, following the design choices of [186], we adopted Inception with Batch Normalization (BNInception) [178] pretrained on ImageNet as baseline architecture due to its good balance between accuracy and efficiency. We also apply the same training strategies of partial-BN (freezing the parameters of all Batch Normalization layers except the first one) and adding an extra dropout layer after the global pooling layer in BNInception architecture. For fc6, fc7 and fc8 layers in Fig. 3.10, we used one-layer MLPs with 256, 512 and class-number units, respectively.

For the Jester dataset, we modify the weights of the first convolution layer of the pretrained BNInception model to accommodate MFFs. Specifically, the weights across the RGB channels are averaged and replicated through the appended optical flow channels. For ChaLearn and nvGesture datasets, training is started with the pretrained models on the Jester dataset.

Learning. We use stochastic gradient descent (SGD) applied the to mini-batch of 32 videos with standard categorical cross-entropy loss. The momentum and weight decay are set to 0.9 and 5×10^{-4}, respectively. The learning rate is initialized with 1×10^{-3} for all the experiments. For the Jester dataset, the learning rate is reduced twice with a factor of 10^{-1} after 25^{th} and 40^{th} epochs, and optimization is completed after 5 more epochs. For the ChaLearn dataset, the learning rate is reduced twice with a factor of 4^{-1} after 15^{th} and 30^{th} epochs, and optimization is completed after 10 more epochs. Finally, for the nvGesture dataset, the learning rate is reduced twice with a factor of 4^{-1} after 40^{th} and 80^{th} epochs, and optimization is completed after 20 more epochs. These training rules are applied for the 8-MFFs-3f1c architecture and approximately the same for the other architectures.

Regularization. We apply several regularization techniques to reduce over-fitting. Weight decay ($\gamma = 5 \times 10^{-4}$) is applied to all parameters of the network. We use a dropout layer after the global pooling layer (before fc6 in Fig. 3.10) of BNInception architecture. For the Jester dataset, the dropout ratio in this layer is kept at 0.8 throughout the whole training process. However, over-fitting is much more severe for ChaLearn and nvGesture datasets since the average number of training samples per class is much smaller compared to the Jester dataset (4391, 144 and 42 training samples per class for Jester, ChaLearn and nvGesture datasets, respectively). Therefore, we apply an additional dropout layer

after the fc7 layer for these datasets. The dropout ratio is initialized with 0.5 for both dropout layers and increased to 0.8 and 0.9 when the learning rates are reduced. Gradual increase of dropout ratio helps faster convergence while keeping over-fitting in control, which helps to save a considerable amount of training time.

Data Augmentation. Various data augmentations steps are applied in order to increase the diversity of the training videos: (a) Random scaling ($\pm 20\%$), (b) random spatial rotation ($\pm 20^{\circ}$), (c) spatial elastic deformation [187] with pixel displacement of $\alpha = 15$ and standard deviation of the smoothing Gaussian kernel $\sigma = 20$ pixels (applied with probability 50%), (d) random cropping, scale jittering and aspect ratio jittering as in [186], (e) flipping horizontally with probability 50% (for only ChaLearn dataset), (f) temporal scaling ($\pm 10\%$) and jittering ($\pm$ 2 frames) (for only nvGesture dataset). All these data augmentation steps are applied online and the input is finally resized to 224×224 for network training.

Implementation. We have implemented our approach in PyTorch [188] with a single NVIDIA Titan Xp GPU. We make our code publicly available at `https://github.com/okankop/MFF-pytorch` for the reproducibility of the results.

3.3.4 Experiments

The performance of the proposed approach is tested on three publicly available datasets: Jester dataset, Chalearn LAP RGB-D Isolated Gesture dataset and NVIDIA Dynamic Hand Gesture dataset. For the evaluation part, center cropping with equidistant frames (middle frame in each segment) in the videos is used for all the datasets.

3.3.4.1 Results Using Jester Dataset

The details of the Jester dataset are provided in Section 2.5. We initially investigated the effects of the number of appended optical flow frames on the performance of single segment architectures (1-MFFs-nf1c). So, we took the complete gesture videos as one segment and tried to classify them using a single RGB image with varying number of appended optical flow frames. We started with 0 optical flow frames and gradually increased it to 11. The results in the first part of Table 3.4 show that every extra optical flow frame improves the performance further (from 63.60% to 82.93%). The performance boost is significant for the very first optical flow frame with around 9% accuracy gain.

Secondly, we analyze the effects of the segment number selection for gesture videos. Fixing the number of appended optical frames to 3, we have experimented with 2, 4, 6, 8, 10 and 12-MFFs architectures. The results in the second part of Table 3.4 show that the performance increases as we increase the number of segments until reaching the 8 segmented architecture. Then the performance decreases gradually as we keep increasing the segment number. In this analysis, it is found that 8 segmented architecture performs best.

Lastly, we analyze the effects of the number of appended optical flow frames on the best performing segment size (8-MFFs-nf1c) by varying the number of optical flow frames from 0 to 3. Results in the last part of Table 3.4 show that every extra optical flow

Model	Top1 Acc.(%)	Top5 Acc.(%)
1-MFFs-0f1c	63.60	92.44
1-MFFs-1f1c	72.83	93.96
1-MFFs-2f1c	73.66	94.10
1-MFFs-3f1c	74.09	94.17
1-MFFs-5f1c	78.39	95.84
1-MFFs-7f1c	81.15	96.69
1-MFFs-9f1c	82.69	97.06
1-MFFs-11f1c	82.93	97.07
2-MFFs-0f1c	75.65	94.40
2-MFFs-3f1c	84.22	97.84
4-MFFs-3f1c	92.18	99.41
6-MFFs-3f1c	94.72	99.66
8-MFFs-3f1c	95.36	99.75
10-MFFs-3f1c	95.12	99.69
12-MFFs-3f1c	94.73	99.69
8-MFFs-0f1c	92.90	99.41
8-MFFs-1f1c	94.20	99.61
8-MFFs-2f1c	94.67	99.62
8-MFFs-3f1c	95.36	99.75
8-MFFs-3f1c (5 crop)	**96.33**	**99.86**

Table 3.4: Results on the validation set of Jester dataset V1.

frame again boosts the performance further. However, the performance boost is more significant for smaller segment architectures like 2-MFFs or 1-MFFs. Out of all models, 8-MFFs-3f1c with 5-crop data augmentation shows the best performance.

We evaluate the 8-MFFs-3f1c architecture on the test set and submit our predictions to the official leaderboard of the Jester dataset [4]. At the submission time, our approach is in the second place as shown in Table 3.5.

3.3.4.2 Results Using ChaLearn Dataset

The details of the ChaLearn dataset can be found in Section 2.5. Experiments on the Jester dataset proved that applying MFFs on 8 segmented videos performs better than applying smaller segments. Therefore, we have experimented MFFs on 8 segmented videos with varying the number of optical flow frames. Acquired results for models 8-MFFs-nf1c, where n ranges from 0 to 3, are given in Table 3.6 and Table 3.7 for validation and test sets, respectively. Compared to the Jester dataset, there is a

Model	**Top1 Acc.(%)**
C3D	94.62
Multiscale TRN [91]	94.78
SJ	94.87
Guangming Zhu	95.01
DIN	95.31
NUDT_PDL	95.34
MFNet	96.22
8-MFFs-3f1c	**96.28**
DRX3D	**96.60**

Table 3.5: Results on the test set of Jester dataset V1.

remarkable performance boost (accuracy gain of 15.6% and 13.9% for validation and test sets, respectively) as the number of optical flow frames increases. It must be noted that created MFFs represents a larger time span for the ChaLearn dataset since frames are captured with a rate of 10 fps. This gives an intuition that acquired performance at Jester dataset can also be boosted by appending flow frames from earlier timestamps. However, we leave this issue as a future research work.

The best performing model (8-MFFs-3f1c) is compared with several state-of-the-art methods. According to Table 3.8, best results are reported in case three modalities are used at the same time. Our approach performs better than most of the approaches reported in the table without using the depth modality, which is a significant advantage of the proposed approach.

3.3.4.3 Results Using nvGesture Dataset

The details of the nvGesture dataset can be found in Section 2.5. Since the gesture videos are weakly segmented in the nvGesture dataset, we cropped the first and the last 10 frames and used the center 60 frames for the test set evaluation, where the gestures occur most of the time.

Method	**Modality**	**Acc.(%)**
8-MFFs-0f1c	RGB	41.3
8-MFFs-1f1c	RGB + Flow	48.4
8-MFFs-2f1c	RGB + Flow	50.0
8-MFFs-3f1c	RGB + Flow	**56.9**

Table 3.6: Results on the validation set of ChaLearn dataset.

Method	**Modality**	**Acc.(%)**
8-MFFs-0f1c	RGB	42.8
8-MFFs-1f1c	RGB + Flow	53.7
8-MFFs-2f1c	RGB + Flow	53.9
8-MFFs-3f1c	RGB + Flow	**56.7**

Table 3.7: Results on the test set of ChaLearn dataset.

Method	Modality	Acc.(%)
8-MFFs-0f1c	RGB	41.36
ResC3D [162]	RGB	45.07
ResC3D [162]	Depth	48.44
ResC3D [162]	Flow	44.45
Scene Flow [189]	RGBD	36.27
Wang et al. [190]	RGBD	39.23
Pyramidal C3D [191]	RGBD	45.02
2SCVN+3DDSN [192]	RGBD	49.17
32-frame C3D [193]	RGBD	49.20
C3D+LSTM [194]	RGBD	51.02
8-MFFs-3f1c	RGB + Flow	56.9
8-MFFs-3f1c (5 crop)	RGB + Flow	**57.4**
Zhang et al. [195]	RGBD + Flow	58.65
Wang et al. [139]	RGBD + Flow	60.81
ResC3D [162]	RGBD + Flow	**64.40**

Table 3.8: Comparison with state-of-the-art on the validation set of ChaLearn dataset.

Method	Modality	Acc. (%)
HOG+HOG^2 [196]	RGB	24.5
Spatial stream CNN [93]	RGB	54,6
iDT-HOG [197]	RGB	59.1
C3D [85]	RGB	69.3
R3DCNN [133]	RGB	74.1
iDT-HOF [197]	Flow	61.8
Temporal stream CNN [93]	Flow	68,0
iDT-MBH [197]	Flow	76.8
R3DCNN [133]	Flow	77.8
Two stream CNN [93]	RGB + Flow	65,6
iDT [197]	RGB + Flow	73.4
R3DCNN [133]	RGB + Flow	79.3
6-MFFs-3f1c	RGB + Flow	82.4
8-MFFs-3f1c	RGB + Flow	**84.7**
R3DCNN [133]	all modalities*	83.8
Human [133]	RGB	**88.4**

Table 3.9: Comparison with state-of-the-art on nvGesture dataset. *All modalities refer to RGB, optical flow, depth, infrared and infrared disparity modalities.

Although this training set is a lot smaller compared to other datasets, using pretrained models on the Jester dataset helps us to remove the over-fitting impact considerably. In Table 3.9, we give the comparison of our approach with the state-of-the-art models. Compared to the popular C3D and two stream CNN architectures, our approach can achieve 14.4% and 19.1% accuracy gain, respectively. Our approach performs state-of-the-art performance on this benchmark, although we only use color and optical flow modalities.

The dataset providers also evaluated the human performance by asking six subjects to label each gesture video in the test set for the color modality. Gesture videos are presented to human subjects in random order and only once to be consistent with machine classifiers. Human accuracy is reported as 88.4% for color modality.

3.3.5 Summary

This section presents a novel data level fusion strategy, Motion Fused Frames, by fusing motion information (optical flow frames) into RGB images for hand gesture recognition.

We evaluated the proposed MFFs on several recent datasets and acquired competitive results using only optical flow and color modalities. Our results show that the fusion of more motion information improves the performance further in all cases. The performance increase at the first appended optical flow frame is especially significant.

Chapter 4

Video Analysis With Clip-Level Features

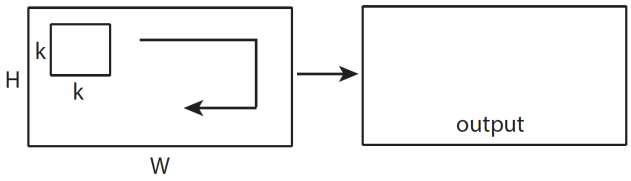
This chapter presents video analysis with clip-level features. First, motivations of using clip-level features are mentioned in Section 4.1. Then, Section 4.2 presents a single-stage spatiotemporal action localization architecture, which makes use of clip-level features extracted by 3D CNNs. Afterwards, Section 4.3 presents several resource efficient 3D CNN architectures. Section 4.4 addresses the challenges of online hand gesture recognition by using clip-level features in a two-level hierarchical architecture. Finally, Section 4.5 presents Dissected 3D CNN architecture, which applies a caching mechanism in order to reduce the computational complexity for the extraction of clip-level features at online operation.

4.1 Introduction

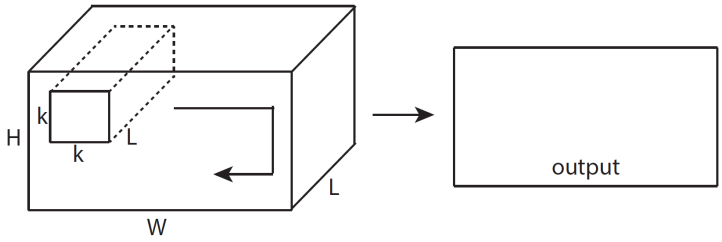
A video is a 3D tensor with 2 spatial dimensions and 1 temporal dimension. Therefore, it is a natural choice to apply 3D convolutions to capture spatiotemporal information in video clips. Fig. 4.1 illustrates the application of 2D and 3D convolution operations.

In recent years, 3D CNNs have dominated the tasks related to video understanding due to their supremacy at capturing motion patterns. In general, 3D CNNs contain two main advantages: (i) They capture clip-level features by operating directly on video clips without needing a separate ST modeling technique. We note that the ST modeling techniques explained in Section 3.2 can still be applied over clip-level features to cover much longer temporal content. (ii) They achieve superior results compared to the methods based on frame-level features. Due to these advantages, we also make use of 3D CNNs in all sections of this chapter, mostly addressing their drawbacks. As an example, Section 4.2 makes use of a 3D CNN in its spatiotemporal branch to extract clip-level features for a unified action localization architecture.

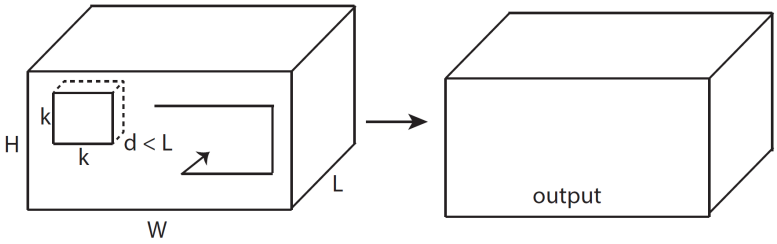
On the other hand, there are also several drawbacks of 3D CNNs: (i) 3D CNNs require significantly more parameters and computations compared to their 2D counterparts which make them harder to train and prone to overfitting. Section 4.3 addresses this drawback by presenting several resource efficient 3D CNN architectures. Moreover, in order to save resources at online operation when the system needs to be idle, Section 4.4 presents a two-level hierarchical architecture. (ii) 3D CNNs always operate on inputs with fixed size (mostly 8, 16 or 32 frames). (iii) 3D CNNs mostly apply temporal downsampling in order to reduce the computational cost at the later stages of the network, which results in an architecture that does not preserve temporal



(a) 2D convolution



(b) 2D convolution on multiple frames



(c) 3D convolution

Figure 4.1: Comparison of 2D and 3D convolution operations. Adapted from [85].

resolution. (iv) At online operation, 3D CNNs are mostly used with a sliding window approach. However, using a smaller stride compared to input clip length results in resource waste due to reprocessing frames in the overlapping regions. In order to address the later 3 drawbacks, Section 4.5 presents Dissected 3D CNN architecture.

Although there is a rising trend in the research community to apply transformer based approaches for the vision tasks [112, 113, 115, 118, 119, 120, 122], 3D CNNs are still been actively used in video understanding tasks [132, 198, 199, 200, 201]. They are specifically valuable for clip-level feature extraction.

4.2 Spatiotemporal Action Localization With Clip-Level Features

This section presents an architecture, which makes use of a 3D CNN in its spatiotemporal branch to extract clip-level features for the task of spatiotemporal action localization.

Spatiotemporal action localization requires the incorporation of two sources of information into the designed architecture: (1) temporal information from the previous frames and (2) spatial information from the key frame. Current state-of-the-art approaches usually extract these two types of information with separate networks and use an extra mechanism for fusion to get detections. In this work, we present You Only Watch Once (YOWO), a unified CNN architecture for real-time spatiotemporal action localization in video streams. YOWO is a single-stage architecture with two branches to extract temporal and spatial information concurrently and predict bounding boxes and action probabilities directly from video clips in one evaluation. Since the whole architecture is unified, it can be optimized end-to-end. The YOWO architecture is fast providing 34 frames-per-second on 16-frames input clips and 62 frames-per-second on 8-frames input clips, which is currently the fastest state-of-the-art architecture on spatiotemporal action localization task. Remarkably, YOWO outperforms the previous state-of-the art results on J-HMDB-21 and UCF101-24 with an impressive improvement of ∼3% and ∼12%, respectively. Moreover, YOWO is the first and only single-stage architecture that provides competitive results on the AVA dataset. This section is based on our publication *You Only Watch Once: A Unified CNN Architecture for Real-Time Spatiotemporal Action Localization* [20].

4.2.1 Motivation

The topic of spatiotemporal human action localization has been spotlighted in recent years, which aims to not only recognize the occurrence of an action but also localize it in both time and space. In such a task, compared to object detection in static images, temporal information plays an essential role. Finding an efficient strategy to aggregate spatial as well as temporal features makes the problem even more challenging. On the other hand, real-time human action detection is becoming increasingly crucial in numerous vision applications, such as human-computer interaction (HCI) systems, unmanned aerial vehicle (UAV) monitoring, autonomous driving, and urban security systems. Therefore, it is desirable and worthwhile to explore a more efficient framework to tackle this problem.

Inspired by the remarkable object detection architecture Faster R-CNN [109], most state-of-the-art works [147, 146] extend the classic two-stage network architecture to action detection, where a number of proposals are produced in the first stage, then classification and localization refinement are performed in the second stage. However, these two-stage pipelines have three main shortcomings in the spatiotemporal action localization task. Firstly, the generation of action tubes, which consist of bounding boxes across frames is much more complicated and time-consuming than the 2D case. The classification performance is extremely dependent on these proposals, where the

Figure 4.2: Standing or sitting? Although the person can be successfully detected, correct classification of the action cannot be made by looking only at the key frame. Temporal information from previous frames needs to be incorporated in order to understand if the person is sitting (left) or standing (right). Examples are from J-HMDB-21 dataset.

detected bounding boxes might be sub-optimal for the following classification task. Secondly, the action proposals focus only on the features of humans in the video, neglecting the relationship between humans and some attributes in the background, which yet can provide considerably crucial context information for action prediction. The third problem of a two-stage architecture is that training the region proposal network and the classification network separately does not guarantee finding the global optimum. Instead, only the local optimum from the combination of two stages can be found. The training cost is also higher than single-stage networks, hence it takes a longer time and needs more memory.

In this section, we propose a novel single-stage framework, YOWO (You Only Watch Once), for spatiotemporal action localization in videos. YOWO prevents all of the three shortcomings mentioned above with a single-stage architecture. The intuitive idea of YOWO arises from human's visual cognitive system. For example, when we are absorbed into the story of a soap opera in front of the TV, each time our eyes capture a single frame. In order to understand which action each artist is performing, we have to relate current frame information (2D features from the key frame) to the obtained knowledge from previous frames saved in our memory (3D features from the clip). Afterwards, these two kinds of features are fused to provide us with a reasonable conclusion. The example in Fig. 4.2 illustrates our inspiration.

YOWO architecture is a single-stage network with two branches. One branch extracts the spatial features of the key frame (i.e. current frame) via a 2D CNN while the other branch models the spatiotemporal features of the clip consisting of previous frames via a 3D CNN. To this end, YOWO is a causal architecture that can operate online on incoming video streams. In order to aggregate these 2D CNN and 3D CNN features smoothly, a channel fusion and attention mechanism is used, where we get the utmost out of inter-channel dependencies. Finally, we produce frame-level detections using the fused features, and provide a linking algorithm to generate action tubes.

In order to maintain real-time capability, we have operated YOWO on RGB modality. However, it must be noted that YOWO architecture is not restricted to operate only on

RGB modality. Different branches can be inserted into YOWO for different modalities such as optical flow, depth, etc. Moreover, in its 2D CNN and 3D CNN branches, any CNN architecture can be used according to the desired runtime performance, which is critical for real-world applications.

YOWO operates with maximum 16 frames input since short clip lengths are necessary to achieve faster runtime for spatiotemporal action localization task. However, such a small clip size is a limiting factor for the accumulation of temporal information. Therefore, we have made use of the long-term feature bank [202] by extracting features with 3D CNN for non-overlapping 8-frame clips for the whole videos using the trained 3D CNN. Training of YOWO performed normally, but at inference time, we have averaged the 3D features centering the key frame. This brought a considerable 6.9% and 1.3% frame-mAP increase on the final performance of the network.

Contributions of this section are summarized as follows:

(i) We propose a real-time single-stage framework for spatiotemporal action localization in video streams, named YOWO, which can be trained end-to-end with high efficiency. To the best of our knowledge, this is the first work that achieves bounding box regression on features extracted by a 2D CNN and 3D CNN, concurrently. These two kinds of features have a complementary effect to each other for the final bounding box regression and action classification. Moreover, we use a channel attention mechanism to aggregate the features smoothly from the two branches above. We experimentally prove that the channel-wise attention mechanism models the inter-channel relationship within the concatenated feature maps and boosts the performance significantly by fusing features more reasonably.

(ii) We perform a detailed ablation study on the YOWO architecture. We examined the effect of 3D CNN, 2D CNN, their aggregation and the fusion mechanism. Moreover, we have experimented with different 3D CNN architectures and different clip lengths to explore a further trade-off between precision and speed.

(iii) YOWO is evaluated on the AVA dataset, which is the first and only single-stage architecture that achieves competitive results compared to the state-of-the-art. Moreover, YOWO is the only causal architecture (i.e. future frames are not leveraged) that is evaluated on the AVA dataset, hence can operate online.

(iv) We evaluate YOWO on J-HMDB-21 and UCF101-24 benchmarks and establish new state-of-the-art results with an impressive 3.3% and 12.2% improvements on frame-mAP, respectively. Moreover, YOWO runs with 34 fps for 16-frames input clips and 62 fps for 8-frames input clips, which is the fastest state-of-the-art architecture available for spatiotemporal action localization task.

4.2.2 Related Work

The literature review on action recognition and spatiotemporal action localization is already provided in Section 2.1 and Section 2.3, respectively. Here, we will provide

related work on attention mechanisms in CNN architectures, which has been actively used at YOWO architecture.

Attention is an effective mechanism to capture long-range dependencies and has been attempted to be used in CNNs to boost the performance in image classification [203, 204, 205] and scene segmentation [206]. Attention mechanism is implemented spatial-wise and channel-wise in these works, in which spatial attention addresses the inter-spatial relationship among features while channel attention enhances the most meaningful channels and weakens the others. As a channel-wise attention block, Squeeze-and-Excitation module [207] is beneficial to increase CNN's performance with little computational cost. On the other hand, for video classification tasks, non-local block [208] takes spatiotemporal information into account to learn the dependencies of features across frames, which can be viewed as a self-attention strategy.

4.2.3 Methodology

In this section, we first present YOWO's architecture in detail, which extracts 2D features from the key frame as well as 3D features from the input clip concurrently and aggregates them together. Afterwards, the implementation of channel fusion and attention mechanism is discussed, which provides the essential performance boost. Finally, we describe the details of the training process for the YOWO architecture and the improved bounding box linking strategy for the generation of action tubes in untrimmed videos.

4.2.3.1 YOWO architecture

The YOWO architecture is illustrated in Fig. 4.3, which can be divided into four major parts: 3D CNN branch, 2D CNN branch, CFAM and bounding box regression parts.

3D CNN Branch

Since contextual information is crucial for human action understanding, we utilize 3D CNN to extract spatiotemporal features. 3D CNNs are able to capture motion information by applying convolution operation not only in spatial dimensions but also in the time dimension. The basic 3D CNN architecture in our framework is 3D-ResNext-101 due to its high performance in Kinetics dataset [131]. In addition to 3D-ResNext-101, we have also experimented with different 3D CNN models in our ablation study. For all 3D CNN architectures, all of the layers after the last convolution layer are discarded. The input to the 3D network is a clip of a video, which is composed of a sequence of successive frames in time order, and has a shape of $[C \times D \times H \times W]$, while the last convolution layer of 3D ResNext-101 outputs a feature map of shape $[C' \times D' \times H' \times W']$ where $C = 3$, D is the number of input frames, H and W are height and width of input images, C' is the number of output channels, $D' = 1$, $H' = \frac{H}{32}$ and $W' = \frac{W}{32}$. The depth dimension of the output feature map is reduced to 1 such that output volume is squeezed to $[C' \times H' \times W']$ in order to match the output feature map of 2D CNN.

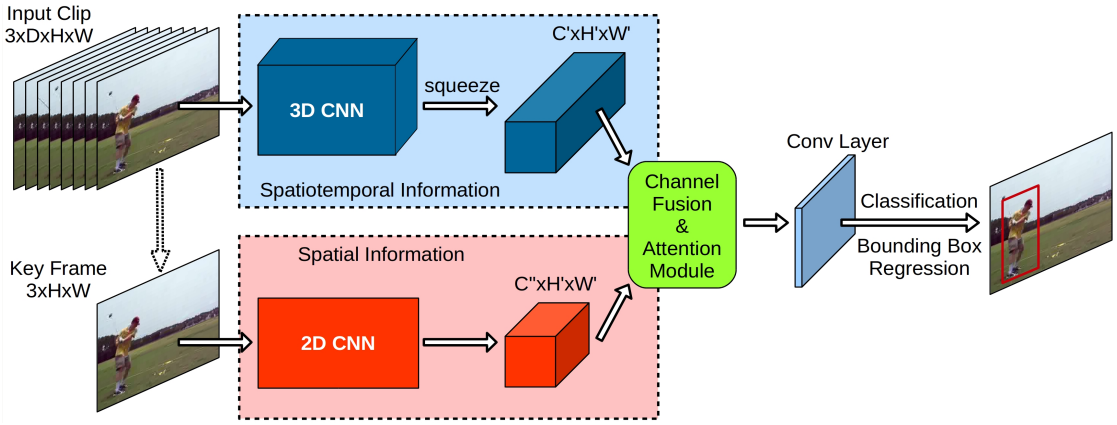


Figure 4.3: The YOWO architecture. An input clip and corresponding key frame is fed to a 3D CNN and 2D CNN to produce output feature volumes of $[C'' \times H' \times W']$ and $[C' \times H' \times W']$, respectively. These output volumes are fed to channel fusion and attention mechanism (CFAM) for a smooth feature aggregation. Finally, one last conv layer is used to adjust the channel number for final bounding box predictions.

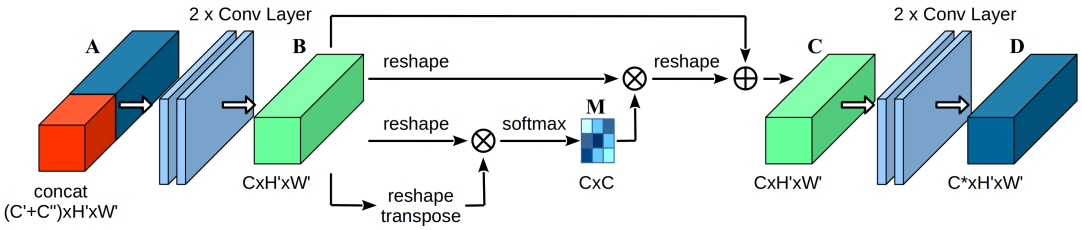


Figure 4.4: Channel fusion and attention mechanism for aggregating output feature maps coming from 2D CNN and 3D CNN branches.

2D CNN Branch

In the meantime, to address the spatial localization problem, 2D features of the key frame are also extracted in parallel. We employ Darknet-19 [209] as the basic architecture in our 2D CNN branch due to its good balance between accuracy and efficiency. The key frame with the shape $[C \times H \times W]$ is the most recent frame of the input clip, thus there is no need for an additional data loader. The output feature map of Darknet-19 has a shape of $[C'' \times H' \times W']$ where $C = 3$, C'' is the number of output channels, $H' = \frac{H}{32}$ and $W' = \frac{W}{32}$ similar to the 3D CNN case.

Another important characteristic of YOWO is that architectures in 2D CNN and 3D CNN branches can be replaced by arbitrary CNN architectures, which makes it more flexible. YOWO is designed to be simple and effort-saving to switch models. It must be noted that although YOWO has two branches, it is a unified architecture and can be trained end-to-end.

Feature aggregation:Channel Fusion and Attention Mechanism (CFAM)
We make the outputs of both 3D and 2D networks are of the same shape in the last two dimensions such that these two feature maps can be fused easily. We fuse the two feature maps using concatenation which simply stacks the features along channels. As a result, the fused feature map encodes both motion and appearance information which we pass as input to the CFAM module, which is based on the Gram matrix to map inter-channel dependencies. Although the Gram matrix based attention mechanism is originally used for style transfer [210] and recently in segmentation task [206], such an attention mechanism is beneficial for fusing features coming from different sources reasonably, which improves the overall performance significantly.

Fig. 4.4 illustrates the used CFAM module. The concatenated feature map $\mathbf{A} \in \mathbb{R}^{(C'+C'')\times H\times W}$ can be regarded as an abrupt combination of 2D and 3D information, which neglects interrelationship between them. Therefore, we first feed A into two convolutional layers to generate a new feature map $\mathbf{B} \in \mathbb{R}^{C\times H'\times W'}$. Afterwards, several operations are performed on the feature map $\mathbf{B}$.

Assume $\mathbf{F} \in \mathbb{R}^{C\times N}$ is the reshaped tensor from feature map $\mathbf{B}$, where $N = H \times W$, which means that features in every single channel is vectorized to one dimension:

$$\mathbf{B} \in \mathbb{R}^{C\times H\times W} \xrightarrow{vectorization} \mathbf{F} \in \mathbb{R}^{C\times N}. \tag{4.1}$$

Then a matrix product between $\mathbf{F} \in \mathbb{R}^{C\times N}$ and its transpose $\mathbf{F}^{\mathrm{T}} \in \mathbb{R}^{N\times C}$ is performed to produce Gram matrix $\mathbf{G} \in \mathbb{R}^{C\times C}$, which indicates the feature correlations across channels [210]:

$$\mathbf{G} = \mathbf{F}\cdot\mathbf{F}^{\mathrm{T}} \quad with \quad G_{ij} = \sum_{k=1}^{N} F_{ik}\cdot F_{jk}, \tag{4.2}$$

where each element G_{ij} in the Gram matrix $\mathbf{G}$ represents the inner product between the vectorized feature maps i and j. After computing the Gram matrix, a softmax layer is applied to generate channel attention map $\mathbf{M} \in \mathbb{R}^{C\times C}$:

$$M_{ij} = \frac{exp(G_{ij})}{\sum_{j=1}^{C} exp(G_{ij})}, \tag{4.3}$$

where M_{ij} is a score measuring the j^{th} channel's impact on the i^{th} channel. Therefore M summaries the inter-channel dependency of features given a feature map. To perform the impact of attention map to original features, a further matrix multiplication between $\mathbf{M}$ and $\mathbf{F}$ is carried out and the result is reshaped back to 3-dimensional space $\mathbb{R}^{C\times H\times W}$, which has the same shape as the input tensor:

$$\mathbf{F}' = \mathbf{M}\cdot\mathbf{F}, \tag{4.4}$$

$$\mathbf{F}' \in \mathbb{R}^{C\times N} \xrightarrow{reshape} \mathbf{F}'' \in \mathbb{R}^{C\times H\times W}. \tag{4.5}$$

The output of channel attention module $\mathbf{C} \in \mathbb{R}^{C\times H\times W}$ combines this result with the original input feature map $\mathbf{B}$ with a trainable scalar parameter α using an element-wise sum operation, and α gradually learns a weight from 0:

$$\mathbf{C} = \alpha\cdot\mathbf{F}'' + \mathbf{B}. \tag{4.6}$$

The Eq. (4.6) shows that the final feature of each channel is a weighted sum of the features of all channels and original features, which models the long-range semantic dependencies between feature maps. Finally, the feature map $\mathbf{C} \in \mathbb{R}^{C \times H' \times W'}$ is fed into two more convolutional layers to generate the output feature map $\mathbf{D} \in \mathbb{R}^{C^* \times H' \times W'}$ of the CFAM module. Two convolutional layers at the beginning and the end of CFAM modules contain utmost importance since they help to mix the features coming from different backbones and having possibly different distributions. Without these convolutional layers, CFAM marginally improves the performance.

Such an architecture promotes the feature representativeness in terms of interdependencies among channels and thus the features from different branches can be aggregated reasonably and smoothly. Besides, the Gram matrix takes the whole feature map into consideration, where the dot product of each two flattened feature vectors presents the information about the relation between them. A larger product indicates that the features in these two channels are more correlated while a smaller product suggests that they are different from each other. For a given channel, we allocate more weights to the other channels that are more correlated and have more impact on it. By means of this mechanism, the contextual relationship is emphasized and feature discriminability is enhanced.

Bounding Box Regression

We follow the same guidelines of YOLO [209] for bounding box regression. A final convolutional layer with 1×1 kernels is applied to generate desired number of output channels. For each grid cell in $H' \times W'$, 5 prior anchors are selected by k-means technique on corresponding datasets with $NumCls$ class conditional action scores, 4 coordinates and confidence score making the final output size of YOWO $[(5 \times (NumCls + 5)) \times H' \times W']$. The regression of bounding boxes are then refined based on these anchors.

We have used input resolution of 224×224 for both training and testing time. Applying multi-scale training with different resolutions has not shown any performance improvement in our experiments. The loss function is defined similar to the original YOLOv2 network [209] except that we apply smooth L_1 loss with beta=1 for localization as in [211], which is given as follows:

$$L_{1,smooth}(x, y) = \begin{cases} 0.5(x-y)^2 & if|x-y| < 1, \\ |x-y| - 0.5 & otherwise, \end{cases} \tag{4.7}$$

where x and y refers to network prediction and ground truth, respectively. Smooth L_1 loss is less sensitive to outliers than the MSE loss and prevents exploding gradients in some cases. We still apply the MSE loss for confidence score, which is given as follows:

$$L_{MSE}(x, y) = (x - y)^2. \tag{4.8}$$

The final detection loss becomes the summation of individual coordinate losses for x, y, width and height; and confidence score loss, which is given as follows:

$$L_D = L_x + L_y + L_w + L_h + L_{conf}. \tag{4.9}$$

We have applied focal loss [212] for classification, which is given as follows:

$$L_{focal}(x, y) = y(1-x)^{\gamma} log(x) + (1-y)x^{\gamma} log(1-x), \tag{4.10}$$

where x is the softmaxed network prediction and y $\in \{0, 1\}$ is the ground truth class label. γ is the modulating factor, which reduces the loss of samples with high confidence (i.e. easy samples) and increases the loss of samples with low confidence (i.e. hard samples). However, the AVA dataset is a multi-label dataset where each person performs one pose action (e.g. walking, standing, etc.) and multiple human-human or human-object interaction actions. Therefore, we have applied softmax to pose classes and sigmoid to the interaction actions. Moreover, AVA is an unbalanced dataset and the modulating factor γ is not enough to tackle dataset imbalance. Therefore, we have used α-balanced variant of focal loss [212]. For the α term, we have used exponentials of class sample ratios.

The final loss that is used for the optimization of YOWO architecture is the summation of detection and classification loss, which is given as follows:

$$L_{final} = \lambda L_D + L_{Cls}, \tag{4.11}$$

where $\lambda = 0.5$ performs best in our experiments.

4.2.3.2 Linking Strategy

As we have already obtained frame-level action detections, the next step is to link these detected bounding boxes to construct action tubes in the whole video for UCF101-24 and J-HMDB-21 datasets. We make use of the linking algorithm described in [145, 146] to find the optimal video-level action detections.

Assume R_t and R_{t+1} are two regions from consecutive frames t and $t+1$, the linking score for an action class c is defined as follows:

$$\begin{aligned} s_c(R_t, R_{t+1}) \;=\; & \psi(ov) \cdot [s_c(R_t) + s_c(R_{t+1}) \\ & + \alpha \cdot s_c(R_t) \cdot s_c(R_{t+1}) \\ & + \beta \cdot ov(R_t, R_{t+1})], \end{aligned} \tag{4.12}$$

where $s_c(R_t)$, $s_c(R_{t+1})$ are class specific scores of regions R_t and R_{t+1}, ov is the Intersection over Union (IoU) of these two regions, α and β are scalars. $\psi(ov)$ is a constraint which is equal to 1 if an overlap exists ($ov > 0$), otherwise $\psi(ov)$ is equal to 0. We extend the linking score definition in [146] with an extra element $\alpha \cdot s_c(R_t) \cdot s_c(R_{t+1})$, which takes the dramatic change of scores between two successive frames into account and is able to improve the performance of video detection in experiments. After all the linking scores are computed, Viterbi algorithm is deployed to find the optimal path to generate action tubes.

4.2.3.3 Long-Term Feature Bank

Although YOWO's inference is online and causal with small a clip size, 16-frame input limits the temporal information required for action understanding. Therefore, we make

use of a long-term feature bank (LFB) similar to [202], which contains features coming from the 3D backbone at different timestamps. At inference time, 3D features centering the key frame are averaged and the resulting feature map is used as input to the CFAM block. LFB features are extracted for non-overlapping 8-frame clips using the pretrained 3D ResNeXt-101 backbone. We have used 8 features (if available) centering the key frame. So, a total number of 64 frames are utilized at inference time. The utilization of LFB increases action classification performance similar to the difference between clip accuracy and video accuracy in video datasets. However, the introduction of LFB makes the resulting architecture non-causal since future 3D features are used at inference time.

4.2.3.4 Implementation details

We initialize the 3D and 2D network parameters separately: 3D part with pretrained models on Kinetics [127] and 2D part with pretrained models on PASCAL VOC [213]. Although our architecture consists of 2D CNN and 3D CNN branches, the parameters can be updated jointly. We select the mini-batch stochastic gradient descent algorithm with momentum and weight decay strategy to optimize the loss function. The learning rate is initialized as 0.0001 and reduced with a factor of 0.5 after 30k, 40k, 50k, and 60k iterations. For the dataset UCF101-24, the training process is completed after 5 epochs while for J-HMDB-21 after 10 epochs. The complete architecture is implemented and trained end-to-end in PyTorch using a single Nvidia Titan XP GPU.

In the training, because of the small number of samples in J-HMDB-21, we freeze all parameters of the 3D CNN backbone, thus the convergence is faster and over-fitting risk can be reduced. In addition, we deploy several data augmentation techniques at training time such as flipping images horizontally in the clip, random scaling and random spatial cropping. During testing, only detected bounding boxes with a confidence score larger than threshold 0.25 are selected and then post-processed with non-maximum suppression with a threshold of 0.4 for the UCF101-24 and J-HMDB-21 datasets; and 0.5 for the AVA dataset.

YOWO architecture is implemented in PyTorch and trained with a single Nvidia Titan Xp GPU. We make our code publicly available at `https://github.com/wei-tim/YOWO` for the reproducibility of the results.

4.2.4 Experiments

To evaluate YOWO's performance, three popular and challenging action detection datasets are selected: UCF101-24 [6], J-HMDB-21 [173] and AVA [3]. Each of these datasets contains different characteristics, which are compared in Table 4.1. We follow the official evaluation metrics strictly to report the results and compare the performance of our method with the state-of-the-art.

Dataset	# of labelled person per frame	# of labels per person	background people	densely annotated
UCF101-24	one or two	one	✓	✗
J-HMDB-21	one	one	✗	✓
AVA	multiple	multiple	✗	✓*

Table 4.1: Comparison of evaluated datasets. Background people refers that in there are people in some of the frames who are not annotated. *AVA is densely annotated with 1Hz rate.

4.2.4.1 Datasets and evaluation metrics

Datasets: The details of the UCF101-24, J-HMDB-21, and AVA datasets can be found in Section 2.5. For UCF101-24 and J-HMDB-21 datasets, we perform all the experiments on the first split. For AVA dataset, we report results on version 2.2, if not stated otherwise.

Evaluation metrics: We employ two popular metrics that are actively used for the task of spatiotemporal action detection to generate convincing evaluations. Following strictly the rule applied by the PASCAL VOC 2012 metric [214], frame-mAP measures the area under the precision-recall curve of the detections for each frame. On the other hand, video-mAP focuses on the action tubes [145]. If the mean of IoU scores between detected bounded boxes and ground truth across all frames of the video is greater than a threshold and the action label is correctly predicted in the meanwhile, then this detected tube is regarded as a correct instance. Finally, the average precision for each class is computed and the average of all classes is reported. For the AVA dataset, we only use frame-mAP with IoU threshold of 0.5 since annotations are sparsely provided with 1 Hz.

4.2.4.2 Ablation study

3D network, 2D network or both? Depending only on its own, neither 3D CNN nor 2D CNN can solve the spatiotemporal localization task independently. However, if they operate simultaneously, there is potential to benefit from one another. Results on comparing the performance of different architectures are reported in Table 4.2. We first observe that a single 2D network can not provide a satisfying result since it does not take temporal information into account. A single 3D network is better at capturing motion information and the fusion of 2D and 3D networks (simple concatenation) can improve the performance by around 3%, 6% and 2% mAP compared to 3D network on UCF101-24, J-HMDB-21 and AVA datasets, respectively. This indicates that 2D CNN learns finer spatial features and 3D CNN concentrates more on the motion process yet the spatial drift of actions in the clip may lead to a lower localization accuracy. It is also shown that the CFAM module further boosts the performance from 73.8% to 79.2% on UCF101-24, from 47.1% to 64.9% on J-HMDB-21 and from 16.0% to 16.4% on AVA dataset. This clearly shows the importance of the attention mechanism which

Model	UCF101-24	J-HMDB-21	AVA
2D	61.6	36.0	13.2
3D	70.5	41.5	13.7
2D + 3D	73.8	47.1	16.0
2D + 3D + CFAM	**79.2**	**64.9**	**16.4**

Table 4.2: Frame-mAP @ IoU 0.5 results on UCF101-24, J-HMDB-21 and AVA datasets for different models. For all architectures, the input to 3D CNNs is 8 frames clips with downsampling of 1.

	Model	Localization (recall)	Classif.
UCF101-24	2D	91.7	85.9
	3D	90.8	92.9
	2D + 3D	93.2	93.7
	2D + 3D + CFAM	**93.5**	**94.5**
J-HMDB-21	2D	94.3	50.6
	3D	76.3	69.3
	2D + 3D	94.5	63.0
	2D + 3D + CFAM	**97.3**	**76.1**

Table 4.3: Localization @ IoU 0.5 (recall) and classification results on UCF101-24 and J-HMDB-21 datasets. For all architectures, the input to 3D CNNs is 8 frames clips with downsampling of 1.

strengthens the inter-dependencies among channels and helps aggregating features more reasonably.

Moreover, in order to explore the impact of each 2D CNN, 3D CNN and CFAM blocks, we investigate the localization and the classification performance of different architectures, which is given in Table 4.3. For localization, we look at the recall value, which is the ratio of the number of correctly localized actions to the total number of ground truth actions. For classification, we look at the classification accuracy of the correctly localized detections. For this analysis, we have excluded the AVA dataset since it contains multiple actions per person, hence classification accuracy cannot be calculated. For both UCF101-24 and J-HMDB-21 datasets, the 2D backbone is better at localization while the 3D backbone performs better at classification. It is also obvious that the CFAM module boosts both localization and classification performance.

We have also visualized the activations maps [215] for 2D and 3D backbones of the trained model, which is shown in Fig. 4.5. Conforming our findings in Table 4.3, the 3D backbone focuses on the parts of the clip where a motion is occurring and the 2D

Input	UCF101-24	J-HMDB-21	AVA
8-frames (d=1)	79.2	64.9	16.4
8-frames (d=2)	78.5	61.5	16.1
8-frames (d=3)	78.4	61.0	16.0
16-frames (d=1)	80.4	**74.4**	17.9
16-frames (d=2)	79.0	71.4	17.2

Table 4.4: Frame-mAP @ IoU 0.5 results on UCF101-24, J-HMDB-21 and AVA datasets for different clip lengths and different downsampling rates d.

backbone focuses on fine spatial information on complete body parts of people. This validates that the backbones of YOWO extract complementary features.

How many frames are suitable for temporal information? For the 3D CNN branch, different clip lengths with different downsampling rates can change the performance of overall YOWO architecture [30]. Therefore, we conduct experiments with 8-frames and 16-frames clips with different downsampling rates, which is given in Table 4.4. For example, 8-frames (d=3) refers to selecting 8 frames from 24 frames window with a downsampling rate of 3. Specifically, we compare three downsampling rates $d = 1, 2, 3$ for clip length 8-frames and two downsampling rates $d = 1, 2$ for 16-frames clip length. As expected, we observe that the framework with the input of 16

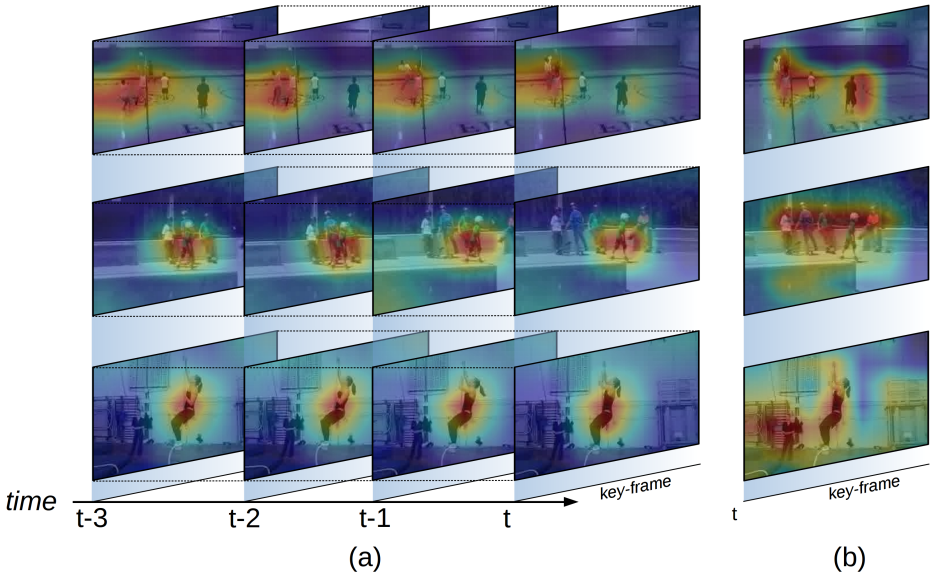


Figure 4.5: Activation maps for (a) 3D CNN backbone and (b) 2D CNN backbone. 3D CNN backbone focuses on areas where there is a movement/action happening, whereas 2D CNN backbone focuses on all the people in the key frame. Examples are volleyball spiking (top), skate boarding (middle) and rope climbing (bottom).

Model	GFLOPs	Frame-mAP (@ IoU 0.5)		
		UCF101-24	J-HMDB-21	AVA
3D-ResNext-101	38.6	**80.4**	**74.4**	**17.9**
3D-ResNet-101	54.7	78.1	70.8	17.7
3D-ResNet-50	39.3	77.8	61.3	17.1
3D-ResNet-18	33.3	72.6	57.5	15.1
3D-ShuffleNetV1 2.0x	2.2	71.3	54.8	16.1
3D-ShuffleNetV2 2.0x	1.8	71.4	55.3	15.1
3D-MobileNetV1 2.0x	2.6	67.3	48.5	14.9
3D-MobileNetV2 1.0x	2.2	66.6	52.5	16.1

Table 4.5: Performance comparison of different 3D backbones on UCF101-24, J-HMDB-21 and AVA datasets. For all architectures, Darknet-19 is used as 2D backbone. The number of floating point operation (FLOPs) are calculated for corresponding 3D backbones for 16 frames (d=1) clips with spatial resolution of 224 × 224.

frames performs better than 8 frames since the longer frame sequence contains more temporal information. However, as the downsampling rate is increased, the performance becomes worse. We conjecture that downsampling hinders capturing motion patterns properly and too long sequences may break the temporal contextual relationship. Especially for some quick motion classes, a long sequence may contain several unrelated frames, which can be viewed as noise.

Is it possible to save model complexity with more efficient networks? We have chosen 3D-ResNext-101 [131] since it has multiple cardinalities thus is able to learn more complicated features. However, it is a heavy backbone with a huge number of parameters and computational complexity. Therefore, we have replaced the 3D backbone with 3D-ResNet for different depths and with some other resource efficient 3D CNN architectures [21]. Table 4.5 reports the achieved performance on all three datasets together with the number of floating point operations (FLOPs) for each 3D backbone. We find that even with lightweight architecture in 3D backbones, our framework is still better than the 2D network. However, Table 4.5 clearly shows the importance of the 3D backbone. Stronger the 3D CNN architecture we use, better the achieved results.

4.2.4.3 State-of-the-art comparison

We have compared YOWO with other state-of-the-art architectures on J-HMDB-21, UCF101-24 and AVA datasets. For the sake of fairness, we have excluded VideoCapsuleNet [150] as it uses different video-mAP calculations without constructing action tubes via some linking strategies. However, YOWO still performs around 9% and 8% better than VideoCapsuleNet in terms of frame-mAP @ 0.5 IoU on J-HMDB-21 and UCF101-24, respectively.

Method	Frame-mAP	Video-mAP		
		0.2	0.5	0.75
Peng w/o MR [146]	56.9	71.1	70.6	48.2
Peng w/ MR [146]	58.5	74.3	73.1	-
ROAD [149]	-	73.8	72.0	44.5
T-CNN [147]	61.3	78.4	76.9	-
ACT [216]	65.7	74.2	73.7	52.1
P3D-CTN [217]	71.1	84.0	80.5	-
TPnet [218]	-	74.8	74.1	**61.3**
YOWO (16-frame)	74.4	87.8	85.7	58.1
YOWO+LFB*	**75.7**	**88.3**	**85.9**	58.6

Table 4.6: Comparison with state-of-the-art methods on the J-HMDB-21 dataset. Results are reported for frame-mAP under IoU threshold of 0.5 and video-mAP under different IoU thresholds. * version of YOWO is non-causal.

Performance comparison on the J-HMDB-21 dataset. YOWO is compared with the previous state-of-the-art methods on J-HMDB-21 in Table 4.6. Using the standard metrics, we report the frame-mAP at the IoU threshold of 0.5 and the video-mAP at various IoU thresholds. YOWO (16-frame) consistently outperforms the state-of-the-art results on dataset J-HMDB-21, with a frame-mAP improvement of 3.3% and a video-mAP improvement of 3.8%, 5.2% at IoU thresholds of 0.2 and 0.5, respectively. The utilization of LFB brings further improvements to the performance. However, this improvement is marginal since the video duration of videos of the J-HMDB-21 dataset is maximum 40 frames.

Performance comparison on the UCF101-24 dataset. Table 4.7 presents the comparison of YOWO with the state-of-the-art methods on UCF101-24. YOWO (16-frame) achieves 80.4% frame-mAP, which is significantly better than the others by preceding the second best result with 5.4% improvement. As for video-mAP, our framework also produces very competitive results even though we just utilize a simple linking strategy. Utilization of LFB brings considerable improvement this time since the duration of UCF101-24 videos is much bigger than J-HMDB-21 videos. LFB further increases frame-mAP performance by around 7%.

Performance comparison on the AVA dataset. We have compared the performance of YOWO on the AVA dataset in Table 4.8. YOWO is currently the first and only single-stage architecture, which provides competitive results on the AVA dataset. All the methods outperforming YOWO are non-causal (i.e. utilizing future frames) and multi-stage architectures mostly utilizing Faster-RCNN architecture. Moreover, these methods either require high-resolution input such as 600 pixels for [222] and 400 pixels for [117], or strong and computationally heavy SlowFast architecture as 3D CNN backbone such as [130]. On the other hand, YOWO operates only on the current and previous frames

Method	Frame-mAP	Video-mAP		
		0.1	**0.2**	**0.5**
Peng w/o MR [146]	64.8	49.5	41.2	-
Peng w/ MR [146]	65.7	50.4	42.3	-
ROAD [149]	-	-	73.5	46.3
T-CNN [147]	41.4	51.3	47.1	-
ACT [216]	69.5	-	77.2	51.4
MPS [219]	-	82.4	72.9	41.1
STEP [220]	75.0	83.1	76.6	-
YOWO (16-frame)	80.4	82.5	75.8	48.8
YOWO+LFB*	**87.3**	**86.1**	**78.6**	**53.1**

Table 4.7: Comparison with state-of-the-art methods on the UCF101-24 dataset. Results are reported for frame-mAP under IoU threshold of 0.5 and video-mAP under different IoU thresholds. * version of YOWO is non-causal.

Method	Single Stage	Input	AVA	Pretrain	mAP
I3D [3]	✗	V+F		K400	15.6
ACRN, S3D [221]	✗	V+F		K400	17.4
STEP, I3D [220]	✗	V+F		K400	18.6
RTPR [222]	✗	V+F		ImageNet	22.3
Action Transformer, I3D [117]	✗	V		K400	25.0
LFB, R101+NL [202]	✗	V	v2.1	K400	**27.4**
SlowFast, R101, 8x8 [130]	✗	V		K400	26.3
YOWO (8-frame)	✓	V		K400	15.7
YOWO (16-frame)	✓	V		K400	17.2
YOWO (32-frame)	✓	V		K400	18.3
YOWO+LFB*	✓	V		K400	**19.2**
SlowFast, R101+NL, 8x8 [130]	✗	V		K600	**29.0**
YOWO (8-frame)	✓	V		K400	16.4
YOWO (16-frame)	✓	V	v2.2	K400	17.9
YOWO (32-frame)	✓	V		K400	19.1
YOWO+LFB*	✓	V		K400	**20.2**

Table 4.8: Comparison with state-of-the-art methods on the AVA dataset. Results are reported for frame-mAP under IoU threshold of 0.5. * version of YOWO is non-causal.

(i.e. causal) with an input resolution of 224×224. Increasing clip size from 8-frames to 32-frames brings an improvement of almost 3% mAP. Utilization of LFB further improves the performance by around 1% maP. We also evaluate performance of YOWO

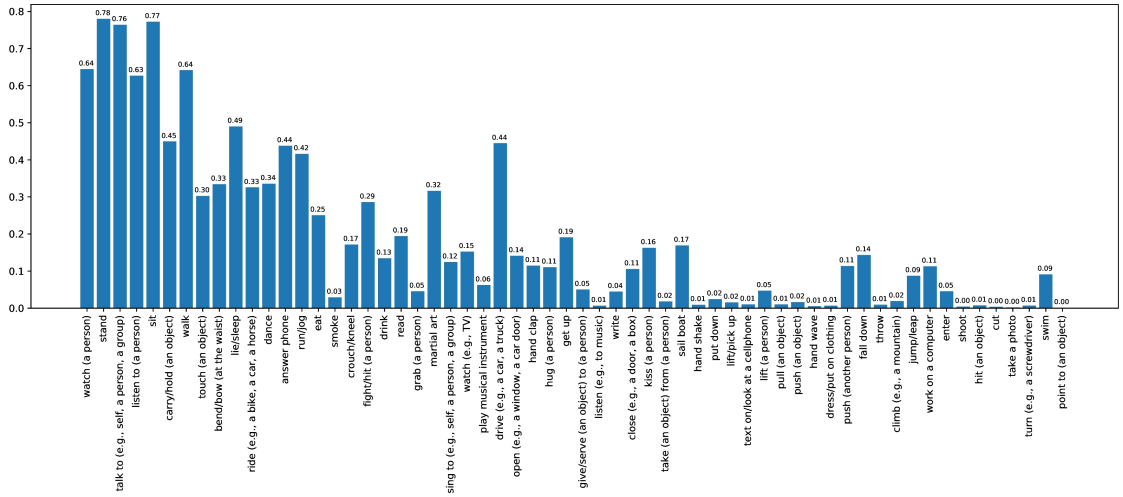


Figure 4.6: Performance of YOWO (32-frames, d=1) on each class of AVA dataset v2.2. Classes are sorted by number of training samples in decreasing order.

Model	Speed (fps)	F-mAP	V-mAP
Saha *et al.* [148]	4	-	36.4
ROAD (A) [149]	40	-	40.9
ROAD (A+RTF)[149]	28	-	41.9
ROAD (A+AF)[149]	7	-	46.3
YOWO (8-frames, d=1)	**62**	79.2	47.6
YOWO (16-frames, d=1)	34	**80.4**	**48.8**

Table 4.9: Runtime and performance comparison on dataset UCF101-24 for F-mAP and V-mAP at 0.5 IoU threshold. For YOWO, ResNeXt-101 is used in its 3D backbone.

(32-frames, d=1) per each class in Fig. 4.6. The classes are sorted by the number of training samples. Although we observe some correlation with the amount of training data, there exist some classes with enough data with poor performance such as smoking.

Runtime comparison Most of the state-of-the-art methods are two-stage architectures, which are computationally expensive to run in real-time. YOWO is a unified architecture, which can be trained end-to-end. In addition, we do not employ optical flow, which is computationally burdensome. In Table 4.9, we compare the runtime performance of YOWO with other state-of-the-art methods. YOWO's speed is calculated in terms of frames per second (fps) on a single Nvidia Titan Xp GPU with a batch size of 8. It must be noted that YOWO's 2D and 3D backbones can be replaced with any arbitrary CNN model according to desired runtime performance. Moreover, additional new backbones can be easily introduced for different information source such as *depth* or *infrared* modalities. The only thing to do is the modification of CFAM block in order to accommodate new features.

Figure 4.7: Visualization of action localizations for UCF101-24 and J-HMDB-21 datasets. Red bounding boxes are ground truth while green and orange are true and false positive localizations, respectively.

4.2.4.4 Model visualization

In general, YOWO architecture performs a decent job at localizing actions in videos, which is illustrated in Fig. 4.7 for UCF101-24 and J-HMDB-21 dataset; and in Fig. 4.8 for AVA dataset. However, YOWO also has some drawbacks. Firstly, since YOWO produces its predictions according to all the information available at the key frame and the clip, it sometimes makes some false positive detections before the actions are performed. For example, in Fig. 4.7 first row last image, YOWO sees a person holding a ball at a basketball court and detects him very confidently although he is not shooting the ball yet. Secondly, YOWO needs enough temporal content to make correct action localization. If a person starts performing an action suddenly, localization at initial frames lacks temporal content and false actions are recognized consequently, as in Fig. 4.7 second row last image (climbing stair instead of running). Similarly, in the bottom row right-most image in Fig. 4.8, the processed clip and key frame does not contain pose information of the person, hence YOWO cannot confidently deduce if the person is sitting or standing. Results in Table 4.4 confirm that increasing clip length increases the available temporal information and consequently increases YOWO's performance. LFB is also leveraged for the purpose of increasing temporal content.

4.2.5 Summary

This section presented a novel unified architecture for spatiotemporal action localization in video streams. Our approach, YOWO, models the spatiotemporal context from successive frames for action understanding while extracting the fine spatial information from key frame to address the localization task in parallel. In addition, we make use of a channel fusion and attention mechanism for effective aggregation of these two kinds of information. Since we do not separate human detection and action classification procedures, the whole network can be optimized by a joint loss in an end-to-end framework. We have carried out a series of comparative evaluations on three challenging datasets, UCF101-24, J-HMDB-21 and AVA, each

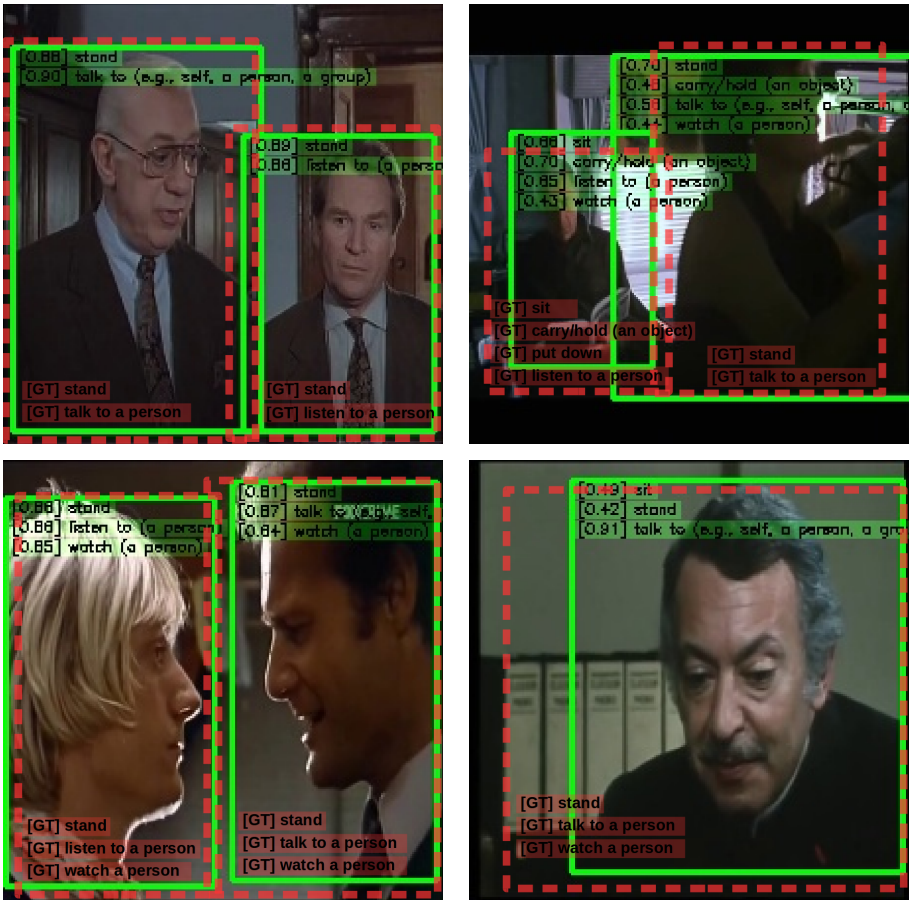


Figure 4.8: Visualization of action localizations for AVA dataset. Red dashed bounding boxes are ground truth while green bounding boxes are YOWO predictions.

having different characteristics. Our approach outperforms the other state-of-the-art results on UCF101-24 and J-HMDB-21 datasets while achieving competitive results on the AVA dataset. Moreover, YOWO is a causal architecture and can be operated in real-time, which makes it possible to deploy YOWO on mobile devices.

4.3 Lightweight Clip-Level Feature Extraction

This section addresses the complexity drawback of 3D CNNs and presents several resource efficient 3D CNN architectures.

Recently, convolutional neural networks with 3D kernels (3D CNNs) have been very popular in the computer vision community as a result of their superior ability to extract spatiotemporal features within video frames compared to 2D CNNs. Although there have been great advances recently to build resource efficient 2D CNN architectures considering memory and power budget, there is hardly any similar resource efficient architectures for 3D CNNs. In this section, we have converted various well-known resource efficient 2D CNNs to 3D CNNs and evaluated their performance on three major benchmarks in terms of classification accuracy for different complexity levels. We have experimented on (1) Kinetics-600 dataset to inspect their capacity to learn, (2) Jester dataset to inspect their ability to capture motion patterns, and (3) UCF101 dataset to inspect the applicability of transfer learning. We have evaluated the runtime performance of each model on a single Titan XP GPU and a Jetson TX2 embedded system. The results of this study show that these models can be utilized for different types of real-world applications since they provide real-time performance with considerable accuracies and memory usage. Our analysis on different complexity levels shows that the resource efficient 3D CNNs should not be designed too shallow or narrow in order to save complexity. This section is based on our publication *Resource Efficient 3D Convolutional Neural Networks* [21].

4.3.1 Motivation

Ever since AlexNet [14] won the ImageNet Challenge (ILSVRC 2012 [79]), convolutional neural networks (CNNs) have dominated the majority of the computer vision tasks. Then the primary trend has been more on creating deeper and wider CNN architectures to achieve higher accuracies [88, 223, 224]. However, in real-world computer vision applications such as face recognition, robot navigation, and augmented reality, the tasks need to be carried out under runtime constraints on a computationally limited platform. Only recently, there has been a rising interest in building resource efficient convolutional neural networks but it is limited with 2-dimensional kernels (2D) [87, 174, 175, 176, 177].

The same history is repeating for CNNs with 3-dimensional (3D) kernels [131]. Since the large video datasets became available, the primary trend for video recognition tasks is again to achieve higher accuracies by building deeper and wider architectures [225, 129, 128, 131, 130]. Considering the fact that 3D CNNs achieve better performance for video recognition tasks compared to 2D CNNs [127], it is very likely that this 3D CNN architecture search will continue until the achieved accuracies saturate. However, real-world applications still require resource efficient 3D CNN architectures taking runtime, memory and power budget into account. This work aims to fill this research gap.

In this section, we first have created the 3D versions of the well-known 2D resource efficient architectures: SqueezeNet, MobileNet, ShuffleNet, MobileNetV2 and

ShuffleNetV2. We have evaluated the performance of these architectures on three publicly available benchmarks:

(1) Kinetics-600 dataset[127] to learn models' capacities.

(2) Jester dataset [4] to learn how well the models capture the motion.

(3) UCF101 dataset [6] to evaluate the applicability of transfer learning for each model.

The computational complexity of the implemented architectures is measured in terms of floating point operations (FLOPs), which is a widely used metric among resource efficient architectures. In this section, the number of FLOPs refers to the number of multiply-adds. However, as highlighted by [176], the number of FLOPs is an indirect metric, which does not give an actual performance indication like speed or latency. Therefore, for all the implemented architectures we have also evaluated their runtime performance on two different platforms, which are Nvidia Titan XP GPU and Jetson TX2 embedded system-on-module (SoM) with integrated 256-core Pascal GPU.

4.3.2 Related Work

The literature review on action and gesture recognition is already provided in Section 2.1 and Section 2.2, respectively. Here, we will provide related work on resource efficient CNN architectures.

Lately, there is a rising interest in building small and efficient neural networks [87, 174, 176, 226, 227, 228]. The common approaches used for this objective can be categorized under two categories: (i) Accelerating the pretrained networks, or (ii) directly constructing small networks by manipulating kernels. For the first one, [228, 229, 230, 231] proposes to prune either network connections or channels without reducing the performance of pretrained models. Additionally, many other methods apply quantization [226, 232, 227] or factorization [233, 234, 235] for the same objective. However, our focus is on the second one for directly designing small and resource efficient 3D CNN architectures.

Current well-known resource efficient CNN architectures are all constructed with 2D convolutional kernels and benchmarked at ImageNet. SqueezeNet [87] reduced the number of parameters and computation while maintaining the classification performance. MobileNet [174] makes use of depthwise separable convolutions to construct lightweight deep neural networks. The depthwise separable convolutions factorize the standard convolutions into a depthwise convolution followed by a 1x1 pointwise convolution. Compared to standard convolutions, depthwise separable convolutions use between 8 to 9 times fewer parameters and computations. ShuffleNet [175] proposes to use pointwise group convolutions and channel shuffle in order to reduce computational cost. MobileNetv2 [177] makes use of the inverted residual structure where the intermediate expansion layer uses depthwise convolutions. ShuffleNetV2 [176] builds on top of ShuffleNet [175] using channel split together with channel shuffle which realizes a feature reuse pattern.

These architectures intensively make use of group convolutions and depthwise separable convolutions. Group convolutions are first introduced in AlexNet [14] and efficiently utilized in ResNeXt [90]. Depthwise separable convolutions are introduced in Xception [236] and they are the main building blocks for the majority of lightweight architectures.

All of the above-mentioned resource efficient architectures are 2D CNNs. They are designed to operate on static images and evaluated on a very large benchmark (i.e., ImageNet). To the best of our knowledge, this is the first work that proposes and evaluates resource efficient 3D CNNs on large-scale video benchmarks.

Up to now, nearly all the 3D CNN architectures in the literature are heavyweight, requiring 10s and even 100s billions of floating point operations (FLOPs). Moreover, the majority of these architectures also use optical flow modality, which increases the complexity even further. Our focus in this work is to evaluate 3D CNNs having less than 500 MFLOPs. Consequently, we have implemented the 3D version of SqueezeNet [87], MobileNet [174], MobileNetV2 [177], ShuffleNet [175] and ShuffleNetV2 [176] for 4 different complexity levels and then evaluated them on 3 different video benchmarks. We have evaluated our architectures only using the RGB modality without computing costly optical flow modality.

4.3.3 Methodology

In this section, we explain the details of the resource efficient 3D CNN architectures that have been proposed and evaluated within the scope of this work. We initially introduce the 3D versions of the well-known resource efficient 2D CNN architectures by explaining their building blocks and networks structures. Then we compare these models in terms of number of layers, nonlinearities, and skip connections. We conclude with the training details of the models.

4.3.3.1 3D Versions of Well-known Architectures

In this section, we give the implementation details of our resource efficient architectures with 3-dimensional kernels, which are converted from well-known resource efficient 2D CNN architectures. The main building blocks of each architecture are depicted in Fig. 4.9. The input is always considered as a clip of 16 frames with a spatial resolution of 112 pixels. For all of the 3D CNN architectures, first convolutions always apply stride of (1,2,2). For the rest of the architectures, the depth dimension is reduced together with spatial dimensions.

3D-SqueezeNet

SqueezeNet [87] is considered as one of very first resource efficient CNN architectures with notable accuracy performance. It achieves the AlexNet [14]-level accuracy with 50 times fewer parameters and less than 0.5 MB model size.

The main building block of SqueezeNet is the Fire block whose 3D version is depicted in Fig. 4.9 (a). As illustrated in Table 4.10, 3D-SqueezeNet begins with a convolution

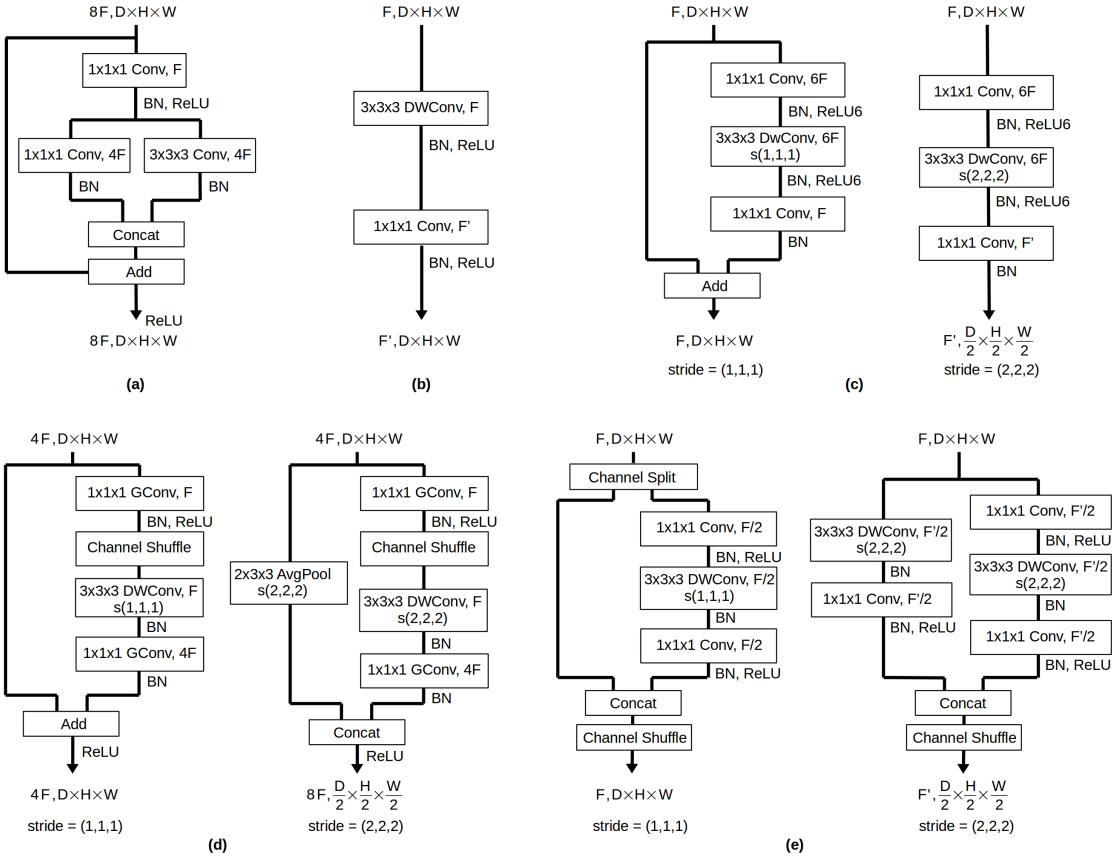


Figure 4.9: Main building block for each resource efficient 3D CNN architecture. F is the number of feature maps and D × H × W stands for Depth × Height × Width for the input and output volumes. *DWConv* and *GConv* stand for depthwise and group convolution, respectively. *BN* and *ReLU(6)* stand for Batch Normalization and Rectified Linear Unit (capped at 6), respectively. **(a)** SqueezeNet's Fire block; **(b)** MobileNet block; **(c)** left: MobileNetv2 block, right: MobileNetv2 block with spatiotemporal downsampling (2x); **(d)** left: ShuffleNet block, right: ShuffleNet block with spatiotemporal downsampling (2x); **(e)** left: ShuffleNetv2 block, right: ShuffleNetv2 block with spatiotemporal downsampling (2x).

layer (Conv1), followed by 8 Fire blocks (Fire-2-9), ending with a final convolutional layer (Conv10).

In our experiments, we use SqueezeNet with simple bypass since it achieves the best result in its 2D version for ImageNet. SqueezeNet does not apply depthwise convolutions which is the main building block for the majority of resource efficient architectures. Instead, it uses three strategies to reduce the number of parameters while maintaining accuracy: (i) Replacing 3x3 filters with 1x1 filters, (ii) decreasing the number of input channels to 3x3 filters, and (iii) downsampling late in the network so that convolution layers have large activation maps. Moreover, compared to other resource efficient architectures, SqueezeNet cannot be modified with *width_multiplier*

Layer/Stride	Filter size	Output size
Input clip		3x16x112x112
Conv1/s(1,2,2)	3x3x3	64x16x56x56
MaxPool/s(2,2,2)	3x3x3	64x8x28x28
Fire2		128x8x28x28
Fire3		128x8x28x28
MaxPool/s(2,2,2)	3x3x3	128x4x14x14
Fire4		256x4x14x14
Fire5		256x4x14x14
MaxPool/s(2,2,2)	3x3x3	256x2x7x7
Fire6		384x2x7x7
Fire7		384x2x7x7
MaxPool/s(2,2,2)	3x3x3	384x1x4x4
Fire8		512x1x4x4
Fire9		512x1x4x4
Conv10/s(1,1,1)	1x1x1	*NumCls*x1x4x4
AvgPool/s(1,1,1)	1x4x4	*NumCls*

Table 4.10: 3D-SqueezeNet architecture. Details of Fire block is given in Fig. 4.9 (a).

parameter resulting in different complexities. Therefore, it is only experimented with its default configuration.

3D-MobileNetV1

MobileNets [174] apply depthwise separable convolutions which have a form that factorizes a standard convolution into a depthwise convolution and 1×1 convolution, which is called as pointwise convolution. In MobileNet architectures, the depthwise convolution applies a single filter to each input channel and then the pointwise convolution applies a 1×1 convolution to combine the outputs of the depthwise convolution. Different from the standard convolution, the depthwise separable convolution involves two layers, which separates filtering and combining operations as illustrated in Fig. 4.9 (b). This process helps to decrease computation time and model size significantly. Unlike all recent popular CNN architectures, MobileNet does not contain skip connections. Therefore, the depth of the network cannot be increased too much which hinders gradient flow.

Table 4.11 shows the details of the 3D-MobileNet architecture. 3D-MobileNet begins with a convolutional layer, followed by 13 MobileNet blocks, ending with a linear layer. MobileNet has 28 layers in case the depthwise and pointwise convolutions in each MobileNet block are counted as separate layers.

Layer/Stride	Repeat	Output size
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	32x16x56x56
Block/s(2x2x2)	1	64x8x28x28
Block/s(2x2x2)	1	128x4x14x14
Block/s(1x1x1)	1	128x4x14x14
Block/s(2x2x2)	1	256x2x7x7
Block/s(1x1x1)	1	256x2x7x7
Block/s(2x2x2)	1	512x1x4x4
Block/s(1x1x1)	5	512x1x4x4
Block/s(1x1x1)	1	1024x1x4x4
Block/s(1x1x1)	1	1024x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	1024x1x1x1
Linear(1024x*NumCls*)	1	*NumCls*

Table 4.11: 3D-MobileNet architecture. Details of Block is given in Fig. 4.9 (b).

3D-MobileNetV2

MobileNetV2 [177] is another 2D resource efficient architecture. It builds upon the main idea of MobileNetV1 by using depthwise separable convolutions; however, it introduces two new components: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. The idea behind 1) is both keeping the

Layer/Stride	Repeat	Output size
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	32x16x56x56
Block/s(1x1x1)	1	16x16x56x56
Block/s(2x2x2)	2	24x8x28x28
Block/s(2x2x2)	3	32x4x14x14
Block/s(2x2x2)	4	64x2x7x7
Block/s(1x1x1)	3	96x2x7x7
Block/s(2x2x2)	3	160x1x4x4
Block/s(1x1x1)	1	320x1x4x4
Conv(1x1x1)/s(1,1,1)	1	1280x1x4x4
AvgPool/s(1,1,1)	1	1024x1x1x1
Linear	1	*NumCls*

Table 4.12: 3D-MobileNetV2 architecture. Block is inverted residual block whose details are given in Fig. 4.9 (c) with stride 1 (left) and spatio temporal 2x downsampling (right).

Layer/Stride	Repeat	Output size (groups=3)
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	24x16x56x56
MaxPool(3x3x3)/s(2,2,2)	1	24x8x28x28
Block/s(2x2x2)	1	240x4x14x14
Block/s(1x1x1)	3	240x4x14x14
Block/s(2x2x2)	1	480x2x7x7
Block/s(1x1x1)	7	480x2x7x7
Block/s(2x2x2)	1	960x1x4x4
Block/s(1x1x1)	3	960x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	960x1x1x1
Linear	1	*NumCls*

Table 4.13: 3D-ShuffleNet architecture. Its main building block is given in Fig. 4.9 (d) with stride 1 (left) and spatio temporal 2x downsampling (right).

size of the model low by decreasing the number of channels and extracting as much as information by applying depthwise convolution after decompressing the data. This convolutional module allows reducing memory usage during inference. On the other hand, 2) allows training faster and constructs deeper models like ResNet architectures [88].

Fig. 4.9 (c) shows the MobileNetV2 block. Table 4.12 shows the layers of 3D-MobileNetV2 architecture. 3D-MobileNetV2 begins with a convolutional layer, followed by 17 MobileNetV2 blocks, and then a convolutional layer and finally ending with a linear layer.

3D-ShuffleNetV1

According to [175], ShuffleNet provides superior performance compared to MobileNet [174] by a significant margin, which is reported as absolute 7.8% lower ImageNet top-1 error at level of 40 MFLOPs. The model is also reported to achieve 13× actual speedup over AlexNet while maintaining comparable accuracy.

The architecture uses two new operations, which are pointwise group convolution and channel shuffle which is depicted in Fig. 4.9 (d).

As illustrated in Table 4.13, 3D-ShuffleNet begins with a convolutional layer followed by 16 ShuffleNet blocks, which are grouped into three stages. In each stage, the number of output channels is kept the same with the applied ShuffleNet blocks. For the next stage, the output channels are doubled and the spatial and depth dimensions are reduced to half. ShuffleNet architecture ends with a final linear layer. In ShuffleNet units, group number g controls the connection sparsity of pointwise convolutions. In this study, the group number is selected as 3.

LayerStride	Repeat	Output size
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	24x16x56x56
MaxPool(3x3x3)/s(2,2,2)	1	24x8x28x28
Block/s(2x2x2)	1	c_1x4x14x14
Block/s(1x1x1)	3	c_1x4x14x14
Block/s(2x2x2)	1	c_2x2x7x7
Block/s(1x1x1)	7	c_2x2x7x7
Block/s(2x2x2)	1	c_3x1x4x4
Block/s(1x1x1)	3	c_3x1x4x4
Conv(1x1x1)/s(1,1,1)	1	c_4x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	c_4x1x1x1
Linear	1	*NumCls*

Table 4.14: 3D-ShuffleNetV2 architecture. Its main building block is given in Fig. 4.9 (e) with stride 1 (left) and spatio temporal 2x downsampling (right). The number of channels (c_1, c_2, c_3, c_4) for different complexities are given in Table 4.15.

3D-ShuffleNetV2

In ShuffleNetV2 [176] architecture, channel split operator is introduced different from V1. As illustrated in Fig. 4.9 (e), at the beginning of each block, the input of c feature channels are split into two branches with c-$c^{'}$ and $c^{'}$ channels, respectively. One branch remains as identity, and the other branch includes three convolutions with the same input and output channels. Different from ShuffleNet, the two 1×1 convolutions are not groupwise. After the convolutions, the two branches are concatenated and the number of channels keeps the same. At the end of the block, the channel shuffle operation is applied to enable information communication between the two branches.

Table 4.14 shows the layers of 3D-ShuffleNetV2 architecture. 3D-ShuffleNetV2 architecture begins with a convolutional layer, followed by 16 ShuffleNetV2 blocks, then a convolutional layer, and finally ending with a linear layer. Similar to 3D-ShuffleNet, the stack of blocks is grouped into three stages, and at each stage, the number of output channels is kept the same while with the next stage, they are doubled. Different from the 3D-ShuffleNet, the number of channels in each stage is not fixed. Table 4.15 shows the number of channels (c_1, c_2, c_3, c_4) for different levels of complexities. Also, in 3D-ShuffleNet, the number of output channels in the final layer (c_4) is the same after the third stage, whereas in 3D-ShuffleNetV2, different number of output channels are selected for different levels of complexities (Table 4.15).

4.3.3.2 Comperative Analysis

In this section, we compare the experimented architectures according to the number of layers, nonlinearities, and skip connections. These design criteria play an important

	Output channels				
	0.25x	**0.5x**	**1.0x**	**1.5x**	**2.0x**
c_1	32	48	116	176	244
c_2	64	96	232	352	488
c_3	128	192	464	704	976
c_4	1024	1024	1024	1024	2048

Table 4.15: The number of channels used in 3D-ShuffleNetv2 architecture for different levels of complexities.

Model	**Number of**		
	layers	**non-lin.**	**skip-con.**
3D-SqueezeNet	18	18	4
3D-ShuffleNetV1	50	33	16
3D-ShuffleNetV2	51	34	16
3D-MobileNetV1	28	27	0
3D-MobileNetV2	53	35	10

Table 4.16: Comparison of resource efficient 3D architectures according to the number of layers, non-linearity and skip-connections.

role in the performance of the architectures. Comparison of the architectures is given in Table 4.16. For the number of layers, we counted the convolutional and linear layers. For the skip-connections, we have counted the addition or concatenation operations in the architectures. Finally, for the number of non-linearity, we have counted the ReLU operations in one inference time since it is the only non-linearity used for all the architectures.

It is noticeable that comparatively earlier architectures (i.e. SqueezeNet and MobileNetV1) have a smaller number of layers, non-linearity, and skip-connections. On the other hand, recent resource efficient architectures (i.e. ShuffleNetV1, ShuffleNetV2 and MobileNetV2) are deeper, in the order of 50 layers and 30 non-linearity. Corollary, they require more skip connections in order to facilitate a better gradient update mechanism.

4.3.3.3 Training Details

Learning: For the training of the architectures, Stochastic Gradient Descent (SGD) with standard categorical cross-entropy loss is applied. For the mini-batch size of SGD, the largest fitting batch size is selected, which is usually in the order of 128 videos. The momentum, dampening, and weight decay are set to 0.9, 0.9 and $1\text{x}10^{-3}$, respectively. When the networks are trained from scratch, the learning rate is initialized with 0.1 and reduced 3 times with a factor of 10^{-1} when the validation loss converges. For the

training of the UCF101 benchmark, we have used the pretrained models of Kinetics-600. We have frozen the network parameters and fine-tuned only the last layer. For fine-tuning, we start with a learning rate of 0.01 and reduce it two times after 30^{th} and 45^{th} epochs with a factor of 10^{-1} and optimization is completed after 15 more epochs.

Regularization: Although Kinetics-600 and Jester are very large benchmarks and immune to over-fitting, UCF101 still requires intensive regularization. Weight decay of $1x10^{-3}$ is applied for all the parameters of the network. A dropout layer is applied before the final convolution/linear layer of the networks. While the dropout ratio is kept at 0.2 for Kinetics-600 and Jester, it is increased to 0.9 for UCF101.

Augmentation: For temporal augmentation, input clips are selected from a random temporal position in the video clip. If the video contains a smaller number of frames than the input size, loop padding is applied. For the input to the networks, always 16-frame clips are used. For the Jester benchmark, it is critical to capture the full content of the gesture video in the selected input clip. Therefore, we have applied downsampling of 2 by selecting 16 frames from 32 frames for the Jester benchmark as proposed by [30].

For spatial augmentation, we have selected a random spatial position from the input video. Moreover, we have selected a scale randomly from $\{1, \frac{1}{2^{1/4}}, \frac{1}{2^{3/4}}, \frac{1}{2}\}$ in order to perform multi-scale cropping as in [131]. For Kinetics-600 and UCF101, input clips are flipped with 50% probability. After the augmentations, the input clip to the network has the size of 3 x 16 x 112 x 112 referring to the number of input channels, frames, width, and height pixels, respectively.

Recognition: For Kinetics-600 and UCF101, we select non-overlapping 16-frame clips from each video sample. Then center cropping with scale 1 is applied to each clip. Using the pretrained models, class scores for each clip are calculated. For each video, we average the scores of all clips. The class with the highest score indicates the class label of the video.

Implementation: Network architectures are implemented in PyTorch and trained with a single Nvidia Titan Xp GPU. We make our code publicly available at `https://github.com/okankop/Efficient-3DCNNs` for the reproducibility of the results.

4.3.4 Experiments

In this section, we first explain the experimented datasets. Then, we discuss the achieved results for the experimented network architectures together with their runtime performance on both Nvidia Titan Xp and Jetson TX2 embedded system.

4.3.4.1 Datasets

Kinetics-600 [124], Jester [4] and UCF101 [6] datasets are selected for the evaluation of the created resource efficient 3D CNN architectures. The details of these datasets can be found in Section 2.5.

We selected the Kinetics-600 benchmark in order to evaluate the capacity of the experimented networks. A real-life application rarely tries to classify 600 different classes. However, these kinds of very large-scale datasets are very useful to evaluate the

capacity of the networks to learn. Although it is still necessary to capture the motion patterns in the video, the network should especially capture the spatial content in order to identify the correct class label of the video. For example, there are 9 different "eating something" classes where "something" is one of "burger, cake, carrot, chips, doughnut, hotdog, ice cream, spaghetti, watermelon". Although "eating" action is same for all these, the true label can only be identified when the network captures discriminative features of what is being eaten.

Unlike the Kinetics-600 benchmark, in the Jester dataset, the spatial content of all video samples is the same: A person sitting in front of a camera performs a hand gesture from almost the same distance. Moreover, the selection of classes is more focused on the movement of the hand. That is why the Jester benchmark is suitable to inspect the ability of the networks in capturing motion patterns.

Lastly, the UCF101 benchmark is selected in order to inspect the applicability of transfer learning for the experimented network architectures.

4.3.4.2 Results

In this section, we elaborate on our findings in the experiments that we have conducted for 5 different network architectures, 4 levels of complexity (except for SqueezeNet) on 3 different benchmarks. Moreover, the runtime performance of the models is evaluated on 2 different platforms, namely Nvidia Titan XP GPU and Nvidia Jetson TX2 embedded system. According to the results in Table 4.17, the following conclusions can be inferred:

Accuracy:

(i) The deeper architectures (3D-ShuffleNet, 3D-ShuffleNetV2, 3D-MobileNetV2) achieve better results compared to shallower architectures (3D-SqueezeNet, 3D-MobileNetV1). Accordingly, resource efficient 3D CNNs should not be designed too shallow in order to save complexity.

(ii) Motion patterns are better captured with depthwise convolutions. Since depthwise convolutions have kernels of 3x3x3, they can capture relations in the depth dimension together with the spatial dimensions. The main building block of 3D-MobileNetV2 is the inverted residual block, which expands the number of channels to the input of depthwise convolution layers with an expansion ratio. Therefore, it contains more depthwise convolution filters compared to other architectures. Consequently, it achieves by far the best performance in the Jester benchmark, although it has inferior results in Kinetics-600 and UCF101 benchmarks.

(iii) All models showed comparatively similar performance on both Kinetics-600 and UCF101 datasets. This shows transfer learning is a valid approach for resource efficient 3D CNNs since there is a direct correlation between model performances on these two datasets.

Complexity level:

(iv) There is a severe performance degradation if the networks are scaled with very small *width_multiplier* in order to satisfy the required computational complexity. For example, in the first block of the Table 4.17, we can see that 3D-MobileNetV2 0.2x

Model	MFLOPs	Params	Speed (cps)		Accuracy (%)		
			Titan XP	Jetson TX2	Kinetics-600	Jester	UCF101
3D-ShuffleNetV1 0.5x	78	0.55M	398	69	35.51	89.23	64.39
3D-ShuffleNetV2 0.25x	116	0.83M	442	82	25.73	86.91	56.52
3D-MobileNetV1 0.5x	98	1.17M	290	57	31.74	87.61	62.17
3D-MobileNetV2 0.2x	63	0.96M	357	42	24.14	86.43	55.56
3D-ShuffleNetV1 1.0x	199	1.52M	269	49	45.31	92.27	76.00
3D-ShuffleNetV2 1.0x	195	1.91M	243	44	46.10	91.96	77.90
3D-MobileNetV1 1.0x	241	3.91M	164	31	40.07	90.81	70.95
3D-MobileNetV2 0.45x	177	1.40M	203	19	36.47	90.21	68.31
3D-ShuffleNetV1 1.5x	347	2.92M	204	31	52.75	93.12	81.73
3D-ShuffleNetV2 1.5x	291	3.16M	186	34	52.05	93.16	82.32
3D-MobileNetV1 1.5x	429	8.22M	116	19	48.24	91.28	76.00
3D-MobileNetV2 0.7x	325	2.05M	130	13	45.59	93.34	77.32
3D-ShuffleNetV1 2.0x	538	4.76M	161	24	56.84	93.54	84.96
3D-ShuffleNetV2 2.0x	438	6.64M	146	26	55.17	93.71	83.32
3D-MobileNetV1 2.0x	662	14.10M	88	15	48.53	92.56	76.18
3D-MobileNetV2 1.0x	561	3.12M	93	9	50.65	94.59	81.60
3D-SqueezeNet	926	2.15M	682	46	40.52	90.77	74.94
ResNet-18	8323	33.36M	334	17	57.65	93.34	80.09
ResNet-50	9835	44.54M	183	11	63.00	93.70	88.92
ResNet-101	13664	83.58M	142	8	64.18	94.10	87.02
ResNeXt-101	9652	48.75M	122	7	68.30	94.89	89.08
I3D [124]	111331	12.70M	—	—	71.90	—	—

Table 4.17: Comparison of resource efficient 3D architectures over video classification accuracy, number of parameters and speed on two different platforms and four levels of computation complexity. K600 stands for Kinetics-600 dataset. The calculations of MFLOPs, parameters and speeds are done for Kinetics-600 benchmark. For speed calculations (clips per second (cps)), the used platforms are Titan Xp and Jetson TX2; and the batch size is set to 8. All models takes 16 frames input with 112 x 112 spatial resolution except for I3D, which takes 64 frames input with 224 x 224 spatial resolution.

and 3D-ShuffleNetV2 0.25x achieve 5-9% worse than 3D-ShuffleNetV1 0.5x and 3D-MobileNetV1 0.5x in Kinetics-600 benchmark. The capacity of the models degrades severely as the *width_multiplier* gets smaller, especially when it is less than 0.5. We can see the same pattern on all three benchmarks that we have experimented with.

(v) The main design criteria of the 3D-SqueezeNet is to save the number of parameters, not computations. Therefore it has the smallest number of parameters at the highest complexity level. However, it also has around 300 million more FLOPs compared to other architectures since it does not make use of depthwise convolutions.

Runtime performance:

(vi) Although the network architectures contain similar FLOPs, some architectures are much faster than others. As highlighted by [176], this is due to several other factors affecting speed such as memory access cost (MAC) and degree of parallelism, which are not taken into account by FLOPs.

(vii) 3D-SqueezeNet is the only architecture that does not make use of depthwise convolutions, hence contains the highest number of FLOPs. However, surprisingly it has the highest runtime performance. This is due to the latest CUDNN [237] library which is specifically optimized for standard convolutions. Similar results can also be observed with ResNet and ResNeXt architectures.

(viii) Runtime performance heavily depends on the hardware that the network architecture is running. For example, for the highest two complexity levels, 3D-ShuffleNetV1 is faster than 3D-ShuffleNetV2 on GPU, whereas 3D-ShuffleNetV2 achieves higher runtime than 3D-ShuffleNetV1 on Jetson TX2.

State-of-the-art comparison:

(ix) Architectures with more parameters and FLOPs like ResNets, ResNeXt-101 and I3D achieve generally better results for datasets measuring the capacity of the tested architectures like Kinetics dataset as evaluated and shown in Table 4.17. However, network design makes a huge difference. For example, 3D-ShuffleNetV1 2.0x achieves similar performance with ResNet-18, although ResNet-18 requires 7 times more parameters and around 15 times more FLOPs.

(x) The architecture design should be done according to the given task. As inverted residual block excels at capturing dynamic motions, 3D-MobileNetV2 1.0x achieves better results than much wider and deeper ResNet-101 (around 20 times more parameters and FLOPs) at the Jester benchmark.

4.3.5 Summary

In recent years, the research in action recognition has mostly focused on obtaining the best accuracy by generating deep and wide CNN architectures. However, real-world applications require resource efficient architectures that take runtime, memory and power budget into account. Recently, several resource efficient 2D CNN architectures have been proposed. However, there is a lack of architectures for 3D counterparts. This work aims to fill this research gap.

The proposed architectures are generated by implementing the 3D versions of SqueezeNet, MobileNet, MobileNetV2, ShuffleNet, ShuffleNetV2 architectures for 4 different complexity levels. The performance of these architectures has been evaluated using 3 different benchmarks, which are selected according to analyze models' capacities, how well the models capture the motion and the applicability of transfer learning for each model.

According to the analysis for 4 different complexity levels, the results show that these resource efficient 3D CNN architectures provide considerable classification performances. Using the *width_multiplier*, the capacity of the architectures can be modified flexibly. The results on the Jester benchmark show that depthwise convolutions are very good at

capturing motion patterns. Moreover, nearly all models run in real-time both at Titan XP and Jetson TX2. As the results proved the applicability of transfer learning, these architectures can be used for other real-world applications by using pretrained models.

4.4 Two-Model Hierarchical Architecture to Reduce Computational Complexity

This section addresses the challenges of online recognition of hand gestures and proposes a two-level hierarchical architecture to reduce system resource usage when the system remains idle.

Real-time recognition of dynamic hand gestures from video streams is a challenging task since (i) there is no indication when a gesture starts and ends in the video, (ii) performed gestures should only be recognized once, and (iii) the entire architecture should be designed considering the memory and power budget. In this work, we address these challenges by proposing a hierarchical structure enabling offline-working convolutional neural network (CNN) architectures to operate online efficiently by using sliding window approach. The proposed architecture consists of two models: (1) A detector which is a lightweight CNN architecture to detect gestures and (2) a classifier which is a deep CNN to classify the detected gestures. In order to evaluate the single-time activations of the detected gestures, we propose to use Levenshtein distance as an evaluation metric since it can measure misclassifications, multiple detections, and missing detections at the same time. We evaluate our architecture on two publicly available datasets - EgoGesture and NVIDIA Dynamic Hand Gesture Datasets - which require temporal detection and classification of the performed hand gestures. ResNeXt-101 model, which is used as a classifier, achieves the state-of-the-art offline classification accuracy of 94.04% and 83.82% for depth modality on EgoGesture and NVIDIA benchmarks, respectively. In real-time detection and classification, we obtain considerable early detections while achieving performances close to the offline operation. This section is based on our publication *Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks* [22].

4.4.1 Motivation

Computers and computing devices are becoming an essential part of our lives day by day. The increasing demand for such computing devices increased the necessity of easy and practical computer interfaces. For this reason, systems using vision-based interaction and control are becoming more common, and as a result of this, gesture recognition is getting more and more popular in the research community due to various application possibilities in human-machine interaction. Compared to mouse and keyboard, any vision-based interface is more convenient, practical and natural because of the intuitiveness of gestures.

Gesture recognition can be practiced with mainly three methods: Using (i) glove-based wearable devices [238], (ii) 3-dimensional locations of hand keypoints [239] and (iii) raw visual data. The first method comes with the obligation of wearing an additional device with which lots of cables come even though it provides good results in terms of both accuracy and speed. The second, on the other hand, requires an extra step of hand-keypoints extraction, which brings additional time and computational cost. Lastly, for (iii), only an image capturing sensor is required such as camera, infrared sensor or depth

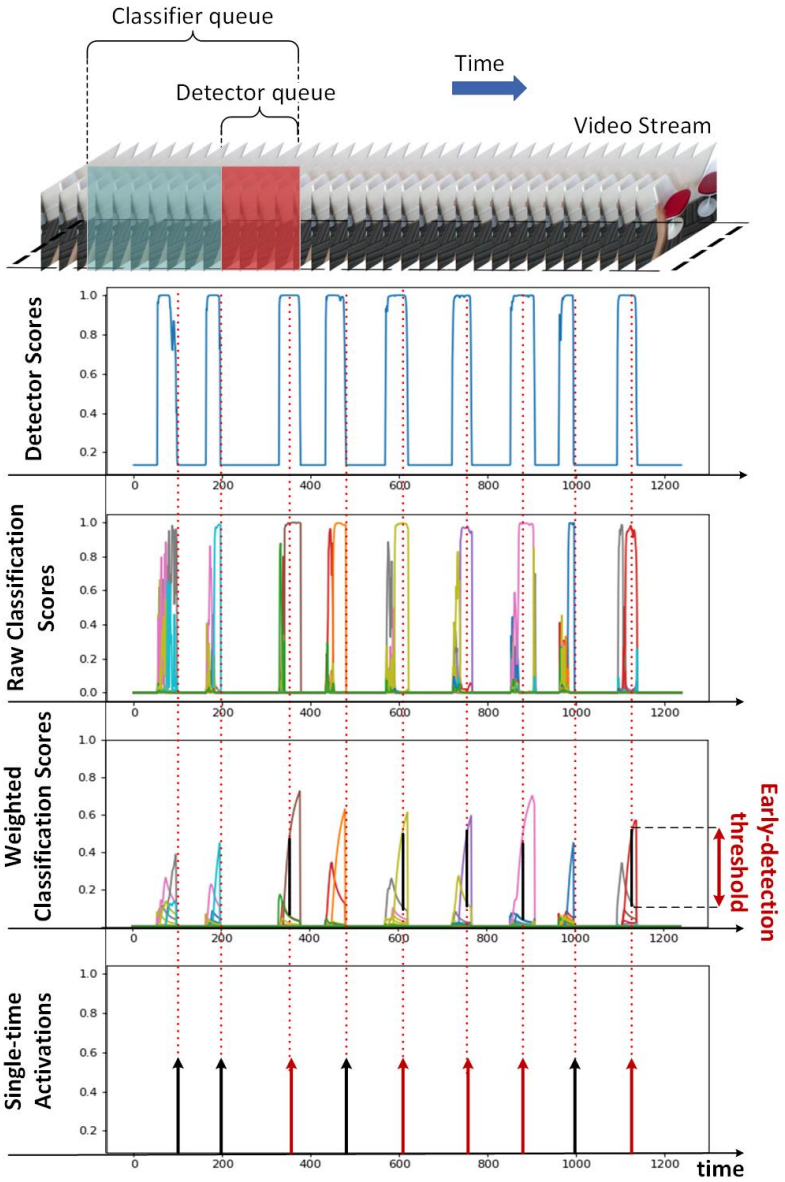


Figure 4.10: Illustration of the proposed pipeline for real-time gesture recognition. The video stream is processed using a sliding window approach with stride of one. The top graph shows the detector probability scores which is activated when a gesture starts and kept active till it ends. The second graph shows the classification score for each class with a different color. The third graph applies weighted-average filtering on raw classification scores which eliminates the ambiguity between possible gesture candidates. The bottom graph illustrates the single-time activations such that red arrows represent early detections and black ones represent detections after gestures finalize.

sensor, which are independent of the user. Since the user does not require to wear a burdensome device to achieve an acceptable accuracy in recognition and sufficient speed in computation, this option stands out as the most practical one. The infrastructure of any gesture recognition system needs to be practical. After all, we aim to use it in real-life scenarios.

In this work, in order to provide a practical solution, we have developed a vision-based gesture recognition approach using deep convolutional neural networks (CNNs) on raw video data. Currently, CNNs provide the state-of-the-art results for not only image-based tasks such as object detection, image segmentation and classification, but also for video-based tasks such as activity recognition and action localization as well as gesture recognition [19, 133, 194].

In real-time gesture recognition applications, there are several characteristics that the system needs to satisfy: *(i)* An acceptable classification accuracy, *(ii)* fast reaction time, *(iii)* resource efficiency and *(iv)* single-time activation per each performed gesture. All these items contain utmost importance for a successful real-time vision-based gesture recognition application. However, most of the previous research only considers *(i)* and tries to increase the offline classification accuracy in gesture recognition disregarding the remaining items. Some proposed approaches are even impossible to run in real-time since they consist of several deep CNNs on multiple input modalities, which is forcing the limits of memory and power budget [141].

In this section, we propose a hierarchical architecture for the task of real-time hand gesture detection and classification that allows us to integrate offline working models and still satisfy all the above-mentioned attributes. Our system consists of an offline-trained deep 3D CNN for gesture classification (classifier) and a lightweight, shallow 3D CNN for gesture detection (detector). Fig. 4.10 illustrates the pipeline of the proposed approach. A sliding window is used over the incoming video stream feeding the input frames to the detector via detector queue. The top graph in Fig. 4.10 shows the detector probability scores which become active when the gestures are being performed, and remain inactive for the rest of the time. The classifier becomes active only when the detector detects a gesture. This is very critical since most of the time, no gesture is performed in real-time gesture recognition applications. Therefore, there is no need to keep the high-performance classifier always active, which increases the memory and power consumption of the system considerably. The second graph shows the raw classification scores of each class with a different color. As it can be seen from the graph, scores of similar classes become simultaneously high especially at the beginning of the gestures. In order to resolve these ambiguities, we have weighted the class scores to avoid making a decision at the beginning of the gestures (third graph in Fig. 4.10). Lastly, the bottom graph illustrates the single-time activations, where red arrows represent the early detections and black ones represent the detections after gestures end. Our system can detect gestures earlier in their nucleus part, which is the part distinguishing the gesture from the rest. We propose to use the Levenshtein distance as an evaluation metric to compare the captured single-time activations with ground-truth labels. This metric is more suitable and evaluative since it can measure misclassifications, multiple detections and missing detections at the same time.

We evaluated our approach on two publicly available datasets, which are EgoGesture Dataset [171] and NVIDIA Dynamic Hand Gesture Dataset [133] (nvGesture). For the classifier of the proposed approach, any offline working CNN architecture can be used. For our experiments, we have used well-known C3D [85] and 3D-ResNeXt-101 [131]. We have achieved the state-of-the-art offline classification accuracies of 94.03% and 83.82% on depth modality with ResNeXt-101 architecture on EgoGesture and nvGesture datasets, respectively. For real-time detection and classification, we achieve considerable early detections by relinquishing a little amount of recognition performance.

4.4.2 Methodology

We start by elaborating on our two-model hierarchical architecture that enables the state-of-the-art CNN models to be used in real-time gesture recognition applications as efficiently as possible. After introducing the architecture, training details are described. Finally, we give a detailed explanation for the used post-processing strategies that allow us to have single-time activation per gesture in real-time.

4.4.2.1 Architecture

Recently, with the availability of large datasets, CNN based models have proven their ability in action/gesture recognition tasks. 3D CNN architectures especially stand out for video analysis since they make use of the temporal relations between frames together with their spatial content. However, there is no clear description of how to use these models in a real-time dynamic system. With our work, we aim to fill this research gap.

Fig. 4.11 illustrates the used workflow for an efficient real-time recognition system using a sliding window approach. In contrary to offline testing, we do not know when a gesture starts or ends. Because of this, our workflow starts with a detector which is used as a switch to activate the classifier if a gesture gets detected. Our detector and classifier models are fed by a sequence of frames with size n and m, respectively, such as $n \ll m$ with an overlapping factor as shown in Fig. 4.11. The stride value used for the sliding window is represented by s in Fig. 4.11, and it is the same for both the detector and the classifier. Although higher stride provides less resource usage, we have chosen s as 1 since it is small enough not to miss any gestures and allows us to achieve better performance. In addition to the detector and classifier models, one post-processing and one single-time activation service are introduced to the workflow. In the following parts, we are going to explain these blocks in detail.

Detector

The purpose of the detector is to distinguish between *gesture* and *no gesture* classes by running on a sequence of images, which detector queue masks. Its main and only role is to act as a switch for the classifier model, meaning that if it detects a *gesture*, then the classifier is activated and fed by the frames in the classifier queue.

Since the overall accuracy of this system highly depends on the performance of the detector, we require the detector to be (i) robust, (ii) accurate in the detection of true

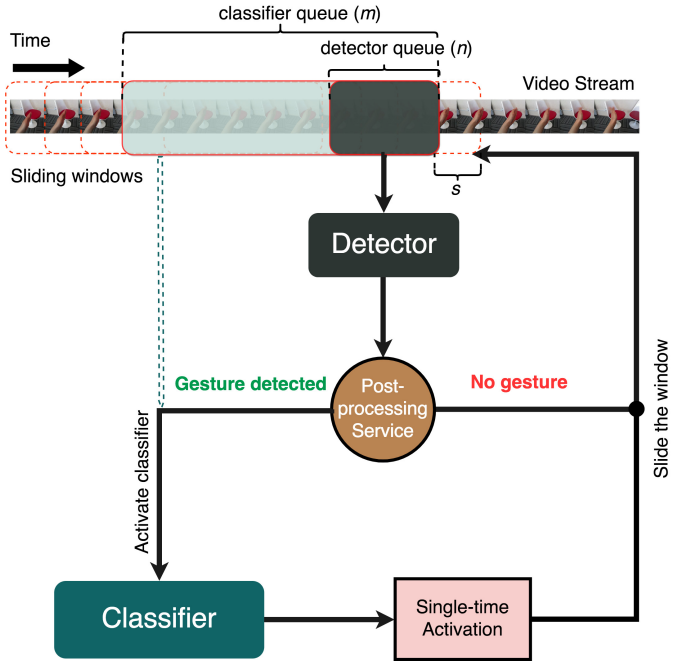


Figure 4.11: The general workflow of the proposed two-model hierarchical architecture. Sliding windows with stride s run through incoming video frames where detector queue placed at the very beginning of classifier queue. If the detector recognizes an action/gesture, then the classifier is activated. The detector's output is post-processed for a more robust performance, and the final decision is made using a single-time activation block where only one activation occurs per performed gesture.

positives (gestures), and (iii) lightweight as it runs continuously. For the sake of (i), the detector runs on a smaller number of frames than the classifier to which we refer as detector and classifier queues. For (ii), the detector queue is placed at the very beginning of the classifier queue as shown in Fig. 4.11, and this enables the detector to activate the classifier whenever a gesture starts regardless of the gesture duration. Moreover, the detector model is trained with a weighted-cross-entropy loss in order to decrease the likelihood of false positives (i.e., achieve a higher recall rate). The class weights for *no gesture* and *gesture* classes are selected as 1 and 3, respectively as our experiments showed that this proportion is sufficient to have 98+% and 97+% recall rates in EgoGesture and nvGesture datasets, respectively. Besides that, we post-process the output probabilities and set a counter for the consecutive number of *no gesture* predictions in the decision to deactivate the classifier. For (iii), ResNet-10 architecture is constructed using the ResNet block in Fig. 4.12 with very small feature sizes in each layer as given in Table 4.18, which results in less than 1M ($\approx$ 862K) parameters. F and N correspond to the number of feature channels and the number blocks in corresponding layers, respectively. *BN*, *ReLU* and *group* in Fig. 4.12 refers to batch normalization, rectified linear unit nonlinearities and the number of group convolutions, respectively.

Layer	Output Size	ResNeXt-101	ResNet-10
conv1	L x 56 x 56	conv(3x7x7), stride (1, 2, 2)	
pool	L/2 x 28 x 28	MaxPool(3x3x3), stride (2, 2, 2)	
conv2_x	L/2 x 28 x 28	N:3, F:128	N:1, F:16
conv3_x	L/4 x 14 x 14	N:24, F:256	N:1, F:32
conv4_x	L/8 x 7 x 7	N:36, F:512	N:1, F:64
conv5_x	L/16 x 4 x 4	N:3, F:1024	N:1, F:128
	NumCls	global average pooling, fc layer with softmax	

Table 4.18: Detector (ResNet-10) and Classifier (ResNeXt-101) architectures. For ResNet-10, max pooling is not applied when input of 8-frames is used.

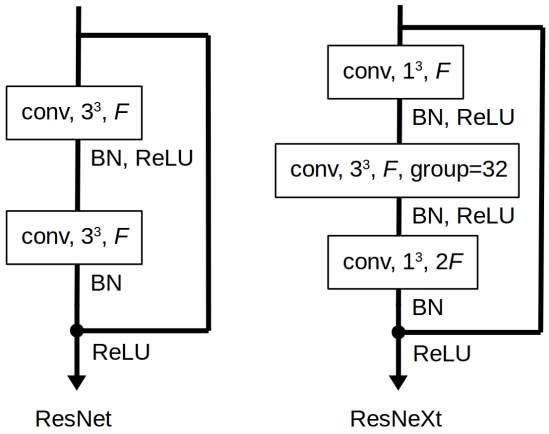


Figure 4.12: ResNet and ResNeXt blocks used in the detector and classifier architectures.

Classifier

Since we do not have any limitation regarding the size or complexity of the model, any architecture providing a good classification performance can be selected as the classifier. This leads us to use two recent 3D CNN architectures (C3D [85] and ResNext-101 [88]) as our classifier model. However, it is important to note that our architecture is independent of the model type.

For the C3D model, we have used the exact same model as in [85], but only changed the number of nodes in the last two fully connected layers from 4096 to 2048. For ResNeXt-101, we have followed the guidelines of [131] and chosen the model parameters as given in Table 4.18 with ResNeXt block as given in Fig. 4.12.

Since the number of parameters for 3D CNNs is much more than 2D CNNs, they require more training data in order to prevent overfitting. Because of this reason, we pretrain our classifier architectures first on Jester dataset [4], which is the largest

publicly available hand gesture dataset, and then fine-tune our model on EgoGesture and nvGesture datasets. This approach increased the accuracy and shortened the training duration drastically.

4.4.2.2 Training Details

We use stochastic gradient descent (SGD) with Nesterov momentum = 0.9, damping factor = 0.9, and weight decay = 0.001 as optimizer. After pretraining on the Jester dataset, the learning rate is started with 0.01, and divided by 10 at 10^{th} and 25^{th} epochs, and training is completed after 5 more epochs.

For regularization, we used a weight decay ($\gamma = 1 \times 10^{-3}$), which is applied to all the parameters of the network. We also used dropout layers in C3D and several data augmentation techniques throughout training.

For data augmentation, three methods were used: (1) Each image is randomly cropped with size 112×112 and scaled randomly with one of $\{1, \frac{1}{2^{1/4}}, \frac{1}{2^{3/4}}, \frac{1}{2}\}$ scales. (2) Spatial elastic displacement [187] with $\alpha = 1$ and $\sigma = 2$ is applied on the cropped and scaled images. For temporal augmentation, (3) we randomly select consecutive frames according to the size of input sample duration from the entire gesture videos. If the sample duration is more than the number of frames in the target gesture, we append frames starting from the very first frame in a cyclic fashion. We also normalized the images into 0-1 scale using the mean and standard deviation of the whole training sets in order to force models to learn faster. The same training details are used for the detector and classifier models.

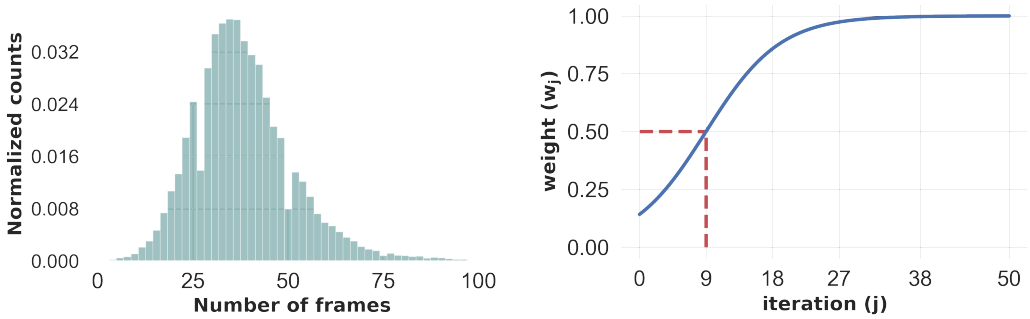
During offline and online testing, we scale images and apply center cropping to get 112×112 images. Then only normalization is performed for the sake of consistency between training and testing.

Implementation: Our architecture is implemented in PyTorch and trained with a single Titan Xp GPU. We make our code publicly available at https://github.com/ahmetgunduz/Real-time-GesRec for the reproducibility of the results.

4.4.2.3 Post-processing

In dynamic hand gestures, it is possible that the hand gets out of the camera view while performing gestures. Even though the previous predictions of the detector are correct, any misclassification reduces the overall performance of the proposed architecture. In order to make use of previous predictions, we add the raw softmax probabilities of the previous detector predictions into a queue (q_k) with size k, and apply filtering on these raw values and obtain final detector decisions. With this approach, the detector increases its confidence in decision making and clears out most of the misclassifications in consecutive predictions. The size of the queue (k) is selected as 4, which achieved the best results for stride s of 1 in our experiments.

We have applied (i) average, (ii) exponentially-weighted average and (iii) median filtering separately on the values in q_k. While average filtering simply takes the mean value of q_k, median filtering takes the median. Exponentially-weighted average filtering,



(a) (b)

Figure 4.13: (a) Histogram of the gesture durations for the EgoGesture dataset. (b) Sigmoid-like weight function used for single-time activations according to the Eq. (4.13).

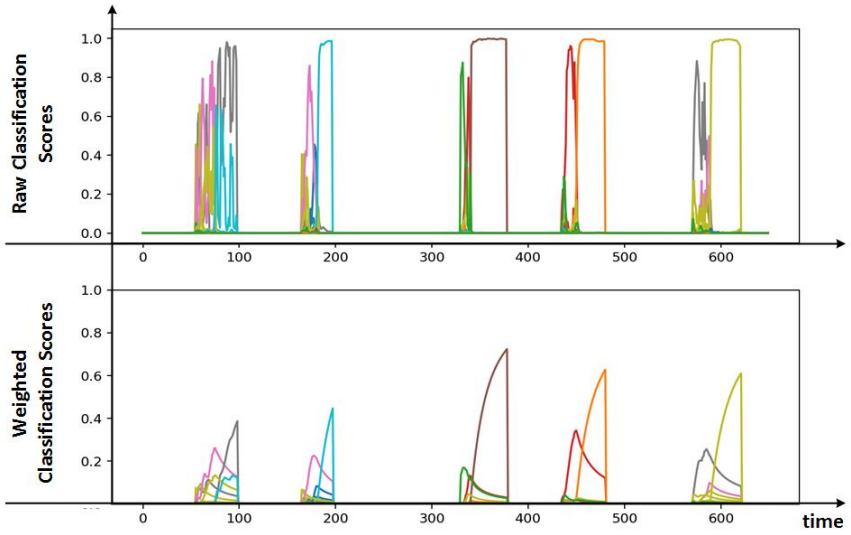


Figure 4.14: Raw (top) and weighted (bottom) classification scores. At the top graph, we observe a lot of noise at the beginning of all gestures; however, close to the end of each gesture, the classifier gets more confident. The bottom graph shows that we can remove this noise part by assigning smaller weights to the beginning part of the gestures.

on the other hand, takes the weighted average of the samples using the weight function of $w_i = \exp^{-(1-(k-i))/k}$ where i stands for the index of the i^{th} previous sample and satisfies $0 \leq i < k$, and w_i is the weight for the i^{th} previous sample. Out of these three filtering strategies, we have used median filtering since it achieves slightly better results.

4.4.2.4 Single-time Activation

In real-time gesture recognition systems, it is extremely important to have smaller reaction time and single-time activation for each gesture. Pavlovic et al. states that dynamic gestures have *preparation*, *nucleus* (*peak* or *stroke* [240]) and *retraction* parts [16]. Out of all parts, nucleus is the most discriminative one, since we can decide which gesture is performed in nucleus part even before it ends.

Single-time activation is achieved through a two-level control mechanism. Either a gesture is detected when a confidence measure reaches a threshold level before the gesture actually ends (early-detection), or the gesture is predicted when the detector deactivates the classifier (late-detection). In late-detection, we assume that the detector should not miss any gesture since we assure that the detector has a very high recall rate.

The most critical part of the early-detection is that the gestures should be detected after their nucleus parts for a better recognition performance. Because several gestures can contain a similar preparation part which creates an ambiguity at the beginning of the gestures, as can be seen on the top graph of Fig. 4.14. Therefore, we have applied weighted-averaging on class scores with a weight function as in Fig. 4.13 (b), and its formula is given as:

$$w_j = \frac{1}{(1 + \exp^{-0.2\times(j-t)})}, \tag{4.13}$$

where j is the iteration index of an active state, at which a gesture is detected, and t is calculated by using the following formula:

$$t = \left\lfloor \frac{\mu}{4 \times s} \right\rfloor, \tag{4.14}$$

where μ corresponds to the mean duration of the gestures (in the number of frames) in the dataset and s is the stride length. Fig. 4.13 (a) shows the distribution of the number of frames per gesture for the EgoGesture dataset, where mean duration μ is equal to 38,4. Accordingly, for stride of $s = 1$, t is calculated as 9, which is similar for also nvGesture dataset. When a gesture starts, we start to multiply raw class scores with weights w_j and apply averaging. These parameters allow us to have weights equal to or higher than 0.5 in the nucleus part of the gestures on average. Fig. 4.14 shows the probability scores of five gestures over each iteration and their corresponding weighted averages. It can easily be observed that the ambiguity of the classifier at the preparation part of the gestures is successfully resolved with this approach.

With this weighted-averaging strategy, we force our single-time activator to make a decision at the mid-late part of the gestures after capturing their nucleus parts. On the other hand, we need a confidence measure for early-detections in real-time since the duration of gestures varies. Hence, we decided to use the difference between weighted average scores of each class as our confidence measure for early-detection. When the detector switches the classifier on, weighted average probabilities for all classes are calculated at each iteration. If the difference between the two highest average probabilities is more than a threshold τ_{early}, then early-detection is triggered;

Algorithm 1 Single-time activation in real-time gesture recognition

Input: Incoming frames from video data.
Output: Single-time activations.

for each "frame-window" w_i of length m **do**
  if a gesture is detected **then**
    state $\leftarrow$ "Active"
    $\alpha \leftarrow probs_{j-1} \times (j-1)$
    $meanprobs = (\alpha + w_j \times probs_j)/j$
    $(max_1, max_2) = \max_{gesture} [meanprobs]_2$
    if $(max1 - max2) \geq \tau_{\textbf{early}}$ **then**
      early-detection $\leftarrow$ "True"
      return gesture with max_1
    $j \leftarrow j + 1$
  if the gesture ends **then**
    state $\leftarrow$ "Passive"
    if *early-detection* $\neq$ "True" & $max_1 \geq \tau_{\textbf{late}}$ **then**
      return gesture with max_1
  $i \leftarrow i + 1$

otherwise, we wait for the detector to switch off the classifier and the class with the highest score above τ_{late} (fixed to 0.15 as it showed the best results in our experiments) is predicted as late-detection. Details for this strategy can be found in Algorithm 1.

4.4.2.5 Evaluation of the Activations

As opposed to offline testing which usually considers only class accuracies, we must also consider the following scenarios for our real-time evaluation:

- Misclassification of the gesture due to the classifier,
- Not detecting the gesture due to the detector,
- Multiple detections in a single gesture.

Considering these scenarios, we propose to use the Levenshtein distance as our evaluation metric for online experiments. The Levenshtein distance is a metric that measures the distance between sequences by counting the number of item-level changes (insertion, deletion, or substitutions) to transform one sequence into the other. For our case, one video and the gestures in this video correspond to a sequence and the items in this sequence, respectively. For example, let's consider the following ground truth and predicted gestures of a video:

$$\begin{aligned} &GroundTruth && [1,2,3,4,5,6,7,8,9] \\ &Predicted && [1,2,7,4,5,6,6,7,8,9] \end{aligned}$$

Model	Input	Modality	
		RGB	Depth
VGG-16 [171]	16-frames	62.50	62.30
VGG-16 + LSTM [171]	16-frames	74.70	77.70
C3D	16-frames	86.88	88.45
ResNeXt-101	16-frames	90.94	91.80
C3D+LSTM+RSTTM [171]	16-frames	89.30	90.60
ResNeXt-101	32-frames	93.75	**94.03***

Table 4.19: Comparison with state-of-the-art on the test set of EgoGesture dataset.

Model	Input	Modality	
		RGB	Depth
C3D	16-frames	86.88	88.45
C3D	24-frames	89.20	89.07
C3D	32-frames	90.57	**91.44**
ResNeXt-101	16-frames	90.94	91.80
ResNeXt-101	24-frames	92.89	93.47
ResNeXt-101	32-frames	93.75	**94.03***

Table 4.20: Classifier's classification accuracy scores on the test set of EgoGesture dataset.

For this example, the Levenshtein distance is 2: The deletion of one of "6" which is detected two times, and the substitution of "7" with "3". We average this distance over the number of true target classes. For this case, the average distance is $2/9 = 0.2222$ and we subtract this value from 1 since we want to measure closeness (in this work it is referred as the Levenshtein accuracy) of our results, which is equal to $(1-0.2222)\times 100 = 77.78\%$.

4.4.3 Experiments

The performance of the proposed approach is tested on two publicly available datasets: EgoGesture and NVIDIA Dynamic Hand Gestures dataset.

Model	Input	Modality	
		RGB	**Depth**
ResNet-10	8-frames	96.58	99.39*
ResNet-10	16-frames	97.00	99.64
ResNet-10	24-frames	97.13	99.15
ResNet-10	32-frames	96.65	**99.68**

Table 4.21: Detector's binary classification accuracy scores on the test set of EgoGesture dataset.

Modality	Recall	Precision	f1-score
RGB	96.64	97.10	96.87
Depth	99.37	99.43	**99.40**

Table 4.22: Detection results of 8-frames ResNet-10 architecture on the test set of EgoGesture dataset.

4.4.3.1 Offline Results Using EgoGesture Dataset

We have provided the details of EgoGesture dataset [171] in Section 2.5. All models are first pretrained on Jester dataset [4]. For test set evaluations, we have used both training and validation set for training.

We initially investigated the performance of C3D and ResNeXt architectures on the offline classification task. Table 4.19 shows the comparison of used architectures with the state-of-the-art approaches. ResNeXt-101 architecture with 32-frames input achieves the best performance.

Secondly, we investigated the effect of the number of input frames on gesture detection and classification performance. Results in Table 4.20 and Table 4.21 show that we achieve a better performance as we increase the input size for all the modalities. This depends highly on the characteristics of the used datasets, especially on the average duration of the gestures.

Thirdly, the RGB and depth modalities are investigated for different input sizes. We always observed that the models with depth modality show better performance than the models with RGB. Depth sensor filters out the background motion and allows the models to focus more on the hand motion, hence more discriminative features can be obtained from depth modality. For real-time evaluation, ResNet-10 with depth modality and input size of 8-frames is chosen as the detector, since a smaller window size allows the detector to discover the start and end of the gestures more robustly. The detailed results of this model are shown in Table 4.22.

Model	Modality	
	RGB	**Depth**
C3D	73.86	77.18
R3DCNN [133]	74.10	80.30
ResNeXt-101	78.63	**83.82**

Table 4.23: Comparison with state-of-the-art on the test set of nvGesture dataset.

Model	Input	Modality	
		RGB	**Depth**
C3D	16-frames	62.67	70.33
C3D	24-frames	65.35	70.33
C3D	32-frames	73.86	**77.18**
ResNeXt-101	16-frames	66.40	72.82
ResNeXt-101	24-frames	72.40	79.25
ResNeXt-101	32-frames	78.63	**83.82***

Table 4.24: Classifier's classification accuracy scores on the test set of nvGesture dataset.

4.4.3.2 Offline Results Using nvGesture Dataset

The details of nvGesture dataset [133] is provided in Section 2.5. We again initially investigated the performance of C3D and ResNeXt architectures on the offline classification task, by comparing them with the state-of-the-art models. As shown in Table 4.23, ResNeXt-101 architecture achieves the best performance. Similar to the EgoGesture dataset, we achieve a better classification and detection performance as we increase the input size, for all the modalities, as shown in Table 4.24 and Table 4.25. Depth modality again achieves better performance than RGB modality for all input sizes. Moreover, ResNet-10 with depth modality and input size of 8-frames is chosen as the detector in the online testing, whose detailed results are given in Table 4.26.

For real-time evaluation, we have selected 8-frames ResNet-10 detectors with depth modality and best performing classifiers in both datasets, which have * sign in corresponding tables.

4.4.3.3 Real-Time Classification Results

EgoGesture and nvGesture datasets have 431 and 482 videos, respectively in their test sets. We evaluated our proposed architecture on each video separately and calculated an average Levenshtein accuracy at the end. We achieve 91.04% and 77.39% Levenshtein accuracies in EgoGesture and nvGesture datasets, respectively.

Model	Input	Modality	
		RGB	**Depth**
ResNet-10	8-frames	70.22	97.30*
ResNet-10	16-frames	85.90	97.82
ResNet-10	24-frames	89.00	**98.02**
ResNet-10	32-frames	93.88	97.30

Table 4.25: Detector's binary classification accuracy scores on the test set of nvGesture dataset.

Modality	Recall	Precision	f1-score
RGB	70.22	80.31	74.93
Depth	97.30	97.41	**97.35**

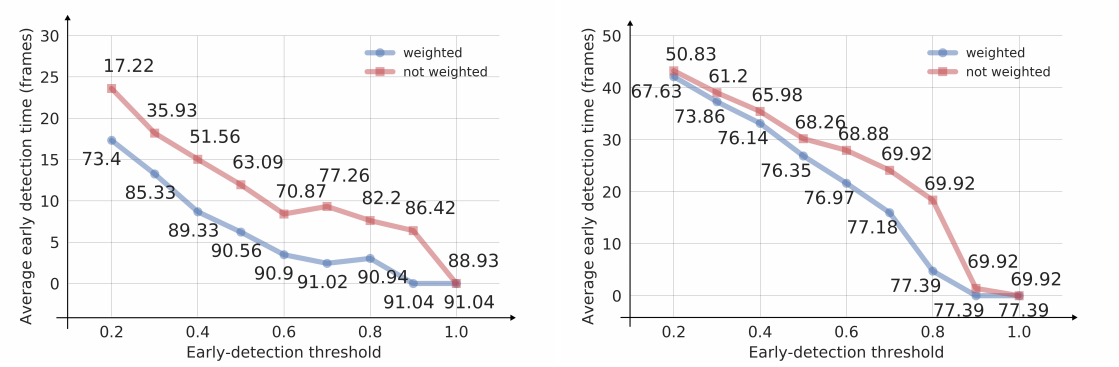
Table 4.26: Detection results of 8-frames ResNet-10 architecture on the test set of nvGesture dataset.

Moreover, the early detection times are investigated by simulating different early-detection threshold levels (τ_{early}) varying from 0.2 to 1.0 with 0.1 steps. Fig. 4.15 compares early detection times of weighted averaging and uniform averaging approaches for both EgoGesture and nvGesture datasets. For both datasets, weighted averaging performs considerably better than uniform averaging. As we increase the threshold, we force the architecture to make a decision towards the end of gestures, hence achieving better performance. However, we can gain considerable early detection performance by relinquishing a little amount of performance. For example, if we set detection threshold τ_{early} to 0.4 for the EgoGesture dataset, we can make our single-time activations 9 frames earlier on average by relinquishing only 1.71% Levenshtein accuracy. We also observe that mean early detection times are longer for the nvGesture dataset since it contains weakly-segmented videos.

Lastly, we investigated the execution performance of our two-model approach. Our system runs on average at 460 fps when there is no gesture (i.e. only detector is active) and 62 (41) fps in the presence of gesture (i.e. both detector and classifier are active) for ResNeXt-101 (C3D) as the classifier on a single Nvidia Titan Xp GPU with a batch size of 8.

4.4.4 Summary

This section presents a novel two-model hierarchical architecture for real-time hand gesture recognition systems. The proposed architecture provides resource efficiency, early detections, and single-time activations, which are critical for real-time gesture recognition applications.



(a) (b)

Figure 4.15: Comparison of early detection time, early detection threshold and acquired Levenshtein accuracies for (a) EgoGesture and (b) nvGesture datasets. Numerals on each data point represent the Levenshtein accuracies. Early detection times are calculated only for correctly predicted gestures. Blue color refers to the "weighted" approach in single-time activation, and green color refers to "not weighted" approach. For both datasets, as early detection threshold increases, average early detection times reduce, but we achieve better Levenshtein accuracies.

The proposed approach is evaluated on two dynamic hand gesture datasets and achieves similar results for both of them. For real-time evaluation, we have proposed to use a new metric, Levenshtein accuracy, which we believe is a suitable evaluation metric since it can measure misclassifications, multiple detections and missing detections at the same time. Moreover, we have applied weighted-averaging on the class probabilities over time, which improves the overall performance and allows early detection of the gestures at the same time.

4.5 Efficient Online Video Processing by Dissected 3D CNNs

This section addresses the drawback of computational redundancy of operating conventional 3D CNN architectures with sliding window approach by proposing a new architecture.

Convolutional Neural Networks with 3D kernels (3D CNNs) currently achieve state-of-the-art results in video recognition tasks due to their supremacy in extracting spatiotemporal features within video frames. There have been many successful 3D CNN architectures surpassing state-of-the-art results successively. However, nearly all of them are designed to operate offline creating several serious handicaps during online operation. Firstly, conventional 3D CNNs are not dynamic since their output features represent the complete input clip instead of the most recent frame in the clip. Secondly, they are not temporal resolution-preserving due to their inherent temporal downsampling. Lastly, 3D CNNs are constrained to be used with fixed temporal input size limiting their flexibility. In order to address these drawbacks, we propose dissected 3D CNNs, where the intermediate volumes of the network are dissected and propagated over depth (time) dimension for future calculations, substantially reducing the number of computations at online operation. For action classification, the dissected version of ResNet models performs 77-90% fewer computations at online operation while achieving ∼5% better classification accuracy on the Kinetics-600 dataset than conventional 3D-ResNet models. Moreover, the advantages of dissected 3D CNNs are demonstrated by deploying our approach onto several vision tasks, which consistently improved the performance. This section is based on our publication *Dissected 3D CNNs: Temporal Skip Connections for Efficient Online Video Processing* [23].

4.5.1 Motivation

Currently, the primary trend in video recognition tasks is to increase network performance by building deeper and wider 3D CNN architectures [131, 130, 127]. However, these architectures are typically designed to operate offline, ignoring the requirements of online operation. Firstly, most of the 3D CNNs deploy temporal downsampling to reduce the computational cost at the later stages of the network and provide translation invariance (in the time dimension) to the internal representation. This causes the network to become non-dynamic, which is of utmost importance for online operation. Moreover, the resulting network is not temporal resolution-preserving. Secondly, 3D CNNs are typically built to work with a fixed number of input frames. Therefore, online operating frameworks usually use 3D CNNs in a sliding window, either with a small temporal stride [22, 25] or larger stride [133]. In the former case, there is severe resource waste due to reprocessing frames in the overlapping regions, which are already processed in the previous timestamps. In the latter case, there is an information loss since relations between some of the frames are not exploited. These issues make most of the 3D CNNs unsuitable for online operation.

In order to address the limitations mentioned above, we propose a novel 3D CNN architecture, Dissected 3D CNNs (D3D), by incorporating temporal skip connections.

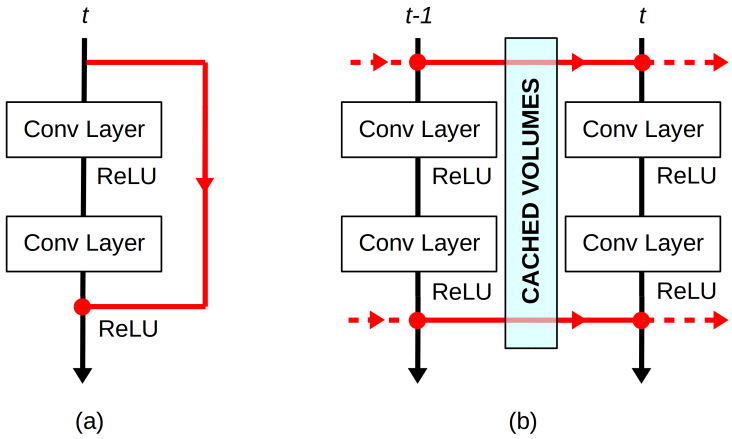


Figure 4.16: Comparison of spatial skip connections (a) first proposed in [88] and temporal skip connections (b) proposed in this work. At every iteration, only the computations for the most recent frame are performed. Afterwards, intermediate volumes from the skip connections are cached to be used for the next iteration. This way, recomputation of previous frames is saved. Skip connections are denoted with red lines.

Skip connections are first proposed in ResNets [88] to overcome the issue of vanishing/exploding gradients and to enhance gradient propagation for deep architectures. Spatial skip connections, which are depicted in Fig. 4.16 (a), can be in the form of summation [88] or concatenation [89, 176]. As opposed to spatial skip connections, we propose temporal skip connections to create a network for efficient online operation. The general idea of the proposed architecture is depicted in Fig. 4.16 (b). Intermediate volumes are always stored in a cache, and only the computations for the new available frame are performed at each iteration. After the computations, the previously cached volumes are replaced with the most recent intermediate feature volumes coming from the skip connections. This way, the volumes in Dissected 3D CNN architecture are propagated without calculating them repeatedly. We incorporate 3D convolutions since we apply concatenation operation in the depth dimension at the skip connections. Although summation is also possible at temporal skip connections, we will show in our ablation study that temporal information is lost with the summation operation, which leads to inferior results. Moreover, spatial skip connections are still applicable on top of temporal skip connections.

The main motivation of this work is to provide a 3D CNN architecture, which satisfies the requirements of online operation. Out of many, the two most important requirements are (i) dynamic operation and (ii) reduced computational complexity. We refer to the term *dynamic* as deployed architecture's ability to adapt its output according to the new coming video frames. Fig. 4.17 illustrates the comparison of conventional 3D CNNs and Dissected 3D CNNs at online operation. Conventional 3D CNNs are non-dynamic since the final decision of the network might be triggered by any previous frame in the input clip, not due to the latest introduced frame. This is specifically critical for the

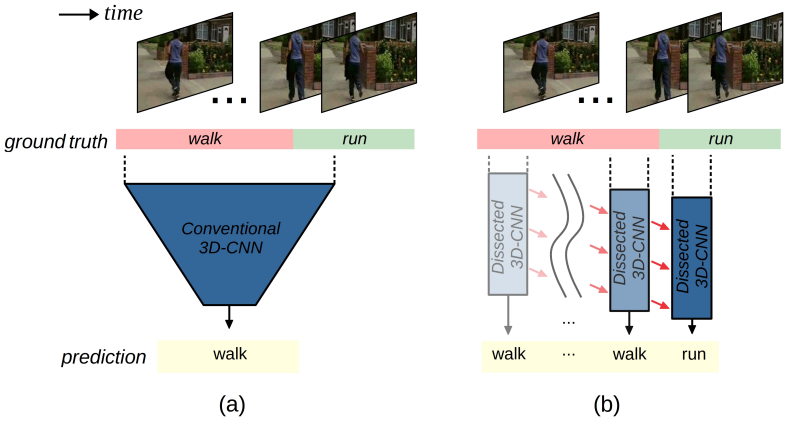


Figure 4.17: Comparison of Conventional 3D CNNs (a) and Dissected 3D CNNs (b) at online operation. Conventional 3D CNNs work with a fixed number of input frames and their predictions can be triggered by any frame in the input clip. Therefore, conventional 3D CNNs are *non-dynamic* and although a new action starts in the video stream, they can continue predicting the previous action, as illustrated in (a). On the other hand, Dissected 3D CNNs processes video with each new coming frame and update their prediction dynamically, as illustrated in (b). Red lines in (b) denote temporal skip connections.

online recognition of actions which is performed in very short time intervals, such as the Driver Micro Hand Gestures (DriverMHG) dataset [26]. On the other hand, Dissected 3D CNNs need to process only the most recent frame at online operation since they can leverage the previously computed intermediate volumes via a caching mechanism. Consequently, Dissected 3D CNNs can update their predictions according to the new coming frame and hence operated dynamically while enjoying reduced computational complexity.

To obtain the network's final decision, dissected 3D CNN architecture still needs a spatiotemporal modeling mechanism at the end. Although the conventional way of using a fully connected layer is a valid option, a Recurrent Neural Network (RNN) block can also be applied. The RNN block makes the D3D architecture independent of the number of input frames and performs better, as shown in our ablation study. Overall, Dissected 3D CNNs bring the following advantages:

1. D3Ds provide frame-level features.
2. D3Ds operate at any number of input frames.
3. Any 3D CNN architecture can be converted to its dissected version[1].

[1]D3D is a general term referring all the dissected 3D CNN architectures which employ temporal skip connections. For the dissected version of a specific network architecture, we use the prefix 'D' (e.g., the dissected version of ResNet-18 is denoted as D-ResNet-18).

4. A large number of computations are saved at online operation. Dissected versions of ResNet-18,50,101 perform 77-90% less computation at online operation while achieving ∼5% better classification accuracy compared to conventional ResNet models on Kinetics-600 dataset.

5. Any frame-level task can leverage from D3D architecture if the frames are obtained from continuous video streams.

4.5.2 Methodology

We first elaborate on the D3D architecture details, which reduces the computational complexity substantially during online operation. Secondly, we mention possible options for spatiotemporal modeling. Finally, training details are described.

4.5.2.1 Dissected 3D CNN Architecture

In order to demonstrate the advantages of the proposed D3D architecture, we have created the dissected version of the ResNet family (named as D-ResNet) and compared its performance with the conventional 3D-ResNet family as in [21]. The details of the proposed D-ResNet models and corresponding basic and bottleneck blocks are shown in Table 4.27 and Fig. 4.18, respectively. Similar to original ResNet architecture [88], spatial downsampling is performed at *conv*1, *pool*, *conv*3_1, *conv*4_1, and *conv*5_1 with a stride of 2. No temporal downsampling is employed. Unlike the 3D-ResNet architectures, we reduced the depth dimension of the initial convolutional layer of the basic block and the middle convolutional layer of the bottleneck block to 2 since we cache only previous intermediate volumes. We also modify the second convolutional layer of the basic block and set its depth dimension to 1. Excluding spatiotemporal modeling mechanisms, these modifications lead to parameter reduction of ∼50% on D-ResNet-18 and ∼23% on D-ResNet-50,101 compared to conventional 3D-ResNet architectures.

An illustration of Dissected 3D CNN architecture with basic D-ResNet block is shown in Fig. 4.19. The primary motivation to create such an architecture is to avoid the recomputation of already processed frames of the video stream during online operation. For that, intermediate volumes of the architecture are stored in a cache (blue region in Fig. 4.19) and used at inference. Throughout the inference, previous intermediate volumes in the cache are replaced with the current ones to be used in the next iteration. Therefore, only the computations within the yellow region in Fig. 4.19 are performed at online operation. Moreover, the designed D3D architecture does not employ (i) temporal downsampling and (ii) padding from right to ensure dynamic online operation.

At inference time, only the current frame or current frame with the previous two frames are passed to the network depending on the task at hand. The reason of leveraging the previous two frames is to capture pixel-wise motion information, which is critical for motion-intensive datasets such as Jester dataset [4]. At the first iteration, same padding is applied at *concat* operations since the cache for the intermediate volumes is empty. For D-ResNet-50,101 architectures, an additional *conv_last* block is used in order to reduce the output feature dimension from 2048 to 512. So, all D-ResNet architectures

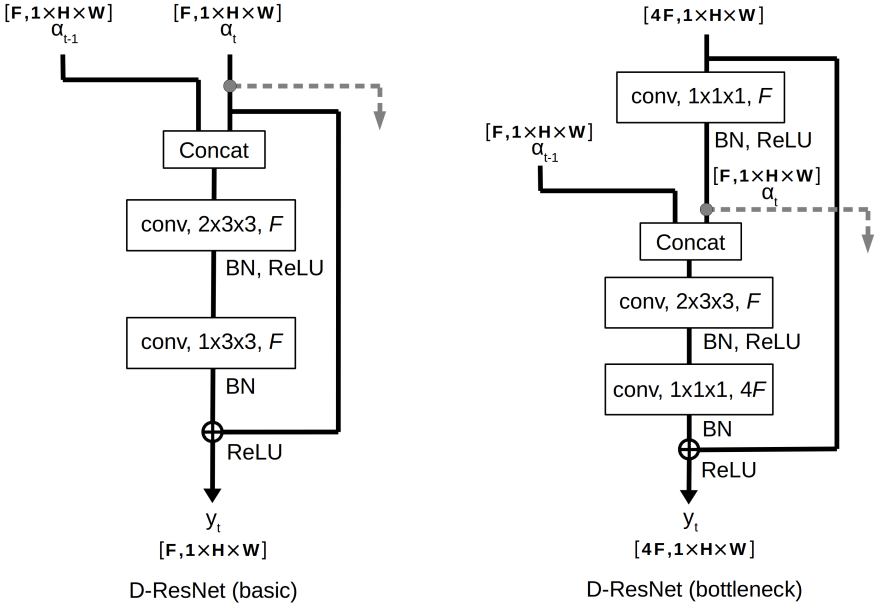


Figure 4.18: Basic and bottleneck blocks used in ResNet architecture. F, BN, and $ReLU$ denote the number of feature maps (i.e. channels), batch normalization [178], and rectified linear unit, respectively. *Concat* denotes concatenation at depth dimension while $\oplus$ denotes to element-wise addition.

produce a 512-dimensional feature vector for every frame. After obtaining frame-level features, a spatiotemporal modeling mechanism is required to produce class-conditional probabilities, which is explained in the next section.

4.5.2.2 Spatiotemporal Modeling Mechanism

The typical approach for spatiotemporal modeling is to conclude the network with a fully connected (fc) layer. This approach is also how we trained our architectures from scratch. However, the fc layer at the end of the network requires a fixed number of frames as input. Moreover, the dynamicity condition of the architecture is not met since the decision is made with all output features coming from each frame in the clip.

In order to achieve a dynamic architecture, we have considered two popular RNN blocks: Long Short-Term Memory (LSTM) [96] and gated recurrent unit (GRU) [241]. However, joint end-to-end training of the feature extraction and RNN blocks is not feasible due to the computational and memory complexity of back-propagating through the long video, as described in [202]. To this end, we have extracted the output features f (before the fully connected layer - see Fig. 4.20) of all video frames for the training and test set and trained the recurrent blocks separately. For example, each video in the Kinetics dataset lasts around 10 seconds, which makes 250 frames if the video is recorded with 25 fps. After applying the recurrent block, an fc layer is used at the last output of the recurrent block to map the hidden feature map to the number of classes. We

Layer	**ResNet-18**	**ResNet-{50,101}**
block	basic	bottleneck
conv1	conv({1,3}×7×7), stride (1, 2, 2) F:64	
pool	MaxPool(1×3×3), stride (1, 2, 2)	
conv2_x	N:2, F:64	N:3, F:64
conv3_x	N:2, F:128	N:4, F:128
conv4_x	N:2, F:256	N:{6, 23}, F:256
conv5_x	N:2, F:512	N:3, F:512
conv_last	—	conv(1×1×1), stride (1, 1, 1), F:512
	global average pooling, spatiotemporal modeling	

Table 4.27: Dissected ResNet architectures. F is the number of feature channels corresponding in Fig. 4.18, and N refers to the number of blocks in each layer. Depending on the number of frames used at inference time (only current frame or current frame with two previous frames), convolution kernel for *conv*1 layer is selected as (1×7×7) or (3×7×7).

have named the resulting network as purely dynamic D-ResNet-18 architecture since the network produces a decision using the most recent frame at every iteration. D-ResNet-18 architecture with LSTM spatiotemporal modeling mechanism is shown in Fig. 4.20. In the experiments section, we will validate the advantages of recurrent spatiotemporal modeling techniques.

4.5.2.3 Implementation Details

Learning: We initially train our D3D architectures with fc layer at the end. 19 frames are fed to the network, but only the last 16 output features are used for loss computation. The reason of feeding 3 frames more is to initialize the cached intermediate volumes properly. Stochastic Gradient Descent (SGD) is applied with standard categorical cross-entropy loss as an optimizer. The largest fitting batch size is selected for mini-batch size, which is typically in the order of 128 clips. The networks are trained from scratch with a learning rate initialized with 0.1 and reduced 3 times with a factor of 10^{-1} when the validation loss converges. For temporal augmentation, clips are selected from a random position in the video. For spatial augmentation, clips are selected from a random spatial position with a randomly selected scale from $\{1, \frac{1}{2^{1/4}}, \frac{1}{2^{3/4}}, \frac{1}{2}\}$ in order to perform multi-scale cropping as in [131]. For the case of stacked fc layers, a hidden dimension of 1024 is applied.

Model	Skip Connection	Params	MFLOPs	St-Model.	Acc. (%)
D-ResNet-18	None	15.94M	546	fc	58.74
D-ResNet-18	Summation	15.94M	546	fc	58.40
D-ResNet-18	Concatenation	20.66M	747	fc	61.41

Table 4.28: Performance comparison for different temporal skip connections at online operation on the Kinetics-600 validation set.

For the training of the RNN blocks, we again use SGD with identical learning rates. However, we apply different augmentation schemes. First, the number of input features is selected randomly between [16, '*number of frames in the video*'] and padded with zero to obtain a fixed size of input for all videos. In this way, the RNN blocks can learn all *short*-, *medium*- and *long*-range dependencies. Moreover, videos are temporally down-sampled by 2, 3 and 4 with probabilities of 30%, 14% and 11%, respectively. We also replaced random parts of the input features with noise to enable RNN blocks to ignore unrelated parts of the input. In order to increase regularization, we also leverage Gaussian noise with zero mean and 0.005 variance at the input features and 0.3 dropout rate at the hidden layers of RNN blocks. For the hidden layers of RNN blocks, the dimension is set to 1024.

Recognition: Clips are selected by a sliding window with the stride of 1 over the complete video for fc spatiotemporal modeling. Afterwards, class scores are averaged for all the clips. For RNN blocks, the complete input is fed to the network and the last output of the RNN block is used for the final prediction.

Implementation: Network architectures are implemented in PyTorch. Our code and pretrained models will be made publicly available at https://github.com/okankop/Dissected-3D~CNNs.

4.5.3 Experiments

4.5.3.1 Video Activity Recognition On Trimmed Datasets

We perform detailed ablation study on the Kinetics-600 [124] dataset. Kinetics-600 contains trimmed YouTube clips with an average duration of 10 seconds belonging to 600 different categories. For our evaluations, we have used the validation set of the Kinetics-600 dataset since annotations for the test set are not publicly available.

Comparison of different temporal skip connection operations: We first compare the performance of different temporal skip connection operations. Table 4.28 shows the comparison of applying summation, concatenation and no temporal skip connections on D-ResNet-18 architecture. For the sake of fairness, at each iteration, all networks receive the current frame together with the two previous frames as input and apply a 3D convolution layer as the first operation. For summation and no temporal skip connection, a 2D convolution layer is applied afterward, whereas for concatenation

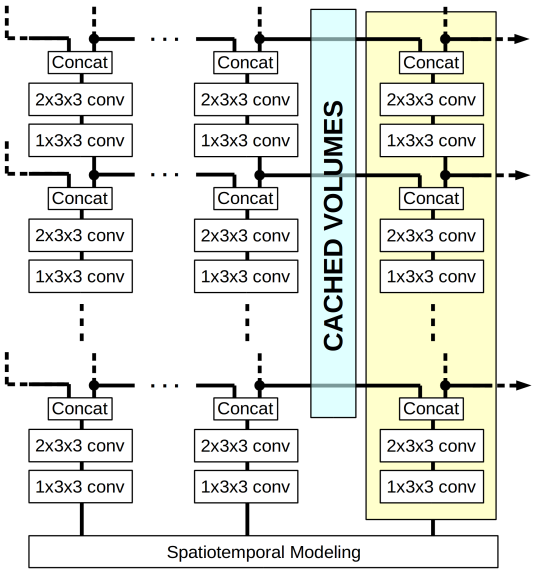


Figure 4.19: Proposed Dissected 3D CNN architecture using basic D-ResNet block. At the online operation time, intermediate volumes from the previous timestamp is stored in a cache (blue region), and only the computations for the current frame frame are performed (yellow region). Spatial skip connections are excluded for the sake of simplicity.

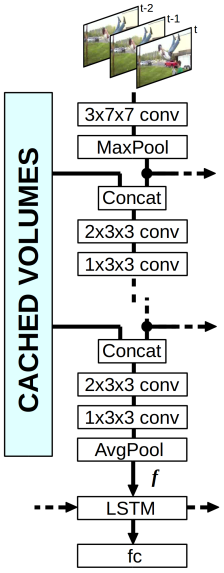


Figure 4.20: D-ResNet-18 architecture with LSTM spatiotemporal modeling mechanism. Spatial skip connections are excluded for the sake of simplicity.

temporal skip connection, a 3D convolution layer is used since volumes are concatenated along the depth dimension. Although using a 3D convolution layer increases the number of parameters and floating point operations, concatenation achieves the best performance with a margin of ∼2.7%.

We would like to note that summation does not bring any performance gain and even performs slightly worse than no temporal skip connection. We infer that this is due to the loss of temporal information after the summation operation.

It is also interesting to see that D-ResNet-18 with no skip connection achieves even better than conventional 3D ResNet-18 architecture in Table 4.30. This contradicts the findings of [128], where f-R2D achieves 1.3% worse accuracy than R3D. Our only difference from f-R2D in [128] is that we apply a 3D convolution layer at the first convolution operation, which was enough to capture necessary motion information to outperform R3D. Besides, we can conclude that preserving temporal resolution in the network (i.e. not applying temporal downsampling) increases classification

Layer	1-layer	2-layer	3-layer
GRU	61.08	61.43	61.38
LSTM	61.10	**62.02**	60.83
fc	61.41	61.25	61.22

Table 4.29: Accuracy on the Kinetics-600 validation set for different spatiotemporal modeling mechanisms using D-ResNet-18 architecture.

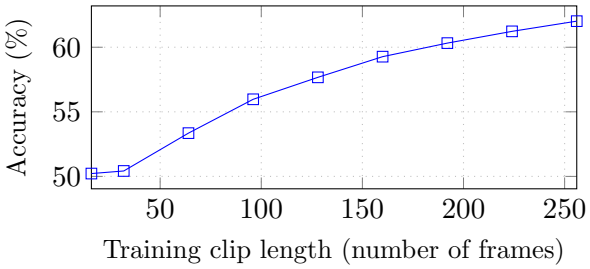


Figure 4.21: Influence of using different clip lengths at training on the accuracy of the Kinetics-600 validation set when training a D-ResNet-18-lstm.

performance, although this also increases the computation and memory load at inference time.

Analysis of different spatiotemporal modeling mechanisms: We investigate the performance of applying fc, LSTM or GRU as the spatiotemporal modeling mechanism. Table 4.29 shows the comparison of fc with LSTM and GRU for different numbers of hidden layers. Using multi-layer fc as the spatiotemporal modeling mechanism slightly reduces performance. Both recurrent blocks perform better with 2 hidden layers while LSTM achieves the best performance. Hence, from this point onwards, we always use two layers for LSTM. However, we must note that RNNs, in general, need more parameters and FLOPs compared to fc. For example, single layer fc requires 4.9M parameters and 9.8M FLOPs, whereas 2 layer LSTM requires 15.3M parameters and 108.2M FLOPs for the settings explained in Section 4.5.2.3.

Effect of different clip lengths on training recurrent blocks: At the training of RNN blocks, clip length plays an important role in the final classification performance. We have investigated the effect of different clip lengths at training time on the classification performance for D-ResNet-18-LSTM, as illustrated in Fig. 4.21. The results clearly show that a longer clip leads to higher classification accuracy. The reason is that LSTMs can learn the important/unimportant features and store/remove them in their cell state more easily when they observe longer clips.

Performance comparison of D-ResNet architectures with different depths: Comparative results are shown in Table 4.30. As usual, increasing network depth

Model	Params	Cache Size	MFLOPs	Speed	St-Model.	Acc. (%)
3D-ResNet-18 [21]	33.24M	–	8323	3.00	fc	57.65
3D-ResNet-50 [21]	44.24M	–	9835	5.46	fc	63.00
3D-ResNet-101 [21]	83.29M	–	13664	7.04	fc	64.18
D-ResNet-18	20.66M	0.97MB	747	0.33	fc	61.41 +3.76
D-ResNet-50	40.81M	1.95MB	1654	0.75	fc	67.35 +4.35
D-ResNet-101	69.83M	2.81MB	3077	1.46	fc	68.78 +4.60
D-ResNet-18	31.05M	0.94MB	845	0.33	LSTM	62.02 +4.37
D-ResNet-50	51.20M	1.92MB	1752	0.75	LSTM	68.22 +5.22
D-ResNet-101	80.22M	2.78MB	3175	1.46	LSTM	**69.17** +4.99

Table 4.30: Comparison of D-ResNet architecture with conventional ResNet architecture over offline classification accuracy, number of parameters, computation complexity (FLOPs) at online operation on the Kinetics-600 validation set. The cache size is calculated according to 32 bit floating point values for intermediate volumes and reported in megabytes (MB). For each architecture, the speed refers to single inference time measured in millisecond (ms) using Nvidia Titan XP GPU for a batch size of 8.

yields higher accuracies. Moreover, D-ResNet performs 74-90% less computation at online operation while achieving ∼5% better classification accuracy compared to conventional ResNet models on Kinetics-600. This is since D3D uses the previous computations efficiently by caching the intermediate volumes of the network. The performance improvement is not also due to the increased number of parameters since D-ResNet models have fewer parameters compared to conventional 3D-ResNet models. The only exception is D-ResNet-50 with LSTM, which has around 7M more parameters compared to 3D-ResNet-50.

Required cache size for D-ResNet architectures: The amount of memory needed for cache to store intermediate volumes depends on (i) applied data precision (e.g., 32-bit, 16-bit or 8-bit floating point values), (ii) used input resolution and (iii) the number of stored intermediate volumes. Accordingly, we report the required cache size in Table 4.30. As expected, deeper architectures store more intermediate volumes in the cache, hence require more memory.

Causality analysis of D-ResNet-18 architecture: The essential property of an online system is that the architecture should be causal. To validate the causality of the proposed D3D, we designed two tests. Firstly, we make a segment-level classification test, where we have divided input videos into ten equal parts, and outputs are averaged within each segment. Fig. 4.22 (a) shows the comparison of fc and LSTM. Since fc treats each clip independently and the middle parts of the videos are typically more informative, a bowed curve is achieved. On the other hand, LSTM stores the relevant features in its cell state over time, leading to increased accuracy with rising segment numbers. Keeping in mind that an entirely causal system should improve monotonically, D3D satisfies this criterion. Secondly, we have replaced the middle parts of the videos with the Gaussian

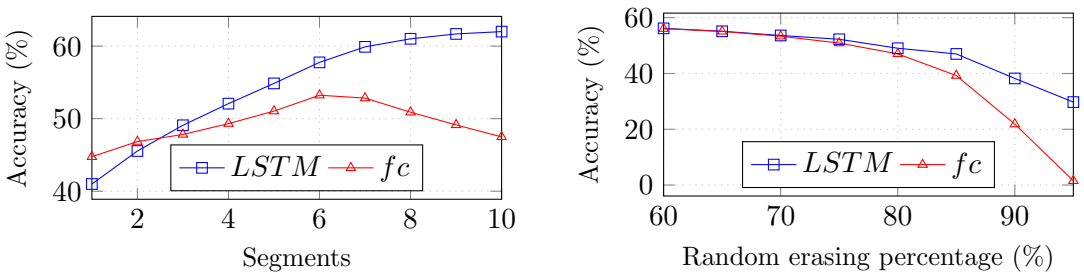


Figure 4.22: Causality analysis of deployed spatiotemporal modeling mechanisms. In (a), videos are separated into ten equal segments and network outputs at each segment are averaged for fc and $LSTM$. In (b), the network outputs at the middle parts of the videos are replaced with the Gaussian noise.

Model	Input Resolution	MFLOPS	Acc. (%)
3D-ShuffleNetV1 2.0x [21]	112×112	538	56.84
3D-ShuffleNetV2 2.0x [21]	112×112	438	55.17
3D-MobileNetV1 2.0x [21]	112×112	662	48.53
3D-MobileNetV2 1.0x [21]	112×112	561	50.65
3D-SqueezeNet [21]	112×112	926	40.52
3D-ResNet-18 [21]	112×112	8323	57.65
3D-ResNet-50 [21]	112×112	9835	63.00
3D-ResNet-101 [21]	112×112	13664	64.18
3D-ResNeXt-101 [21]	112×112	9652	68.30
I3D [124]	224×224	111331	71.90
Oct-I3D [242]	224×224	25600	76.00
X3D-M [132]	224×224	6200	78.80
X3D-XL [132]	224×224	48400	81.90
SlowFast [130]	224×224	234000	81.80
D-ResNet-18	112×112	845	62.02
D-ResNet-50	112×112	1752	68.22
D-ResNet-101	112×112	3175	69.17

Table 4.31: Comparison of D-ResNet architecture with state-of-the-art on validation set of Kinetics-600 dataset. FLOPs are calculated for the inference to produce one output at online operation.

noise and reported the performance comparison of fc and LSTM in Fig. 4.22 (b). As we increase the erased percentage from the middle part of the videos, accuracy drops linearly for both fc and LSTM till 60% erasure. If we keep increasing the erasure percentage, it becomes more and more important to use the information at the beginning and end

of the videos jointly. Therefore, LSTM outperforms fc more and more as the erasure percentage increases. Specifically, with an erasure percentage of 95%, fc achieves 1.53% accuracy, whereas LSTM achieves 29.74% accuracy.

State-of-the-art comparison: Although the motivation of this work is not beating the state-of-the-art, it is beneficial to see how well D3D architecture performs compared to other architectures. Accordingly, we have compared D3D architecture with the state-of-art architectures in Table 4.31.

We first note that FLOPs numbers in Table 4.31 are calculated according to the necessary number of floating point operations to produce one output at online operation. Therefore, FLOPs are calculated for the inference of 'input clip' for all architectures other than D-ResNet family. Secondly, increased input resolution in general leads to better performance, but the computational complexity also increases quadratically. Thirdly, we would like to emphasize again that any 3D CNN architecture in Table 4.31 can be converted to its dissected version. X3D-XL [132] and SlowFast [130] achieves the best performance on the Kinetics-600 dataset, but also require significantly high FLOPs at online operation. Therefore, their dissected version can satisfy both high accuracy and low computational complexity at online operation. If resource efficiency is the main concern, the dissected version of resource efficient architectures such as 3D-ShuffleNetV1 2.0x [21] would require very little computation with less than 100 MFLOPs.

4.5.3.2 Video Activity Recognition On Untrimmed Datasets

The experiment in Fig. 4.22 (b) shows that videos can be successfully recognized with D3D architectures with LSTM spatiotemporal modeling although some parts of them contain activity-unrelated content. Therefore, we have investigated the performance of D3D on untrimmed ActivityNet dataset [9]. The videos in the ActivityNet dataset are on average 117 seconds long and contain activities from 200 different classes. Therefore, with this dataset, we can also test the performance of D3D on longer videos. Similar to Kinetics-600, we have evaluated the performance of D3D on the validation set of the ActivityNet dataset since the test set is not publicly available. Used D-ResNet architectures are exactly same as Kinetics-600 experiments except for the last fc layer, which reduces the number of outputs according to the ActivityNet class number that is 200. All architectures are first pretrained on the Kinetics-600 dataset and then fine-tuned on the ActivityNet dataset.

In Table 4.32, we compare the performance of D-ResNet with the conventional 3D-ResNet architectures. Firstly, the ActivityNet dataset is not as large as Kinetics-600, hence deeper D-ResNet-101 performs worse than D-ResNet-50 due to overfitting. Secondly, D-ResNet achieves around 3% better video classification accuracy compared to conventional 3D-ResNet architectures similar to Kinetics-600 dataset. We conjuncture that this is due to the temporal resolution preserving property of D3D architectures. Lastly, using LSTM instead of fc does not improve accuracy as dramatically as the Kinetics-600 dataset. This shows that increased video length is not that useful for the ActivityNet dataset since it is an untrimmed dataset

Model	St-Modeling	Accuracy (%)
3D-ResNet-18 [21]	fc	59.99
3D-ResNet-50 [21]	fc	68.14
3D-ResNet-101 [21]	fc	67.87
D-ResNet-18	fc	63.01 +3.02
D-ResNet-50	fc	71.32 +3.18
D-ResNet-101	fc	70.97 +3.10
D-ResNet-18	LSTM	63.33 +3.34
D-ResNet-50	LSTM	**71.50** +3.36
D-ResNet-101	LSTM	70.71 +2.84

Table 4.32: Comparison of D-ResNet architecture with conventional 3D-ResNet architecture over video classification accuracy on the validation set of untrimmed ActivityNet dataset.

and most of the video content does not contain auxiliary information for the correct classification of the video. Accordingly, we conclude that longer videos are favorable as long as all video content contributes to the correct prediction of the performed activity. The results of D-ResNet-18 with LSTM in Fig. 4.22 (a) justifies this argument.

4.5.3.3 Gesture Recognition

Gesture recognition and action recognition can be viewed as similar tasks. In action recognition, although it is still necessary to capture motion patterns, the network especially needs to capture spatial patterns. For example, In the Kinetics-600 dataset, there are nine different "eating something" classes where "something" is one of "burger, cake, carrot, chips, doughnut, hotdog, ice cream, spaghetti, watermelon". For the correct classification, the network must recognize the objects in the videos correctly. On the other hand, the spatial content in gesture videos is similar: A person in front of a camera performing a hand gesture. For the correct classification, the motion of the hand must be captured by the network.

To inspect the D3D architecture's ability to capture motion patterns, we have experimented with the Jester dataset [4], which is the largest available hand gesture dataset currently. Training details are kept exactly the same as previous settings. In Table 4.33, D-ResNet-18 achieves 1.24% more classification accuracy than conventional 3D-ResNet-18.

4.5.3.4 Video Person Re-Identification (ReID)

Person re-identification aims to match a queried data with its true owner in the gallery set. In video person ReID, both the query and gallery are person tracklets, which usually consist of a varying number of frames. Most state-of-the-art approaches leverage 2D CNN architectures for video ReID [243, 244, 245]. However, 2D CNN architectures process

Model	St-Modeling	Accuracy (%)
3D-ResNet-18 [21]	fc	93.34
D-ResNet-18	fc	**94.58** +1.24

Table 4.33: Comparison of dissected and conventional ResNet-18 architectures on the Jester validation set. Both architectures take 16-frames input (downsampling of 2 is applied) with 112×112 spatial resolution.

Model	Accuracy (%)	mAP
2D-ResNet-50	80.8	69.0
D-ResNet-50	**81.3** +0.5	**69.1** +0.1

Table 4.34: Comparison of our D-ResNet-50 architecture with 2D-ResNet-50 on the validation set of the MARS dataset.

individual frames independently, hence they cannot incorporate temporal information between frames. In this section, we demonstrate that our proposed D3D architecture can increase the performance over 2D CNNs.

Our person ReID architecture is as follows. Given input video clips, a backbone network extracts features for each frame and these features are averaged to get the final feature representing the given input clip. We utilized classification loss and triplet loss in order to train the network. For the classification loss, we consider person identities as category-level annotations and train a linear layer followed by a softmax operation to get class-conditional probabilities. Then, our classification loss $\mathcal{L}_\mathcal{C}$ is the cross-entropy error between the predicted classes and the ground truth classes. For the triplet loss $\mathcal{L}_\mathcal{T}$, our data loader randomly selects N video clips for each person, which is used for hard sample mining [246]. The final loss is $\mathcal{L} = \mathcal{L}_\mathcal{C} + \mathcal{L}_\mathcal{T}$. The architecture is trained end-to-end using the final loss $\mathcal{L}$. In our experiments, we used $N = 4$ and each clip contains 4 frames at training time. At test time, we have loaded all frames in person videos to get final video features.

In our experiments, we have used the MARS dataset [247] for performance evaluation. For the backbone network in the architecture described above, we have compared the conventional 2D-ResNet-50 with our D-ResNet-50 architecture. Both models are inflated from ImageNet pretrained model. We trained the networks for 150 epochs using Adam optimizer with an initial learning rate of 0.0003, which is divided by 10 every 60 epochs. In the MARS dataset, all person detections are already cropped, hence there is no pixel-wise correspondence at consecutive frames in tracklets. Therefore, we used single frames at the input of the D-ResNet-50 architecture. The comparative results are shown in Table 4.34. D-ResNet-50 architecture manages to capture discriminative motion information of identities, possibly gait-related information, which slightly increases the performance.

Model	Fusion	Accuracy (%)			
		5	25	50	100
2D-ResNet-18	avg all	**93.18**	93.66	93.64	93.66
D-ResNet-18	avg all	92.40	**93.90**	93.74	93.74
D-ResNet-18	avg 5:end	93.12	93.80	**93.92**	**94.10**

Table 4.35: Evaluation on YouTube Faces dataset resampled to different number of frames.

4.5.3.5 Video Face Recognition

In the domain of video face recognition, typical approaches [248, 249, 250] leverage the features obtained by training on big datasets containing still images followed by simple average pooling of the features without emphasis on the quality of every frame. More sophisticated approaches combine the feature extraction network to aggregate the features based on their importance with a feature aggregation network [251, 252, 253, 254]. However, temporal information is discarded as frames are treated as an unordered set of faces. Compared to these approaches, our D3D architecture can cope with this task while only consisting of one single network.

For the video face recognition task, we use the VoxCeleb2 dataset [255] for pretraining the architectures and YouTube Faces dataset [256] to evaluate them. Before training the network, we preprocess the VoxCeleb2 dataset by extracting 3 frames per clip, which are aligned using facial landmarks extracted using the MTCNN [257] and cropped to 112×112 pixels. First, we pretrain a 2D-ResNet-18 with a 256-dimensional bottleneck layer on single image recognition on the VoxCeleb2 dataset using cross-entropy loss with Adam optimizer, 50% dropout, an initial learning rate of 0.05 and a batch size of 100 for 50 epochs. We decide against pretraining on a bigger dataset containing still images, as otherwise, the adaption to D-ResNet-18 gets overshadowed by the dataset change. For training the D-ResNet-18, we inflate the weights of the 2D-ResNet-18 and fine-tune using 5 frames per sample and a frame at the input with a lower learning rate of 0.01 and additional motion blur data augmentation for 1 epoch. Apart from these changes, parameters are identical to the pretraining. Our experiments show that motion blur data augmentation does not improve the accuracy of the 2D-ResNet-18, whereas it improves accuracy while fine-tuning the D-ResNet-18.

We evaluate our approach on the YouTube Faces dataset following the standard verification protocol. We compute the Euclidean distance after l2-normalization and taking the average of the features. The preprocessing is done similarly to the VoxCeleb2 dataset. However, we resample the videos to obtain a fixed number of frames to show the dependency of the accuracy on the number of frames as shown in Table 4.35. In contrast to the 2D-ResNet-18, our D-ResNet-18 continues to improve with the increased number of frames. We also evaluate discarding the first four features in the average due to cache initialization (denoted by avg 5:end), which results in another minor improvement. Note that for 5 frames, *avg 5:end* is equal to taking only the last feature, which is substantially higher than the accuracy of the

2D-ResNet-18 for a single frame per video (88.78%). This demonstrates that our network is capable of propagating useful information through time.

4.5.4 Summary

In this work, we have addressed the computational complexity drawback of 3D CNNs and proposed a novel Dissected 3D CNN (D3D) architecture. The D3D architecture caches the intermediate volumes of the network and propagates them for future calculation, which reduces the computation around 77-90% during online operation for D-ResNet family. Besides reducing complexity during online operation, the D-ResNet family achieves ∼5% higher classification accuracy compared to the classical ResNet family on the Kinetics-600 dataset. We believe that this performance improvement arises since D3D networks are temporal resolution preserving and produce fine-grained features. In this work, only the ResNet family is converted to its dissected version and evaluated. However, any CNN architecture can be converted to its dissected version for efficient online video processing. The proposed D3D architecture successfully models temporal information and can be employed at any video-based computer vision task. In our experiments, we have successfully validated the effectiveness of D3D architecture on five different vision tasks: activity/action recognition on trimmed and untrimmed datasets, gesture recognition, video person re-identification, and video face recognition. For all these tasks, D3D consistently improves performance. We believe that the D3D architecture will be actively used in many other video-based tasks by the vision community.

Chapter 5

Audio-Visual Video Analysis

This chapter presents video analysis with audio and visual modalities. First, we present the motivations of applying audio-visual video analysis in Section 5.1. We also list several video analysis tasks, which can intensively benefit from mutual leveraging of audio and visual modalities. Afterwards, in Section 5.2, we show how to incorporate audio and visual modalities for the task of active speaker detection.

5.1 Introduction

Video analysis is usually conceived as the visual inspection of videos for a given task. Although audio is also a natural component of videos, it is often overlooked by researchers at video analysis tasks. However, audio and visual modalities contain complementary information and result in improved performance if used together.

There are various video analysis tasks, which can intensively benefit from mutual usage of audio and visual modalities. [258] proposes a neural network architecture to learn audio-visual representations in a self-supervised way for the tasks of sound source localization, audio-visual action recognition, and on/off-screen audio source separation. In [259], the audio-visual temporal synchronization of videos is used in a self-supervised mechanism to learn general and effective models for both the audio and the vision domain. [260] develops a neural network model for visual object segmentation and sound source separation tasks, which is trained in a self-supervised way from natural videos. Similarly, [261, 262] propose approaches for audio-visual source separation with self-supervised learning. On the other hand, [263] make use of keypoint-based structured visual representations for visual sound separation.

For action recognition, [264, 265] leverages multi-stream architectures to process audio and visual modalities. Approaches in [266, 267] make use of audio and video modalities for emotion recognition. There are also several approaches for audio-visual speech recognition [268, 269, 270]. There are also several approaches for audio-visual active speaker detection [153, 154, 167, 168, 169].

For all the tasks mentioned above, either given task cannot be performed if not audio and visual modalities are leveraged jointly, or joint multi-modal learning results in improved performance compared to the single modality cases. For the task of active speaker detection, audio and visual modalities have to be used jointly to achieve a decent recognition performance. In Section 5.2, based on a series of controlled

experiments, we also present a three-stage architecture for the task of audio-visual active speaker detection.

5.2 Active Speaker Detection

This section presents an audio-visual video analysis application, active speaker detection, where audio and video modalities have to be used jointly in order to provide complementary information.

Successful active speaker detection requires a three-stage pipeline: (i) audio-visual encoding for all speakers in the clip, (ii) inter-speaker relation modeling between a reference speaker and the background speakers within each frame, and (iii) temporal modeling for the reference speaker. Each stage of this pipeline plays an important role in the final performance of the created architecture. Based on a series of controlled experiments, this work presents several practical guidelines for audio-visual active speaker detection. Correspondingly, we present a new architecture called ASDNet, which achieves a new state-of-the-art on the AVA-ActiveSpeaker dataset with an mAP of 93.5% outperforming the second best with a large margin of 4.7%. This section is based on our publication *How to Design a Three-Stage Architecture for Audio-Visual Active Speaker Detection in the Wild* [24].

5.2.1 Motivation

The fusion of audio and video modalities has been shown to provide promising solutions to long-standing challenging problems. These include, among others, speaker diarization [271], biometrics [255], and action recognition [272, 273]. Similar to other tasks, AV-ASD has also long been studied in literature [165, 166]. A particularly challenging flavor of this problem is AV-ASD in the wild, where speech is to be detected and assigned to one of possibly multiple active speakers at each instant in time. Clearly, fusing the complementary discriminative information from audio and video modalities is crucial: visual-only approaches can easily be mistaken by other face/mouth motions such as eating, yawning or emotional expressions. Audio-only approaches, although able to perform source clustering and separation [274, 275], aren't sufficiently robust to count the number of speakers and assign speech to the correct source. This is especially challenging with a single microphone input in acoustically adverse conditions, typically encountered in practice.

Recently, the AVA-ActiveSpeaker dataset [167] provided the first large-scale standard benchmark for audio-visual active speaker detection in the wild. Recent research [154, 153] indicates that active speaker detection in the wild requires (i) integration of audio-visual information for each speaker, (ii) contextual information that captures inter-speaker relationships, and (iii) temporal modeling to exploit long term relationships in natural conversation. In this paper, we consolidate this three-stage pipeline for audio-visual speaker detection, illustrated in Fig. 5.1, and study the importance of each stage in detail.

Contributions. We propose a novel three-stage pipeline for audio-visual active speaker detection in the wild. Our architecture, named ASDNet, sets a new state-of-the-art result on AVA-ActiveSpeaker dataset with a 93.5% mAP, and outperforms the second

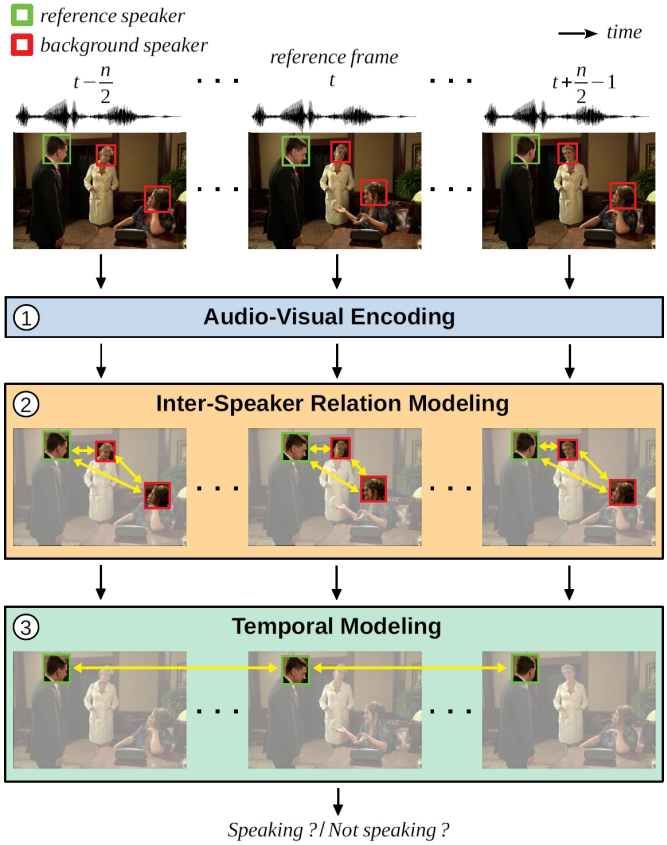


Figure 5.1: Audio-visual active speaker detection pipeline. The task is to determine if the reference speaker at frame t is *speaking* or *not-speaking*. The pipeline starts with audio-visual encoding of each speaker in the clip. Secondly, inter-speaker relation modeling is applied within each frame. Finally, temporal modeling is used to capture long-term relationships in natural conversations. Examples are from AVA-ActiveSpeaker dataset [167].

best method [153] with a large margin of 4.7% mAP (Section 5.2.3.5). As part of ASDNet, we propose:

(1) Architectures for the audio and video backbones of the audio-visual encoder (Section 5.2.2.2), that haven't been previously explored for active speaker detection;

(2) A simple, yet effective inter-speaker relation modeling mechanism (Section 5.2.2.3);

(3) In addition, we provide a detailed ablation study and guidelines for tuning all components of ASDNet. The study includes comparison to the state-of-the-art for the two novel components mentioned above, as well as evaluation of various RNN architectures for temporal modeling (Section 5.2.3.2.).

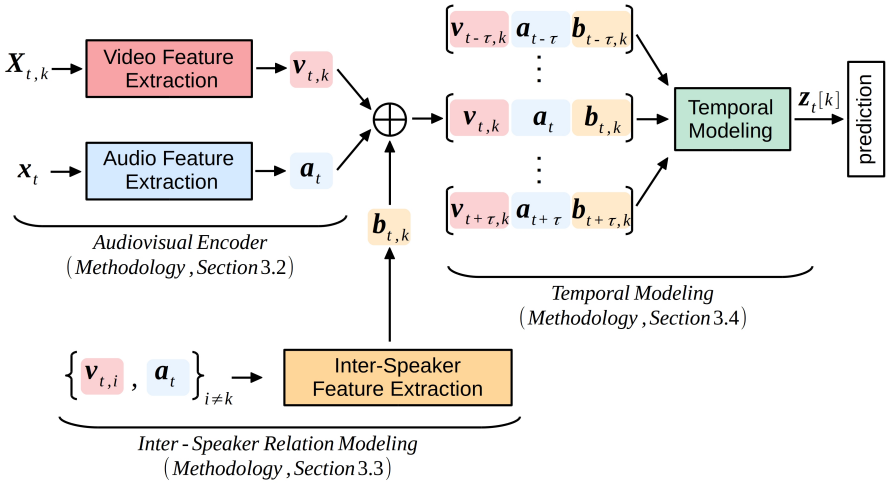


Figure 5.2: Overview of the three-stage pipeline in ASDNet.

5.2.2 Methodology

Drawing inspiration from the insights in recent research, we seek to establish a general pipeline that incorporates audio-visual encoding, inter-speaker (context) modeling, and temporal modeling. By designing an appropriate architecture for each component, we are able to exceed the state-of-the-art performance on the AVA-ActiveSpeaker dataset.

5.2.2.1 Notation and Overview

Let K denote the total number of speakers in a given clip. The data available to the active speaker detection system at time t is a set $\mathcal{X}_t = \{\boldsymbol{X}_{t,1}, \boldsymbol{X}_{t,2}, \dots \boldsymbol{X}_{t,K}, \boldsymbol{x}_t\}$, where $\boldsymbol{X}_{t,k} \in \mathbb{R}^{n \times 3 \times d_h \times d_w}$ is a tensor of face crops corresponding to the k-th speaker. The height and width of the face crops are denoted by d_h and d_w, 3 is the RGB channels and n is the number of consecutive face crops centering time instant t. The vector $\boldsymbol{x}_t$ contains the samples of the audio track corresponding to the duration of the video input. Given the input data, the objective is to produce a binary vector $\boldsymbol{z}_t$, where $\boldsymbol{z}_t[k] = 1$ if the k-th speaker is detected as *speaking* at time frame t, and $\boldsymbol{z}_t[k] = 0$ otherwise.

A high-level overview of our pipeline that maps the raw data $\mathcal{X}_t$ to the predictions $\boldsymbol{z}_t$ is illustrated in Fig. 5.2. Next, in Sections 5.2.2.2-5.2.2.4, we zoom in on the design of the three pipeline components. In Section 5.2.2.5, we discuss the training strategy that enables an end-to-end inference: from face crops and an audio waveform to a prediction *speaking* or *not speaking* for each speaker in the video clip.

5.2.2.2 Audio-Visual Encoder Architecture

Our audio-visual encoder is illustrated in Fig. 5.3. The stack of face thumbnails $\boldsymbol{X}_{t,k}$ consists of n frames, $X_{t-\frac{n}{2},k}, \dots, X_{t,k}, \dots, X_{t+\frac{n}{2}-1,k}$, and the size of the audio input vector $\boldsymbol{x}_t$ is determined by the number of video frames, the video frame rate, and the

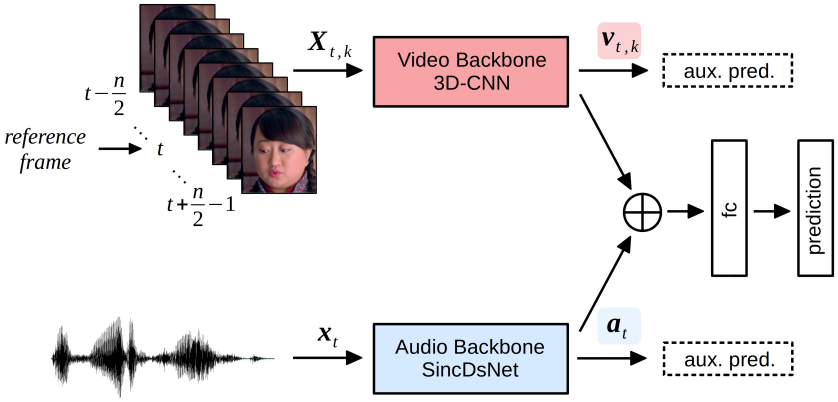


Figure 5.3: Audio-visual encoder architecture. Visual input $\boldsymbol{X}_{t,k}$ and audio input $\boldsymbol{x}_t$ are fed to the respective backbones to produce features $\mathbf{v}_{t,k}$ and $\mathbf{a}_t$. A concatenated feature vector $\mathbf{v}_{t,k} \bigoplus \mathbf{a}_t$ is fed to a fully connected layer which produces a prediction if speaker k is speaking at time t. Prediction heads are removed after training and are not part of the global picture in Fig. 5.2.

audio signal sampling rate. The encoder produces an embedding vector by concatenating the modality-specific embeddings:

$$\mathbf{v}_{t,k} = f_v(\boldsymbol{X}_{t,k}; w_v), \quad \mathbf{a}_t = f_a(\boldsymbol{x}_t; w_a), \tag{5.1}$$

where f_v and f_a are neural networks with trainable parameters w_v and w_a, respectively.

The concatenated features $\mathbf{v}_{t,k} \bigoplus \mathbf{a}_t$ are fed into a fully connected layer to get final predictions. To train the audio-visual encoder, we apply cross-entropy loss between the predictions and ground-truth labels. To ensure that consistent discriminative features are extracted from both modalities, we apply auxiliary classification networks after each backbone, following previous works [167, 154, 153]. The auxiliary networks are also trained with cross-entropy loss. The final loss becomes $L_{final} = L_{av} + L_a + L_v$. After training is completed, supervision heads are discarded and only the audio-visual backbone is used to extract features $\mathbf{v}_{t,k}$ and $\mathbf{a}_t$ for all speakers and time instants.

While the described high-level architecture is similar to that of existing audio-visual encoders [167, 154, 153], our contribution lies in the choice and design of the video and audio backbones, discussed next.

Video backbone. Movements of mouth and facial muscles are indicative of active speaking. Hence, to fully exploit the available video data, it is important to accurately model motion patterns. To this end, we propose using a 3D-CNN as the visual encoder function f_v, in contrast to the state-of-the-art approaches that apply 2D-CNN [153, 154, 167, 168, 169]. As part of our study, we experimented with various resource efficient and high-performance 3D-CNN architectures [21] and found 3D-ResNeXt-101 to be the best performing candidate for our video backbone. Further insights from our investigation are discussed in Section 5.2.3.1.

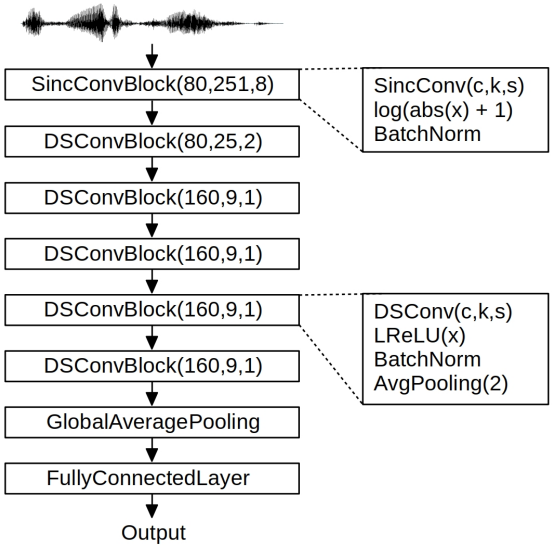


Figure 5.4: Audio encoder utilizing Sinc Convolutions (SincConv) and Depthwise Separable Convolutions (DSConv). The convolution parameters, *c, k, s* corrspond to the number of output channels, kernel size, and stride, respectively.

Audio backbone. For the audio encoding backbone, the majority of existing AV-ASD approaches [167, 154, 153, 168, 169] extract Mel Frequency Cepstral Coefficients (MFCC) from the raw signal, and use the MFCCs as input to 2D CNNs. In contrast, we propose using an audio backbone architecture that directly operates on raw audio signal via sinc convolutions [158]. In this manner, the system doesn't require a dedicated filterbank and directly exploits all available audio information. This is not the case in existing approaches, where phase information is often discarded after the filterbanks. After sinc convolutions, we apply log-compression, i.e., $y = log(abs(x) + 1)$. This non-linearity has been effective in other raw audio processing tasks as well [159, 276]. The features extracted by the sinc-convolutions are used as input to Depthwise Separable Convolutional (DSConv) blocks with Leaky-ReLU nonlinearity [277]. Our full audio encoder architecture, referred to as *SincDSNet*, is shown in Fig. 5.4. Features after the global average pooling are extracted as the audio features $\mathbf{a}_t$. The advantage of the proposed raw-audio backbone over existing feature-based backbones is experimentally demonstrated in Section 5.2.3.1.

5.2.2.3 Inter-Speaker Relation Modeling (ISRM)

The audio-visual encoder extracts features for each individual speaker separately - the features for speaker k do not contain visual information from the remaining speakers in the frame. However, features belonging to background speakers contain complementary information that improves the system performance, as shown in [154]. In this paper, we propose a method to aggregate information from the background speakers efficiently.

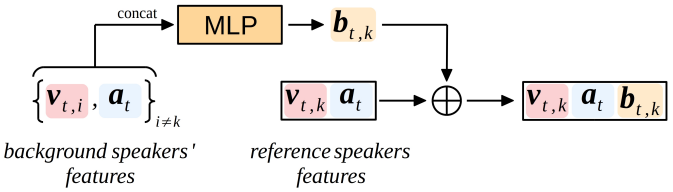


Figure 5.5: Inter-speaker relation modeling architecture. For reference speaker k at time instant t, we extract background features $\mathbf{b_{t,k}}$ by passing the concatenated features of background speakers through one layer MLP. Extracted features are then concatenated to reference speakers video features and audio features.

Consider a reference speaker k and m background speakers in the scene at time t. The output of the audio-visual encoder for the reference speaker is $[\mathbf{v}_{t,k}, \mathbf{a}_t]$. To incorporate information from background speakers, we propose to extract an additional feature vector $\mathbf{b}_{t,k}$ using a single-layer perceptron, as illustrated in Fig. 5.5. The input to the MLP are the concatenated audio-visual embeddings from all background speakers at time t. Note that the number m is fixed from the system's perspective: if there are less than m background speakers at time t, the encoder features are populated with zero vectors. If there are more than m speakers, only m are randomly selected. In this manner, the input dimension of the MLP is fixed. The final feature vector $[\mathbf{v}_{t,k}, \mathbf{a}_t, \mathbf{b}_{t,k}]$ is fed to the temporal model. An experimental study of the proposed ISRM, and comparison to the approach in [154] is provided in Section 5.2.3.2.

5.2.2.4 Temporal Modeling

Speaking is a coherent action in time: if a person is speaking at previous or future time instants, it is likely that the person is speaking at the current time instant. This is also valid for *remaining silent* action. Therefore, temporal modeling is crucial for accurate active speaker detection.

We experimented with several RNN-based temporal modeling architectures: Long Short-Term Memory (LSTM) [96], Gated Recurrent Unit (GRU) [99], Simple Recurrent Unit (SRU) [278] and their bidirectional versions. For the uni-directional methods, the reference frame is at the end of the input, while for the bidirectional methods it is at the center of the input. The hidden state vector of the recurrent block at the reference frame is fed to a fully connected layer to produce a binary output $\mathbf{z}_t[k] \in \{0, 1\}$ (i.e. active speaker or not). In case speakers' features are not available for the selected time window, similar to [154] we apply same padding to the beginning or to the end. Out of all methods, Bidirectional-GRU performs best and becomes our final choice in the temporal modeling stage.

5.2.2.5 Training Details

Training Audio-Visual Encoding Backbones. We train our audio-visual encoder using ADAM optimizer [279] for 70 epochs. Batch size is selected as the highest possible

number that fits to a single Nvidia Titan XP GPU for different backbones. However, gradients are accumulated reaching to effective batch size of 192 before doing backward propagation. The learning rate is initialized with 3×10^{-4} and dropped by a factor of 10 at every 30 epochs. For video input, we apply random cropping, random horizontal flipping and color transformations as data augmentation at the training time. Finally, video input is reshaped to the resolution of 160×160. The audio signals are sampled at 16 kHz. 3D CNNs are pretrained on Kinetics [127], 2D CNNs are pretrained with ImageNet [82], and SincDSNet is trained from scratch. Once the training is finished, prediction heads are discarded and the features $\mathbf{v}_{t,k} \in \mathbb{R}^{512}$ and $\mathbf{a}_t \in \mathbb{R}^{160}$ are used to train the ISRM and the temporal model.

Training ISRM and Temporal Modeling. We used ADAM optimizer with cross-entropy loss to train the ISRM and the temporal model. We train for 10 epochs, with batch size of 256. The learning rate is initialized with 3×10^{-6} and dropped by 10 at the 5^{th} epoch. The MLP in the ISRM extracts the feature $\mathbf{b}_{t,k} \in \mathbb{R}^{128}$ independent from the number of background speakers. For the temporal model, we used two recurrent layers with a hidden state dimension of 128, which experimentally proved to be optimal for our system.

Implementation. Our final architecture ASDNet is implemented in PyTorch and all experiments are performed using a single Nvidia Titan Xp GPU. We make our code publicly available at `https://github.com/okankop/ASDNet`.

5.2.3 Experiments

Dataset. We have used the AVA-ActiveSpeaker dataset [167] in our experiments, whose details are provided in Section 2.5.

Evaluation Metric. We use the official ActivityNet evaluation tool that computes mean average precision (mAP). Unless stated otherwise, we use the AVA-ActiveSpeaker validation set for our evaluations.

5.2.3.1 Audio-Visual Encoder Evaluation

In this section, we investigate the advantage of the proposed audio and video backbones, compared to backbones used in state-of-the-art active speaker detection systems. The encoder architecture is of utmost importance: the overall performance of the AV-ASD pipeline can only be as good as the extracted features. For these experiments, ISRM and temporal modeling are not used.

Which encoder architectures should be used? Following recent works [153, 154, 167, 168, 169], we take 2D-ResNet-18 architecture as the audio and video backbones of a baseline encoder. Inputs to the video backbone are stacked face crops, and inputs to the audio backbone are MFCCs, corresponding to a length of eight frames. This baseline achieves 79.0 mAP as shown in Table 5.1.

Audio Backbone	Video Backbone	mAP
2D-ResNet-18	2D-ResNet-18	79.0
2D-ResNet-18	3D-ResNet-18	83.9
SincDSNet	2D-ResNet-18	80.8
SincDSNet	3D-ResNet-18	**86.1**

Table 5.1: Performance comparison of different audio and video backbones. Input length of 8-frames is used for all evaluations.

Audio Backbone	Params	MFLOP
SincDSNet	0.15M	13.8
2D-ResNet-18	11.2M	19.2

Table 5.2: Complexity comparison of different audio backbones.

To demonstrate the benefit of applying 3D convolution kernels, we keep the baseline audio backbone and replace 2D-ResNet-18 with 3D-ResNet-18. *This change alone brings an improvement of 4.9 mAP over the baseline.* The improvement is achieved solely due to the ability of the 3D convolution kernels to capture motion patterns in the video data.

Similarly, to evaluate the benefit of SincDSNet as the proposed audio backbone, we keep the baseline video backbone and replace the 'MFCC + 2D-ResNet-18' audio backbone by SincDSNet. *This change brings improvement of 1.8 mAP over the baseline*, thanks to the partially learnable feature extraction by SincDSNet, operating on the raw audio data. Importantly, SincDSNet has 75 times fewer parameters than 2D-ResNet-18 and requires fewer floating point operations (FLOPs), as shown in Table 5.2.

Finally, our audio-visual encoder that uses both 3D-ResNet-18 and SincDSNet as backbones, achieves 7.1 mAP improvement over the baseline.

Can we use resource efficient video encoders? One can attribute the performance boost achieved by 3D-ResNet-18 backbone to its increased number of parameters and FLOPs. Therefore, we have used several resource efficient 3D CNNs [21] as video backbone. We report their performance at the bottom of Table 5.3. Notably, all 3D CNN architectures achieve better performance than 2D-ResNet-18. For instance, although 3D-MobileNetV2 1.0x contains a much smaller number of parameters (approx. 7x fewer) and fewer FLOPs compared to 2D-ResNet-18, it achieves around 4 mAP better performance.

We have also experimented with deeper and computationally more expensive 3D-ResNeXt-101 architecture to check how much performance can be increased. 3D-ResNeXt-101 shows 0.6 mAP improvement over 3D-ResNet-18 when 8-frames input is used.

	Video Backbone	Params	GFLOP	mAP
32-f	3D-ResNeXt-101	48.6M	13.2	**88.9**
	3D-ResNet-18	33.2M	10.3	87.4
16-f	3D-ResNeXt-101	48.6M	14.1	**88.9**
	3D-ResNet-18	33.2M	11.2	87.5
8-f	3D-ResNeXt-101	48.6M	13.2	86.7
	3D-ResNet-18	33.2M	10.3	86.1
	2D-ResNet-18	11.2M	0.9	80.8
	3D-MobileNetV1 2.0x	13.9M	0.6	81.6
	3D-MobileNetV2 1.0x	2.1M	0.7	85.1
	3D-ShuffleNetV1 2.0x	4.6M	0.7	85.0
	3D-ShuffleNetV2 2.0x	3.9M	0.6	84.2

Table 5.3: Comparison of video backbones for different clip lengths. SincDSNet is used at the audio backbone, and face crop resolution is 160×160.

# Speakers	0	1	2	3	4	5
mAP	92.6	93.1	**93.4**	**93.4**	**93.4**	**93.3**

Table 5.4: Performance of inter-speaker relation modelling for different number of background speakers.

How does clip length affect performance? Although we used 8-frames clips to train our audio-visual backbones, longer clips would provide larger temporal context. In Table 5.3, we compare clip lengths of 8-frames, 16-frames and 32-frames for the best performing 3D-ResNeXt-101 and 3D-ResNet-18 video backbones. To maintain similar complexity, we removed the initial temporal downsampling for 8-frames input and inserted an additional temporal downsampling to the initial convolution layer for 32-frames input. Applying 16-frames clip length brings a performance improvement of 1.4 mAP and 2.2 mAP over 8-frames clip length for 3D-ResNet-18 and 3D-ResNeXt-101, respectively. Using 32-frames clip length does not show the same performance improvement over using 16-frames. We suspect that inserting additional temporal downsampling hinders the backbone's ability to capture motion patterns.

5.2.3.2 Inter-Speaker Relation Modeling Evaluation

In this section, we investigate the performance of the proposed ISRM and compare it to an existing approach [154] for context modeling. These experiments include the full ASDNet pipeline (encoder, ISRM, and a temporal model), where the temporal model, if present, is a Bidirectional-GRU with a sequence length of 64.

How many background speakers to use for ISRM? We experimented with different numbers of background speakers for ISRM, and the results are reported in Table 5.4.

Method	Temporal Model	mAP
NonLocal [154]		87.2
NonLocal [154]	✓	92.8
ISRM (ours)		89.0
ISRM (ours)	✓	**93.4**

Table 5.5: Comparison of inter-speakers relation modeling methods.

Background features	mAP
only reference frame	93.4
neighboring window of 9 frames	**93.5**

Table 5.6: Performance comparison when background speakers' features at different number of frames are leveraged.

In general, increasing the number of background speakers features increases the performance. ISRM increases the performance by 0.8 mAP compared to the case where only reference speaker's features are used with temporal modeling (0 background speaker case). In the rest of our experiments, we use three background speakers in the ISRM module.

How does our ISRM compare to existing approaches? In Table 5.5, we provide a comparison of our ISRM approach to the *NonLocal* [208] approach proposed in [154]. *NonLocal* captures relationships between all the speakers within clip, whereas our ISRM approach captures relationships between speakers only within reference frame. When used alone, after the audio-visual backbones, neither *NonLocal* nor our ISRM approach bring significant performance improvement (*NonLocal* even degrades the performance). However, ISRM contributes additional 0.8 mAP compared to a system that uses only temporal modeling.

Can ISRM benefit from neighboring frames? At ISRM, we do not have to use background speakers' features at only reference frames. Neighboring frames relative to the reference frame can also provide useful information for ISRM. Therefore we have used background speakers' features at a neighboring window of 9 frames, which shows a modest 0.1 mAP improvement as reported in Table 5.6. For the rest of the paper, we use 9 neighboring frames at ISRM.

5.2.3.3 Temporal Modeling Evaluation

Which RNN architectures are most suitable? Table 5.7 shows the performance comparison of different RNN blocks used for temporal modeling. All one-directional methods take 32-frames features as input and the last output is used as input to the final fc layer (reference frame is placed to the last of input sequence). For bidirectional

Method	Sequence Length	mAP
Bidirectional-GRU	64	**93.5**
Bidirectional-LSTM	64	93.4
Bidirectional-SRU	64	93.2
GRU	32	92.8
LSTM	32	92.7
SRU	32	92.7

Table 5.7: Performance comparison of temporal modeling methods.

Seq. Length	8	16	32	64	128
mAP	92.0	92.8	93.3	**93.5**	**93.5**

Table 5.8: Performance comparison of using different sequence lengths at the training of Bidirectional-GRU.

methods, we have used 64-frames features as input and center output is used as input to the final fc layer (reference frame is placed at the center of input sequence). Compared to their bidirectional versions, one-directional methods perform around 0.7 mAP worse. Out of all methods, bidirectional-GRU achieves the best performance.

What should be the length of the input sequence? We have experimented with different sequence lengths and reported results in Table 5.8. In general, using a larger sequence length does not hurt the final performance. However, after sequence length 64, the performance converges to 93.5 mAP.

5.2.3.4 Component-wise Analysis

How does each component contribute to the performance? We investigated the contribution of each component to the final performance in Table 5.9. We highlight several findings: **(i)** Without ISRM and temporal modeling, suitable backbones alone achieve 88.9 mAP, which is better than any other state-of-the-art approach; **(ii)** ISRM and temporal modeling improve the performance by 0.7 mAP and 3.7 mAP when they are applied alone, respectively, showing the importance of both stages in the pipeline; **(iii)** In rows 6 and 7 in Table 5.9, we investigated the importance of ISRM stage by evaluating the performance without using reference speakers video features. Accordingly, even without looking at the reference speaker's face, information acquired from background speakers and audio enables our architecture to achieve around 68 mAP. This shows that ISRM is an indispensable part of our pipeline; **(iv)** When ISRM and temporal modeling are applied together, our architecture achieves the best performance with 93.5 mAP.

The contribution of temporal modeling and ISRM stages is visually illustrated in Fig. 5.6. With only audio-visual encoding, each speaker is analyzed independently and predictions for speaking probabilities are made without contextual and long-term

#	Speaker Video Feat.	Audio Feat.	ISRM Feat.	Temporal Modeling	mAP
1	✓				78.8
2		✓			49.3
3	✓	✓			88.9
4	✓	✓		✓	92.6
5	✓	✓	✓		89.6
6		✓	✓		64.5
7		✓	✓	✓	67.8
8	✓	✓	✓	✓	**93.5**

Table 5.9: Contribution of each component to the final performance.

temporal information in Fig. 5.6 (a). After applying temporal modeling and ISRM stages, the ASDNet predictions of speaking probabilities for *not-speaking* speakers drop and *speaking* speaker increases considerable as shown in Fig. 5.6 (b).

How does the clip length affect performance? Increased encoder clip length (16-frames instead of 8-frames using 3D-ResNeXt-101 video backbone) improves the performance by 2.2 mAP if ISRM and temporal modeling are not applied. However, in the complete pipeline, this improvement reflects to a marginal 0.1 mAP improvement in the final performance, which is shown in Table 5.10. This shows that increased encoder clip length shifts the improvement that could have been provided by temporal modeling to the encoder. This might not be desirable if complexity is important in the

(a) (b)

Figure 5.6: The network predictions for speaking probabilities of each speaker (a) after only audio-visual encoding (b) after temporal modeling and ISRM are also applied. Ground truths of *speaking* and *not-speaking* classes are denoted with green and red rectangles, respectively.

Encoder Clip Length	ISRM and Temporal Modeling	mAP
8-frames	✗	86.7
16-frames	✗	88.9
8-frames	✓	93.4
16-frames	✓	93.5

Table 5.10: Effect of encoder clip length on the final performance. SincDSNet and 3D-ResNeXt-101 are used for audio and video backbones, respectively.

design of the architecture since doubling encoder clip length means doubling the complexity.

Can ISRM be placed after temporal modeling? If necessary, the order of ISRM and temporal modeling can be changed, which results in only a 0.1 mAP performance degradation.

Can we make the full pipeline causal? The complete pipeline can be made causal by placing the reference frame to the last place of the input for the encoder and temporal modeling stages; and by not using neighboring frames background speakers' features at ISRM. So that, no future information is used for the active speaker detection of the current frame. The causal pipeline achieves 90.6 mAP, which is still better than any state-of-the-art approach.

5.2.3.5 Comparison with the State-of-the-art

How does ASDNet compare to state-of-the-art methods? We compare the performance of ASDNet with several state-of-the-art methods in Table 5.11. For the final ASDNet, we used 16-frames clips at the audio-visual encoding stage, 3 background speakers with 9 neighboring window at the ISRM stage, and bidirectional-GRU with 64-frames sequence length at the temporal modeling stage. ASDNet outperforms the second best approach by 4.7 mAP on the validation set, and by 3.9 mAP on the test set of the AVA-ActiveSpeaker dataset.

How does the number of faces affect the performance? Increased number of faces makes the active speaker detection task more challenging and the performance of ISRM becomes more critical. ASDNet outperforms all other state-of-the-art methods for all different face numbers as shown in Table 5.12. The superiority of ASDNet becomes more significant as the number of faces increases.

How does face size affect the performance? Performance comparison for face size, which is set as small for [0, 64), medium for [64, 128), and large for [128, ∞) pixels, is shown in Table 5.13. ASDNet outperforms all other state-of-the-art methods for all different face sizes. The superiority of ASDNet becomes more significant for smaller faces.

	Method	mAP
validation set	ASDNet (ours)	**93.5**
	Causal ASDNet (ours)	90.6
	MAAS-TAN [153]	88.8
	Chung et al. [168]	87.8
	ASC [154]	87.1
	Zhang et al. [169]	84.0
	Sharma et al. [280]	82.0
	Roth et al. [167]	79.2
test set	ASDNet (ours)	**91.7**
	Chung et al. [168]	87.8
	ASC [154]	86.7
	Zhang et al. [169]	83.5
	Roth et al. [167]	82.1

Table 5.11: Comparison with state-of-the-art methods on the AVA-ActiveSpeaker dataset. mAP results are calculated with the official evaluation tool as explained in [167].

Method	Number of Faces		
	1	**2**	**3**
ASDNet (Ours)	**95.7**	**92.4**	**83.7**
MAAS [153]	93.3	85.8	68.2
ASC [154]	91.8	83.8	67.6
Baseline [167]	87.9	71.6	54.4

Table 5.12: Performance comparison by number of visible faces on each frame.

Method	Face Size		
	S	**M**	**L**
ASDNet (Ours)	**74.3**	**89.8**	**96.3**
MAAS [153]	55.2	79.4	93.0
ASC [154]	56.2	79.0	92.2
Baseline [167]	44.9	68.3	86.4

Table 5.13: Performance comparison by face size.

5.2.4 Summary

In this section, we scrutinized the task of Audio-Visual Active Speaker Detection and proposed a three-stage architecture, called ASDNet. With the proposed audio-visual

encoder and the inter-speaker relation modeling mechanism, ASDNet outperforms the previous state-of-the-art with a significant 4.7 mAP and 3.9 mAP on the validation and test set of the AVA-ActiveSpeaker dataset, respectively. To make the final design and hyperparameter choices for ASDNet, we followed insights from carefully designed experiments each targeted a specific aspect of the system. Each of these experiments was discussed in the paper. We believe that these insights can be useful for other complex audio-visual tasks as well that require context and temporal modeling.

Chapter 6

Conclusion and Outlook

This thesis addresses the automatic recognition of human activities from video data. Considering the amount of generated video data traversing the internet, the addressed task is extremely valuable for applications such as surveillance, autonomous driving, and entertainment. Following the evolution of human activity recognition research, we grouped our contributions under three chapters: (Chapter 3) video analysis with frame-level features, (Chapter 4) video analysis with clip-level features, and (Chapter 5) audio-visual video analysis. The presented contributions in this thesis can be listed as follows: (i) (Section 3.2) we compared different spatiotemporal modeling techniques operating on frame-level features extracted by 2D CNNs, (ii) (Section 3.3) we propose a data level fusion strategy in order to incorporate motion information to frame-level features, (iii) (Section 4.2) we present a unified CNN architecture benefiting clip-level features extracted by 3D CNNs for real-time spatiotemporal action localization, (iv) (Section 4.3) we present resource efficient 3D CNN architectures, (v) (Section 4.4) we present a two-level hierarchical architecture to address the challenges of online recognition of dynamic hand gestures, (vi) (Section 4.5) we present a new 3D CNN architecture for a reduced computational complexity at online video streaming applications, (vii) (Section 5.2) we present a three-stage architecture for audio-visual active speaker detection task, (viii) (Appendix A, B and C) we present 3 new datasets on gesture and action recognition and make them publicly available for the scientific community.

Next, we summarize the aforementioned contributions in this thesis and finally provide future work.

6.1 Summary

Chapter 3 presents video analysis with frame-level features and presents the following contributions:

- In Section 3.2, we analyze various spatiotemporal modeling techniques employed on top of frame-level features extracted by 2D CNNs and compare them in terms of efficiency and accuracy on action and gesture recognition tasks. All analyzed techniques are trained end-to-end together with the feature extraction part. Although each analyzed technique has been used in various works individually, there has not been any detailed comparative analysis yet, and this section aims to fill this gap.

- In Section 3.3, we address the incapacity of frame-level feature based approaches to capture motion information within consecutive frames. Frame-level feature based approaches mostly operate on sparsely sampled frames on the incoming video stream to prevent redundant computations and are incapable of capturing motion information within consecutive frames. Motivated with this, Section 3.3 proposes a data-level fusion strategy, Motion Fused Frames, for optical flow and RGB modalities and demonstrates its effectiveness on the gesture recognition task. Although the optical flow modality contains the necessary motion information within consecutive frames, its computation complexity does not make it suitable for resource efficient architectures.

Chapter 4 presents video analysis with clip-level features and presents the following contributions:

- In Section 4.2, a unified CNN architecture, You Only Watch Once (YOWO), is presented jointly benefiting from clip-level features extracted by 3D CNNs and frame-level features extracted by 2D CNN for the task of real-time spatiotemporal action localization. Given a video clip, presented YOWO architecture predicts the bounding boxes on the key-frame and respective action probabilities in a single evaluation, which gives it the real-time operation capability. We evaluate YOWO on three action localization dataset to show the effectiveness of the proposed architecture.

- In Section 4.3, we address the drawback of 3D CNNs having significantly more parameters and requiring seriously more computation at inference time compared to 2D CNNs. Correspondingly, we present families of resource efficient 3D CNN architectures and evaluate them on 3 different video tasks. Presented architectures are inflated versions of popular resource efficient 2D CNN architectures. Our results show that these architectures can be used for different types of real-world applications having limited memory and power budget.

- In Section 4.4, a two-level hierarchical architecture is proposed for HCI systems to reduce resource waste when the system is not being used. The proposed architecture consists of a lightweight detector and a heavyweight classifier. The detector always remains active and activates the classifier when it detects a hand gesture for its classification. We also address the challenges of online recognition of dynamic hand gestures such as early recognition and single-time activation. Moreover, we present a new metric, Levenshtein accuracy, to evaluate the performance of architectures at online gesture recognition.

- In Section 4.5, we address the challenges of operating 3D CNNs online by proposing a new 3D CNN architecture, Dissected 3D CNNs, for a reduced computational complexity at online video streaming applications. Conventional 3D CNNs work with fixed-size of input and are mostly designed for offline applications. Therefore, online operating frameworks generally use conventional

3D CNNs with a sliding window approach resulting in redundant computations if a small stride is used. On the other hand, Dissected 3D CNNs propagate the intermediate volumes of the network in a cache for future calculations, which prevents these redundant calculations. Accordingly, when a new frame becomes available in the video stream, only this frame is processed, which substantially reduces the number of computations at online operation.

Chapter 5 presents audio-visual video analysis with the following contribution:

- In Section 5.2, we present several practical guidelines for the task of audio-visual active speaker detection, which results in the ASDNet architecture consisting of three stages: (i) audio-visual feature extraction, (ii) inter-speaker relation modeling, and (iii) temporal modeling. Each stage of the ASDNet architecture is tuned with a detailed ablation study and plays an important role in the final performance.

We also make three dataset contributions for the gesture and action recognition tasks:

- In Appendix A, we present Scaled Hand Gesture Dataset (SHGD). SHGD addresses the scalability of hand gestures by constituting each gesture with several preset gesture-phonemes. This way, new hand gestures can be generated using different combinations of gesture-phonemes. We also present a CNN-based framework to recognize hand gestures by learning only their constituents of gesture-phonemes.
- In Appendix B, we present Driver Micro Hand Gesture (DriverMHG) dataset. DriverMHG dataset contains micro hand gestures, which occur within very short time intervals at spatially constrained areas. These gestures contain mostly movement of thumb in order not to distract the drivers from the road. Online recognition of these micro gestures is specifically challenging. Therefore, we also propose a lightweight CNN architecture that operates online efficiently with a sliding window approach.
- In Appendix C, we present Driver Anomaly Detection (DAD) dataset. There are unbounded many distracting actions that a driver can do while driving. Accordingly, instead of recognizing a set of distracting actions that are commonly defined by dataset providers, we approach the problem as an open-set recognition problem. With this motivation, we introduce the DAD dataset that contains normal driving videos together with a set of anomalous actions in its training set and unseen anomalous actions in its test set. Moreover, we propose a contrastive learning approach to learn a metric to differentiate normal driving from anomalous driving.

6.2 Future Work

Abundance of video data in the wild. There is no scarcity of video data. Video data is all around us. However, only a very small proportion of it is annotated to

be used for supervised learning. Accordingly, unsupervised and semi-supervised video understanding is a very promising research direction.

Synthetic Data. The success of deep learning approaches on computer vision tasks comes with the cost of acquiring large enough annotated data. The success of 3D CNNs on video understanding tasks has also been possible only with the availability of large-scale video datasets. Although synthetic datasets are actively been used for various computer vision tasks such as body analysis [281], multi-object tracking [282], semantic segmentation [283] and 3D object detection [284], there has not been enough research effort for video activity recognition. Considering the fact that activity recognition is a high level task and requires capturing motion patterns and person-object, person-person interactions, it is challenging to synthetically create videos containing such complex activities. Although there have been some recent synthetic data generation methodologies for action recognition such as SURREACT [285] and ElderSIM [286], the field is open to contributions that would relax the dependence on annotated real-world video data.

New approaches at video activity recognition. Very recently, mostly over the last year, there has been a boom of Transformer [111] based architectures on computer vision tasks. Recent Transformer based approaches surpass the 3D CNN based approaches achieving new state-of-the-art results [119, 120, 135]. The strength of Transformer based approaches comes from the applied attention mechanism, which differentially weights the significance of each part of the sequential input data. In Section 3.2, we also presented a Transformer based approach as spatiotemporal modeling of frame-level features, which achieved superior results compared to other approaches on action recognition task. However, we believe that capability of 3D CNNs to capture motion patterns in video data is still very valuable. Joint usage of Transformers and 3D CNNs is an interesting next step for video understanding tasks.

Online recognition of actions and gestures. Online recognition of action and gestures from video streams is an understudied topic. The majority of action and gesture recognition architectures are designed for offline applications. Most of the time, these architectures require a separate mechanism to be used in online applications. Moreover, achieved online performance is usually worse than offline performance. For instance, approach proposed in Appendix B achieves 74.00% online (Levenshtein) accuracy compared to 91.56% offline accuracy. Research on online action and gesture recognition is valuable especially for industrial applications and is still open to improvements.

Multimodal approaches. Using multiple modalities at video understanding tasks results in improved performance. Availability of cameras capturing multi-modal data such as RGB, depth, infrared, thermal data synchronously made fusion of these modalities even more popular. What is often overlooked is the audio modality that is

already available at video recordings. Although some recent works use audio data for visual tasks [261, 262, 263], the usage of audio modality and its fusion to visual modality is still an understudied topic.

Appendix A

Scaled Hand Gesture Dataset

The use of hand gestures provides a natural alternative to cumbersome interface devices for Human-Computer Interaction (HCI) systems. As technology advances and communication between humans and machines becomes more complex, HCI systems should also be scaled accordingly in order to accommodate the introduced complexities. In this work, we propose a methodology to scale hand gestures by forming them with predefined gesture-phonemes, and a convolutional neural network (CNN) based framework to recognize hand gestures by learning only their constituents of gesture-phonemes. The total number of possible hand gestures can be increased exponentially by increasing the number of used gesture-phonemes. For this objective, we introduce a new benchmark dataset named Scaled Hand Gestures Dataset (SHGD) with only gesture-phonemes in its training set and 3-tuples gestures in the test set. In our experimental analysis, we achieve to recognize hand gestures containing one and three gesture-phonemes with an accuracy of 98.47% (in 15 classes) and 94.69% (in 810 classes), respectively. Our dataset, code and pretrained models are publicly available at https://www.mmk.ei.tum.de/shgd/.

This section is based on our publication *Talking With Your Hands: Scaling Hand Gestures and Recognition With CNNs* [25].

A.1 introduction

As technology keeps advancing, the use of computers in our lives increases as well with additional new devices such as smartphones, watches, TVs, headphones, autonomous cars, etc. Therefore, the communication between humans and machines gradually becomes more complex, requiring HCI systems to accommodate the introduced complexities. In this work, we propose an approach to scale hand gestures by composing each gesture with multiple gesture-phonemes. So, our motivation is first to learn the gesture-phonemes successfully, then to recognize hand gestures, which contain multiple gesture-phonemes, with only this knowledge.

Structuring hand gestures with this approach enables to scale hand gestures without requiring to collect additional training data. For a given number of gesture-phonemes, the number of all possible hand gestures is exponentially proportional to the number of gesture-phonemes each gesture contains. For the proposed gesture scaling approach, we present a convolutional neural network (CNN) based framework using sliding window approach together with Viterbi-like [287] decoder algorithm. For the CNN model, we

have used 2-dimensional (2D) and 3-dimensional (3D) SqueezeNet and MobileNetV2 models.

This work presents the following contributions:

(i) Our major contribution is creating a hand gesture recognition framework, which is "scalable" according to the complexity of the desired HCI system. To the best of our knowledge, this is the first work that addresses the scalability of hand gestures. The CNN model is only trained with 10 gesture phonemes and 3 signaling classes (*preparation*, *retraction* and *no-gesture*), and the framework can recognize scaled gesture tuples with 3 gesture phonemes (as in this work) or more. Assumed that a HCI system with the recognition capability of 810 different gestures needs to be implemented. With the old fashioned way, you need to define 810 different hand gestures, collect enough training samples (400 training samples for each class), train an architecture to get the desired accuracy (remember that for ChaLearn IsoGD [170], the state-of-the-art accuracy is around 80% for 249 classes). With this framework, you just need to train with 10 gesture phonemes and 3 signaling classes, then for 810 classes (3-tuple gestures), you can achieve around 95% accuracy.

(ii) The second contribution is the benchmark dataset named Scaled Hand Gestures Dataset (SHGD), which will be made publicly available. The videos are collected using a Time-of-Flight (ToF) based 3D Image Sensor, which is shown in Fig. A.1. The dataset contains only gesture-phonemes in its training set. For the test set, SHGD contains gesture-phonemes and 3-tuple gestures.

(iii) The third contribution of this work is that with the designed Viterbi-like decoder, the performed 3-tuple gestures are recognized only once. This contains utmost importance for online HCI systems. Moreover, the designed Viterbi-like decoder is very lightweight as HCI systems should be designed considering the memory and power budget of the HCI system.

A.2 Scaled Hand Gestures Dataset (SHGD)

SHGD contains 15 single hand gestures, each recorded for infrared (IR) and depth modalities using Infineon® IRS1125C REAL3™ 3D Image Sensor. Each recording contains 15 gesture samples (one sample per class). There are in total 324 recordings from 27 distinct subjects in the dataset. Recordings of 8 subjects are reserved for testing, which makes 30% of the dataset. Every subject makes 12 video recordings using two hands under 6 different environments, which are designed for increasing the network robustness against different lighting conditions and background disturbances. These environments are (1) indoors under normal daylight, (2) indoors under daylight and with an extra person in the background, (3) indoors at night under artificial lighting, (4) indoors in total darkness, (5) outdoors under intense sunlight and (6) outdoors under normal sunlight. We have simulated outdoor environments using two bright lights: Two lights for "intense sunlight" and one light for "normal sunlight".

Label	Gesture	Label	Gesture	Label	Gesture
1	Fist	6	Two Fingers	11	Swipe Left*
2	Flat Hand	7	Five Fingers	12	Swipe Right*
3	Thumb Up	8	Stop Sign	13	Pull Hand In*
4	Thumb Left	9	Check	14	Move Hand Up*
5	Thumb Right	10	Zero	15	Move Hand Down*

Table A.1: 15 single gesture classes in Scaled Hand Gesture Dataset (SHGD). * marks the dynamic gestures which are not included as gesture-phonemes.

Fig. A.1 shows data collection setup, used camera and data samples. Subjects performed gestures while observing the computer screen, where the gestures were prompted in random order. Videos are recorded at 45 frames per second (fps) with a spatial resolution of 352×287 pixels. Each recording lasts around 33 seconds.

A.2.1 Single Gestures

In its training set, SHGD contains only single gestures under 15 classes, which are given in Table A.1. Recordings in the dataset are continuous video streams meaning that each recording contains *no-gesture* and *gesture* parts. Moreover, each *gesture* contains *preparation*, *nucleus* and *retraction* phases [288, 289, 133], which are critical for real-time gesture recognition.

Among the single gesture classes listed in Table A.1, static gestures are selected as gesture-phonemes since it is more convenient to perform different static gestures sequentially. For the rest of this work, we will use the term *phoneme* instead of *gesture-phoneme* for the sake of easiness.

A.2.2 Gesture Tuples

Gesture tuple refers to hand gestures that contain sequentially performed phonemes. There are in total 10 different phonemes. When constructing gesture tuples, we leave out the consecutive same phonemes to avoid sequence length confusion. Therefore, the total number of different tuples can be calculated by the following equation:

$$N = m(m-1)^{(s-1)}, \tag{A.1}$$

where m is the number of different phonemes and s is the number of phonemes that the gesture tuple contains.

Besides the test set for single gestures, SHGD also has a test set for gesture tuples containing 3 phonemes. 5 subjects perform gesture tuples under 5 different lighting conditions (excluding the environment of (2)). There are in total $10 \times (10-1)^{(3-1)} = 810$ permutations meaning different classes for 3-tuple gestures. Recordings are not segmented for this case. Therefore, one recording contains *no-gesture*, *3-tuple gesture* and *no-gesture* without exact location of *3-tuple gesture*.

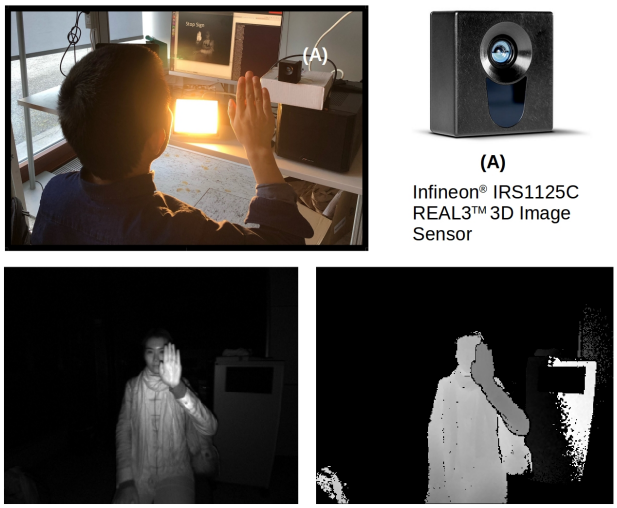


Figure A.1: Data collection setup. Dataset is collected in infrared (bottom-left) and depth (bottom-right) modalities using Infineon® IRS1125C REAL3^{TM} 3D Image Sensor.

Since gestures are performed at different speeds in real-life scenarios, we have also collected 3-tuple gestures at three different speeds: slow, medium and fast. The subjects should finish 3-tuple gestures within 300 frames (6.7 sec), 240 frames (5.3 sec) and 180 frames (4 sec) for slow, medium and fast speed, respectively.

A.2.3 SHGD-15 and SHGD-13

SHGD-15 refers to the standard dataset where all single gestures in Table A.1 are included. On the other hand, SHGD-13 is specifically designed for 3-tuple gesture recognition. Besides 10 phonemes, SHGD-13 also contains *preparation* (raising hand), *retraction* (lowering hand) and *no-gesture* classes. As there is no indication when a gesture starts and ends in the video, we use *preparation* and *retraction* classes to detect Start-of-Gesture (SoG) and End-of-Gesture (EoG). We use *no-gesture* class to reduce the number false alarms since most of the time, "no gesture" is performed in real-time gesture recognition applications [22].

SHGD-15 is a balanced dataset with 96 samples in each class. However, SHGD-13 is an imbalanced dataset, where *preparation* and *retraction* classes contain 10 times more samples than phonemes, whereas *no-gesture* contains around 20 times more samples than phonemes. Therefore, training of SHGD-13 requires special attention.

A.3 Methodology

In this section, we first explain the details of the experimented framework and the applied Viterbi-like decoder.

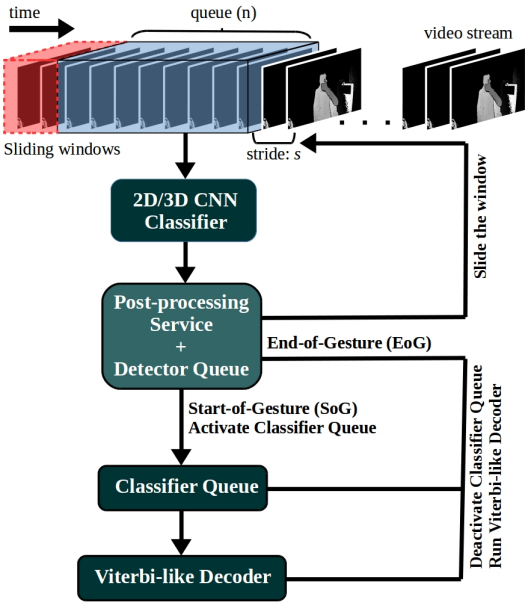


Figure A.2: The general workflow of the proposed architecture. Sliding windows with stride s run through incoming video frames, and these frames in the queue are fed to a 2D or 3D CNN based classifier. The classifier's results are post-processed afterwards. After Start-of-Gesture (SoG) gets detected, the classifier queue is activated. Classifier's results are saved in the classifier queue until End-of-Gesture (EoG) is detected. Then, the Viterbi-like decoder runs on the classifier's queue to recognize the 3-tuple gesture.

A.3.1 Network Architecture

The general workflow of the proposed architecture is depicted in Fig. A.2. A sliding window goes through the video stream with a queue size of 8 frames and stride s of 1. The frames in the input queue are passed to a 2D/3D CNN which is pretrained on SHGD-13. In our experiments, we have used 2D and 3D versions SqueezeNet [87] and MobileNetV2 [177] as classifiers in our architecture. The classification results are then post-processed by averaging with a non-overlapping window size of 5. In this way, we can filter out some fluctuations due to ambiguous states while changing the phonemes. Next, the post-processed outputs are fed into a detector queue, which tries to detect SoG and EoG. When the sum of class scores for *preparation* is higher than the threshold, we set SoG flag on, activate the classifier queue, and start storing the post-processed scores. Then, the detector queue is responsible for detecting the EoG in a similar manner. After the EoG flag is received, we deactivate the classifier queue and run the Viterbi-like decoder which recognizes the 3-tuple gesture. In the next parts, we explain the details of the main building blocks in the proposed architecture.

A.3.2 Viterbi-like Decoder

Viterbi decoding was invented by Andrew Viterbi [287] and is now widely used in decoding convolutional codes. It is an elegant and efficient way to find out the optimal path with minimal error. In this work, we have adapted it and used a Viterbi-like decoder to find out the phoneme sequences in 3-tuple gestures with maximal probability. Similar to the conventional Viterbi algorithm, we narrow down the optional paths systematically for each new input in the classifier queue.

For the Viterbi-like decoder, we introduced a couple of terms for better comprehensibility: K is the number of allowed state transitions in the output sequence, which is 2 as we use 3-tuple gestures. The state refers to a phoneme in a path for the given time instant. P refers to class-conditional probability scores for phonemes stored in Classifier Queue, which is shown in (2), whose columns P_t are the average probability scores of each phoneme for five consecutive time instants. P_t values are softmaxed before putting in P. T is the length of P (i.e. number of columns), and N is the number of phoneme classes, which is 10 in our case. Therefore, the size of P is $T \times N$.

$$P = \begin{bmatrix} | & \cdots & | & \cdots & | \\ P_0 & \cdots & P_t & \cdots & P_{T-1} \\ | & \cdots & | & \cdots & | \end{bmatrix}, \; P_t = \begin{bmatrix} p_{t,0} \\ p_{t,1} \\ \vdots \\ p_{t,N-1} \end{bmatrix} \tag{A.2}$$

The probability of a path is the sum of the probability scores of all the states that this path goes through. Besides the number of allowed transitions K, we introduce another constraint, transition cost δ, in order to prevent false state transitions in the path. A path metric M holds the paths $m_{t,i}$ with their sequence record $\pi_{t,i}$, path score $s_{t,i}$ and the transition times $k_{t,i}$. The path $m_{t,i}$ is shown as follows:

$$m_{t,i} = [\pi_{t,i}, s_{t,i}, k_{t,i}], \quad 0 \leq i < \gamma, \;\; 0 \leq t < T. \tag{A.3}$$

The state of path $m_{t,i}$ at time instant t is denoted as $n_{t,i}$, and the last state in $\pi_{t,i}$ is also denoted as $\pi_{t,i}^{last}$. The transition cost is set to -0.2. The path scores s, transition record k and sequence record π are updated with every new P_t as following:

$$s_{t+1,i} = s_{t,i} + p_{t+1,i} + \delta, \;\; \delta = \begin{cases} -0.2, & \text{if } n_{t+1,i} \neq \pi_{t,i}^{last} \text{ and } k_{t,i} < K, \\ 0, & \text{otherwise.} \end{cases} \tag{A.4}$$

$$\pi_{t+1,i} = \begin{cases} \pi_{t,i} \cup n_{t+1,i}, & \text{if } n_{t+1,i} \neq \pi_{t,i}^{last} \text{ and } k_{t,i} < K, \\ \pi_{t,i}, & \text{otherwise.} \end{cases} \tag{A.5}$$

$$k_{t+1,i} = \begin{cases} k_{t,i} + 1, & \text{if } n_{t+1,i} \neq \pi_{t,i}^{last} \text{ and } k_{t,i} < K, \\ k_{t,i}, & \text{otherwise.} \end{cases} \tag{A.6}$$

In order to reduce computation, we limit the number of paths in M to γ, which is set to 300. The working mechanism of the proposed Viterbi-like decoder is given in

Algorithm 2 Viterbi-like decoder for 3-tuple gesture recognition

function VITERBI-LIKE DECODER(P, S)
  Initialize s, π and k at P_0;
  for each P_t **do**
    Create all possible paths
    Update s, π and k according to (4), (5) and (6)
    Descending sort all m in M with their scores s
    Keep no more than the first γ paths
  return π of m with maximum s and k=K

algorithm 2. Fig. A.3 depicts the illustration of our Viterbi-like decoder. Our decoder can inherently deal with the ambiguities at phoneme transitions as it naturally makes use of temporal ensembling.

A.4 Experiments

A.4.1 Results using SHGD-15 and SHGD-13

The performance of our models for SHGD-15 and SHGD-13 using different modalities are given in Table A.2. The best results are achieved by 2D-SqueezeNet (98.47%) and 3D-MobileNetV2 (96.06%) for SHGD-15 and SHGD-13, respectively, both at IR+D modality.

For SHGD-15, 2D CNNs always achieve better results than 3D CNNs for all modalities. This is because around 66.67% of samples in SHGD-15 are static gestures, and 2D CNNs captures static content better than 3D CNNs. On the other hand, around 20% of samples in SHGD-13 are static gestures resulting 3D CNNs to perform better.

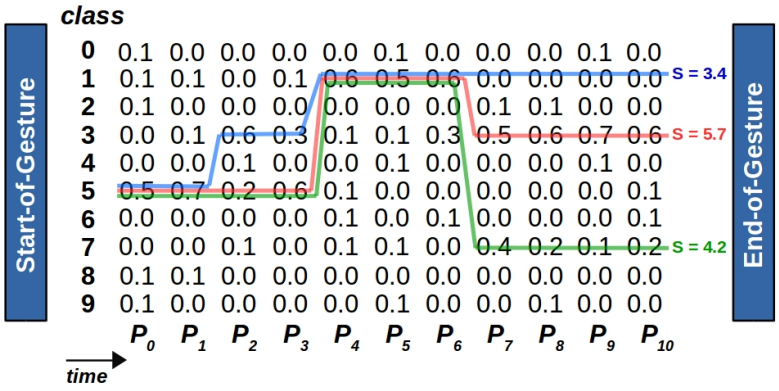


Figure A.3: Illustration of our Viterbi-like decoder for 3-tuple gesture recognition. For the sake of simplicity, we have highlighted only three paths while the correct one is in red. For the correct path, π = [5,1,3], s = 6.1 and k = 2. 2 times the transition cost of 0.2 is subtracted from each path.

Model		Accuracy (%)	
		SHGD-15	**SHGD-13**
IR	2D-SqueezeNet	98.13	92.56
	2D-MobileNetV2	97.36	93.11
	3D-SqueezeNet	92.99	95.87
	3D-MobileNetV2	92.85	94.62
Depth	2D-SqueezeNet	98.13	95.02
	2D-MobileNetV2	98.13	95.64
	3D-SqueezeNet	89.93	95.87
	3D-MobileNetV2	92.78	95.85
IR+D	2D-SqueezeNet	98.47	93.94
	2D-MobileNetV2	97.92	95.06
	3D-SqueezeNet	92.64	95.59
	3D-MobileNetV2	94.31	96.06

Table A.2: Results of different models with different modalities on the test sets of SHGD-15 and SHGD-13.

Different models are sensitive to different data modalities. For instance, 2D-MobileNetV2 performs better at depth modality, whereas 3D-MobileNetV2 performs best at IR+D modality. However, the fusion of different modalities (IR+D) results in better performance most of the time.

A.4.2 Results for 3-tuple gesture recognition

In this section, we evaluate the performance of our models for 3-tuple gesture recognition. The test set for this objective contains 1620 samples from 810 different permutations (i.e. classes). In order to evaluate the performance, three different errors and the total accuracy are defined as follows:

- Detector error: The number of the gesture tuples, in which SoG or EoG is not successfully detected. It includes the flags detected at the wrong time and flags not detected at all.
- Tuple error: The number of the gesture tuples, whose predicted sequence does not match the ground truth.
- Single error: The number of the single phonemes which are recognized mistakenly inside the tuple error. For instance, if the ground truth is [6,8,10] and the recognized tuple is [6,10,12], then the single error is 2.
- Total accuracy: The percentage of the correctly predicted tuples in the whole test set, where $N_{samples}$ is equal to 1620. It is calculated as follows:

	Model	Error			Acc.(%)
		Det	Tup	Sin	
IR	2D-SqueezeNet	191	54	126	84.88
	2D-MobileNetV2	116	103	248	86.60
	3D-SqueezeNet	11	159	375	89.51
	3D-MobileNetV2	10	209	492	86.48
Depth	2D-SqueezeNet	73	127	275	87.65
	2D-MobileNetV2	77	111	259	88.40
	3D-SqueezeNet	68	200	261	83.46
	3D-MobileNetV2	82	169	271	84.51
IR+D	2D-SqueezeNet	125	79	184	87.41
	2D-MobileNetV2	41	71	165	93.09
	3D-SqueezeNet	7	103	228	93.21
	3D-MobileNetV2	3	83	171	94.69

Table A.3: Performance for the tuple detection. Det, Tup and Sin refer to the number of detector, tuple and single phoneme errors out of 1620 test samples.

$$Acc{=}(1 - \frac{Err_{det} + Err_{tup}}{N_{samples}})\% \tag{A.7}$$

For this task, models are trained with SHGD-13. Table A.3 shows the performance of experimented models on different modalities for 3-tuple gesture recognition. For the detection threshold of the detector, 5 and 6 are used for 2D and 3D CNNs, respectively. Similar to previous results, 3D CNNs capture dynamic classes better and make fewer detector errors. On the other hand, 2D CNNs make fewer tuple and single errors as they consist of static classes.

3D-MobileNetV2 achieves the best performance with an accuracy of 94.69% for recognizing 810 different gesture tuples. 3D CNNs surpass 2D CNNs in this task generally, except for depth modality. We assume that this is due to the noise pixels appearing in the depth modality from time to time. Therefore, 3D CNNs fail to capture the temporal relations between noisy frames.

Appendix B

Driver Micro Hand Gesture Dataset

The use of hand gestures provides a natural alternative to cumbersome interface devices for Human-Computer Interaction (HCI) systems. However, real-time recognition of dynamic micro hand gestures from video streams is challenging for in-vehicle scenarios since (i) the gestures should be performed naturally without distracting the driver, (ii) micro hand gestures occur within very short time intervals at spatially constrained areas, (iii) the performed gesture should be recognized only once, and (iv) the entire architecture should be designed lightweight as it will be deployed to an embedded system. In this work, we propose an HCI system for dynamic recognition of driver micro hand gestures, which can have a crucial impact in the automotive sector, especially for safety-related issues. For this purpose, we initially collected a dataset named Driver Micro Hand Gestures (DriverMHG), which consists of RGB, depth and infrared modalities. The challenges for dynamic recognition of micro hand gestures have been addressed by proposing a lightweight convolutional neural network (CNN) based architecture which operates online efficiently with a sliding window approach. For the CNN model, several 3-dimensional resource efficient networks are applied and their performances are analyzed. Online recognition of gestures has been performed with 3D-MobileNetV2, which provided the best offline accuracy among the applied networks with similar computational complexities. The final architecture is deployed on a driver simulator operating in real-time. We make the DriverMHG dataset and our source code publicly available https://www.mmk.ei.tum.de/DriverMHG/.

This section is based on our publication *DriverMHG: A Multi-Modal Dataset for Dynamic Recognition of Driver Micro Hand Gestures and a Real-Time Recognition Framework* [26].

B.1 Introduction

Computers have become an indispensable part of human life. Thus, facilitating natural human-computer interaction (HCI) contains utmost importance to bridge the human-computer barrier. Although there is a growing interest in the development of new approaches and technologies for HCI, gestures have long been deemed to be a more natural, creative and intuitive interaction technique for communicating with our computers.

In this work, we create an HCI system, which is based on dynamic recognition of driver's micro hand gestures. In the automotive sector, this kind of system can have a crucial impact, especially on safety-related issues. While driving, performing a hand gesture, which represents one action, by keeping the hands on the wheel is much safer than pressing a button to activate that action, which causes eyes off the road for a few seconds. For this objective, the following challenges should be taken into account:

1. A suitable dataset must be collected. The gestures in the dataset should be natural and should not distract the driver while performing.
2. The created architecture should distinguish the other hand movements when the driver is not performing any gesture.
3. The architecture should be able to capture micro gestures, which are occurring within very short time intervals at spatially constrained areas, with acceptable accuracy.
4. The entire architecture should be designed considering the memory and power budget.

Considering the aforementioned challenges, we initially collected a multi-modal micro hand gesture dataset using a driver simulator with 25 participants, who performed predefined micro gestures. This dataset is collected with one sensor providing synchronized RGB, infrared and depth modalities. To the best of our knowledge, this is the first multi-modal dataset that consists of micro hand gestures performed on a steering wheel.

Today, several video cameras provide more than one modality, and widely used ones are cameras providing RGB, infrared and depth modalities. Each modality has advantages, e.g. infrared is invariant to illumination and depth modality provides depth information. In this work, in addition to the mono-modal analysis with RGB, infrared and depth modalities, we have also analyzed the impact of fusion on the recognition performance of micro hand gestures.

In real-world applications, resource efficiency, fast reaction time and single time activation are as crucial as reaching an acceptable accuracy for the created HCI system. In this work, we have applied several resource efficient 3-dimensional (3D) CNN architectures proposed in [21] as the CNN model of our dynamic micro hand gesture recognition approach. Among these architectures, 3D-MobileNetV2 has provided the best offline accuracy compared to the architectures with similar computational complexities. Therefore, in this work, online recognition analysis has been performed with 3D-MobileNetV2 as the CNN model. For online recognition, we have also proposed a novel single time activation approach, which does not require a separate *gesture detector* block as in [22].

In the proposed architecture, the video stream is split into two branches each containing only one hand. Then CNN models are trained separately on each hand and deployed to the online recognition architecture. For evaluating online recognition, we have used a recently proposed Levenshtein accuracy [22]. The experiments show that

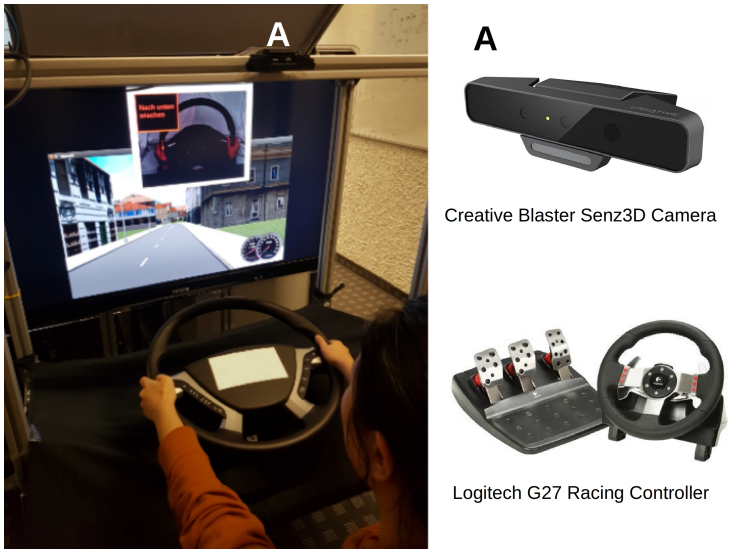


Figure B.1: Driving Simulator - Setup. Left: The Complete Driving Simulator Setup showing a subject performing the driving task. Upper right: a picture of the mounted Creative Blaster Senz3D Camera. Lower Right: The Logitech G27 Racing Controller.

3D-MobileNetV2 can operate online efficiently with a sliding window approach for dynamic micro hand gesture recognition. However, achieved 74.00% online (Levenshtein) accuracy compared to 91.56% offline accuracy shows that online recognition of micro gestures is challenging and open to improvements.

B.2 DriverMHG Dataset

There are a lot of vision-based datasets publicly available, but for the specific task of classifying driver micro hand gestures on a steering wheel, there is none. For this purpose, we recorded the Driver Micro Hand Gesture (DriverMHG) dataset, which fulfills the following criteria:

- Large enough to train a Deep Neural Network

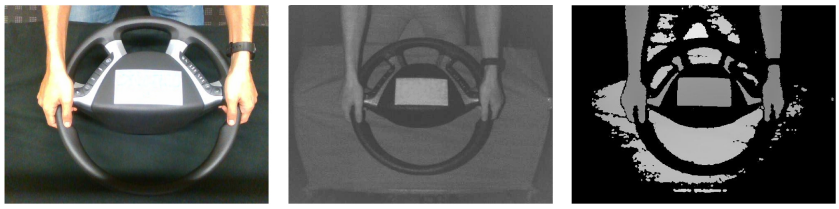
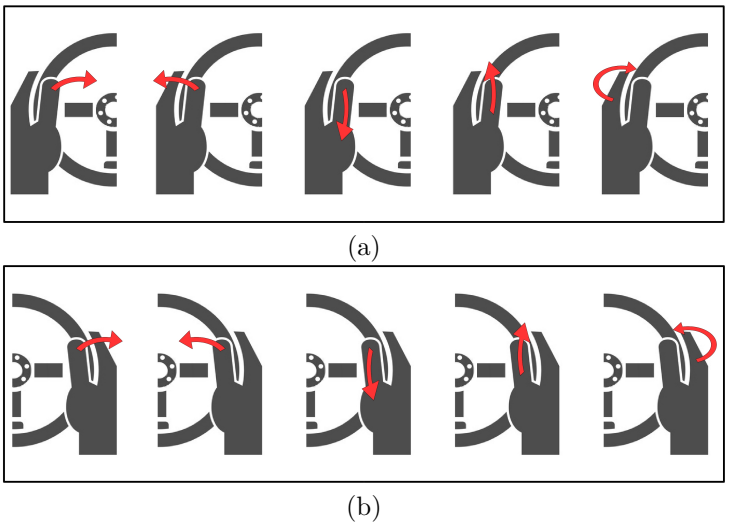


Figure B.2: The dataset is collected for 3 different modalities: (a) RGB, (b) infrared, (c) depth.



(a)

(b)

Figure B.3: Illustration of selected 5 micro gestures for (a) left and (b) right hands. From left to right: Swipe Right, Swipe Left, Flick Down, Flick Up and Tap. Besides these 5 gestures, *none* and *other* gesture classes are also collected for the DriverMHG dataset.

- Contains the desired labeled gestures
- The distribution of labeled gestures is balanced
- Has '*none*' and '*other*' action classes to enable continuous classification
- Has the ability to allow benchmarking

In order to record this dataset, a driving simulator has been set up as shown in Fig. B.1. The dataset was recorded with the help of 25 volunteers (13 males and 12 females) using this simulator, which consists of a monitor, a Creative Blaster Senz3D camera featuring Intel RealSense SR300 technology, a Logitech G27 racing controller, whose wheel is replaced with a truck steering wheel and the OpenDS driving simulator software. The dataset is recorded in synchronized RGB, infrared and depth modalities with the resolution of 320 $\times$ 240 pixels and the frame rate of 30 fps. Example recordings for the three modalities are shown in Fig. B.2.

For each subject, there are in total 5 recordings each containing 42 gestures for 5 different gestures together with *other* and *none* gestures for each hand. Each recording of a subject was recorded under different lighting conditions: at room lights, at darkness, with an external light source from left, with an external light source from right, and under intensive lighting from both sides. We randomly shuffled the order of the subjects and split the dataset by subject into training (72%) and testing (28%) sets. Recordings from subjects 1 to 18 (including) belong to the training set, and recordings from subjects 19 to 25 (including) belong to the test set.

The micro gestures that the subjects had to perform should be natural and should neither distract them nor require them to take their hands off the wheel while performing.

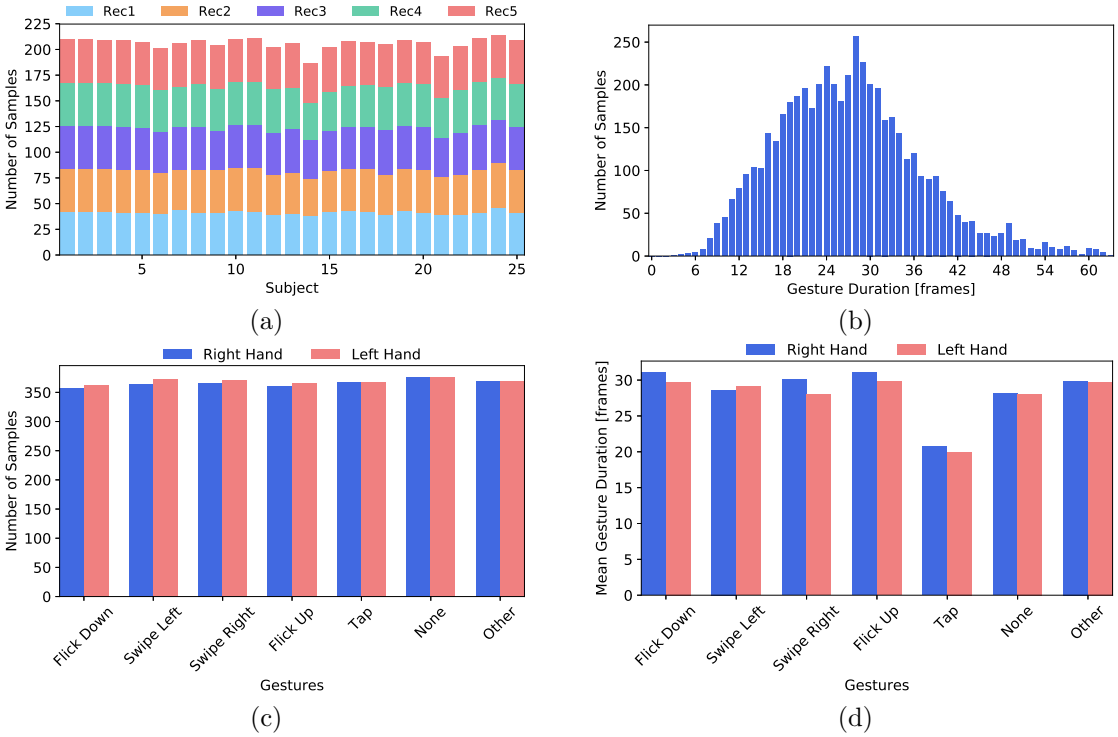


Figure B.4: Statistics of the collected dataset: (a) Number of samples per subject, (b) Histogram of the duration of the gestures, (c) Number of samples per gesture, (d) Mean duration of gestures.

They should also be quickly executable. Therefore, five micro gestures were selected, which are "*swipe right*", "*swipe left*", "*flick down*", "*flick up*" and "*tap*". The former four gestures require the movement of only the thumb, while "*tap*" is performed by lightly patting the side of the wheel with four fingers. Fig. B.3 shows the illustration of the selected five micro gestures for the left and right hands.

Additionally, we introduce the "*other*" and "*none*" gestures. For each record, three "*none*" and "*other*" gestures were specifically selected from the recorded data. With "*other*" label, the network learns the drivers' other movements when a gesture is not performed. Whereas, with "*none*" label, the network learns that the drivers' hands are steady (i.e. there is no movement or gestures). The inclusion of "*none*" and "*other*" action classes in the recorded dataset enables robust online recognition due to the availability of continuous analysis. Regarding the annotations, the gestures were annotated by hand with their starting and ending frame numbers.

Fig. B.4 shows the statistics of the collected dataset. Fig. B.4 (a) shows the number of samples from 5 recordings for each subject, which is around 210. Fig. B.4 (b) shows the histogram of the gesture duration (frames). Fig. B.4 (c) shows that the number of samples for each class are balanced for both right and left hands. In Fig. B.4 (d), mean gesture duration for each action class are given. This figure shows that "*tap*" action

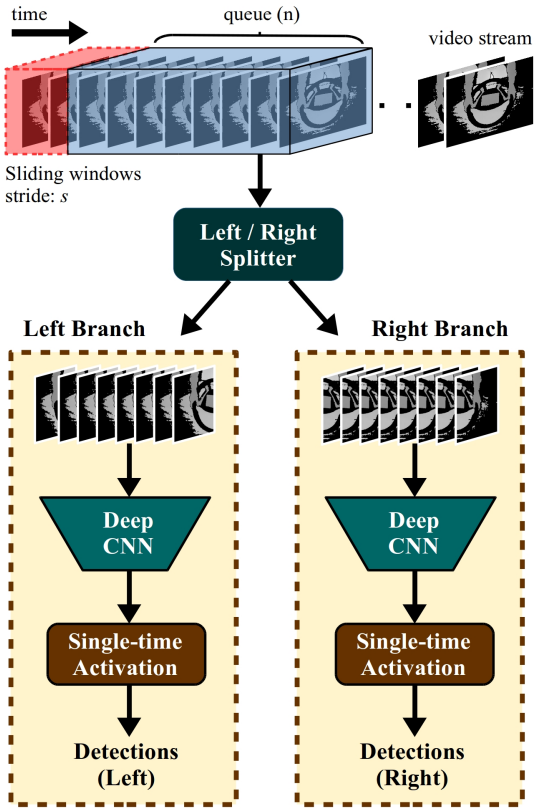


Figure B.5: Online recognition architecture workflow. A sliding window holding n frames is fed to a splitter. Splitter feeds the left and right halves of the clip frames to separate branches. Then, a resource efficient 3D CNN model, which is trained separately for each branch, is applied to obtain class conditional scores of each gesture. Finally, *single-time activation* block is applied to get detections.

can be executed very fast compared to the other action classes. Mean gesture durations for "*none*" and "*other*" action classes are kept quite similar to the "*flick down/up*" and "*swipe left/right*" action classes.

B.3 Methodology

B.3.1 Network Architecture

The general network architecture is depicted in Fig. B.5. For each modality, there is a queue holding the last n frames of the video stream. Then, a video splitter is applied in order to separate the left and right halves of the video. The right and left halves contain the right and left hand related information, respectively. The reason for this splitting is to remove the unrelated video segment, which behaves as noise to the network. Afterwards, left and right video splits are fed to an offline trained CNN to get

Algorithm 3 Online recognition algorithm

Input: Frames in a "sliding window" strided by 1 over a test video
Output: The sequence π of the gestures inside the video

```
for each "sliding window" with the length of l do
    calculate p_{t,i} for each class
    if any p_{t,i}>th_s then
        record classification outputs in c
        if all p_{t,i}<th_e then
            stop recording
            calculate the average classification result of c
            add the result to π
```

class conditional gesture scores for each hand. After this step, a single-time activation block is applied to get final detections.

B.3.2 Offline Recognition

Several resource efficient 3D CNN models are used for offline trainings: 3D-MobileNet, 3D-MobileNetV2, 3D-ShuffleNet, 3D-ShuffleNetV2, and 3D-SqueezeNet. The details of these models can be found in [21]. The left and right hands are trained separately, which gives the opportunity to recognize them independently. This way, simultaneous gestures from left and right hands can be recognized as tuples, which can be registered to an extra class as in [25].

B.3.3 Online Recognition

Online recognition is designed and evaluated for real driving scenarios. The test videos for online recognition from the DriverMHG dataset are unsegmented. Each test video has roughly 6500 frames and includes around 30 hand gestures. For online recognition, there are basically three requirements: (i) Detection of the starting/ending of the performed micro gestures, (ii) single-time recognition of the performed micro hand gestures, and (iii) classification of the performed micro hand gestures. All these tasks above should be implemented in real-time. Considering the aforementioned requirements, we propose *Algorithm 3* for online recognition, whose details are as follows.

B.3.3.1 Detection of the starting/ending of the performed micro gestures

It is essential to detect the starting and ending of a micro gesture for the created HCI system. In our proposed algorithm, there is no need for a separate detector, which saves a lot of computation and memory cost.

According to the recorded videos, the transition from *none* or *other* to any micro hand gesture represents the start of a micro gesture; on the contrary, the transition from any

micro gesture to *none* or *other* represents the end of a micro gesture. To detect such a transition, we use a sliding window to calculate its probability as follows:

$$p_{t,i} = \frac{\overbrace{\sum_{n=t-l+1}^{t-\frac{l}{2}} P_{n,other/none}}^{\text{none/other score}} + \overbrace{\sum_{n=t-\frac{l}{2}+1}^{t} P_{n,i}}^{i^{th}\text{ class score}}}{l}, \tag{B.1}$$

where i, l and t denote the class type, window length and time step, respectively. l is set to 64 in our experiments as it achieves the best performance. P is the class conditional softmax score of the CNN output. The first half of Eq. (B.1) is the average score of *other/none*, and the second half is the average score of i^{th} class. Correspondingly, $p_{t,i}$ becomes the transition probability from *other/none* to i^{th} class at time instant t. A simplified example in Fig. B.6 helps to better understand the probability calculation. Class 5 and 6 are *none* and *other* classes, respectively.

We set two hyperparameters th_s and th_e to indicate the start and the end of a micro gesture. For every timestamp, we calculate the transition from *none* as well as from *other* to each micro gesture (totally 10 possible paths). If any probability is larger than the threshold th_s, the micro gesture starts; if no probability is larger than th_e, the micro gesture ends. Those two thresholds can be different depending on different models.

B.3.3.2 Single-time recognition of the performed micro gestures

The frames between the start and the end of a micro gesture are regarded as a valid clip. It is self-evident to pick the micro gesture with the highest score. This approach benefits from temporal ensembling since it avoids the fluctuations of some false positive classifications.

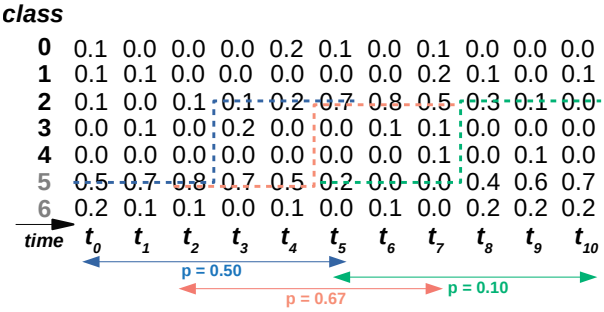


Figure B.6: Illustration for the detection of the start and the end of a micro gesture. For the sake of simplicity, the length of sliding window is set to 6. Only three time instances and the transition from *none* to gesture class 2 are depicted. Values 0.45, 0.62, 0.10 are the probabilities of detecting the transition pattern for the three time instances. Best in color.

Combining the detection of a valid clip and single-time recognition, the algorithm of the whole online detection is described in *Algorithm 3*.

B.3.3.3 Classification of performed micro gestures

To evaluate the performance of this online recognition algorithm, we use the Levenshtein distance to measure the difference between the output sequences and the ground truth sequences of input videos as in [22]. The Levenshtein distance represents the number of item-level changes, such as insertion, deletion, or substitutions. If the prediction is the same as the ground truth, the Levenshtein distance is then 0. The accuracy is 1 minus the fraction of the Levenshtein distance and the length of the ground truth sequence.

B.4 Experiments

In this section, we evaluated the performances of our approach for offline classification accuracy, including the impact of modality fusion on offline classification accuracy, and also for the online classification accuracy.

B.4.1 Evaluation for Offline Classification Accuracy

The recorded DriverMHG is evaluated for offline classifications accuracy using different types of resource efficient 3D CNN architectures. However, although the number of training samples per class is sufficient to train a deep CNN, the small number of classes leads to a relatively small dataset compared to other publicly available datasets, which leads to overfitting. Therefore we have initialized the weights of our models with Jester pretraining. Jester dataset is currently the largest available public dataset, which is a large collection of densely labeled video clips that shows humans performing pre-defined hand gestures in front of a laptop camera or webcam. It contains around 150,000 gesture videos.

Table B.1 shows our offline classification accuracy results using both the Jester and the DriverMHG datasets. The models are trained separately for left and right hands and the average is reported in this table. The evaluations are done by applying five 3D resource efficient architectures, which are 3D-SqueezeNet, 3D-ShuffleNetV1, 3D-ShuffleNetV2, 3D-MobileNetV1, and 3D-MobileNetV2 for four complexity levels using the scaling factor *width multiplier*. This scaling was not possible for the 3D-SqueezeNet hence the result is reported only for one complexity level for this architecture.

The applied approach provides very good offline classification accuracies on both datasets. The best classification accuracies are obtained with 3D-MobileNetV2 1.0x and 3D-ShuffleNetV2 2.0x for the Jester and DriverMHG datasets, respectively, on the RGB modality.

In order to understand the performance of the proposed architecture on different modalities, we also evaluated offline classification accuracies with the infrared and depth modalities existing in our dataset. Table B.1 shows the results achieved for different modalities. Out of all modalities, the infrared modality provides the best result (91.56%

Model	MFLOPs	Params	Speed (cps)		Accuracy (%)					
					Jester	DriverMHG			DriverMHG (fusion)	
			Titan XP	Jetson TX2	RGB	RGB	IR	D	RGB-IR	RGB-IR-D
3D-ShuffleNetV1 0.5x	76	0.27M	398	69	89.23	89.49	88.53	77.59	90.46	90.40
3D-ShuffleNetV2 0.25x	115	0.24M	442	82	86.91	87.83	87.90	73.21	89.08	88.39
3D-MobileNetV1 0.5x	97	0.88M	290	57	87.61	86.93	84.09	75.73	88.45	88.25
3D-MobileNetV2 0.2x	61	0.23M	357	42	86.43	88.18	85.68	76.49	89.22	88.46
3D-ShuffleNetV1 1.0x	198	0.97M	269	49	92.27	89.41	90.25	83.48	91.02	91.22
3D-ShuffleNetV2 1.0x	194	1.33M	243	44	91.96	88.93	90.12	77.32	90.46	90.60
3D-MobileNetV1 1.0x	240	3.33M	164	31	90.81	89.15	86.45	81.60	89.77	89.29
3D-MobileNetV2 0.45x	176	0.67M	203	19	90.21	89.56	89.15	83.61	90.39	90.74
3D-ShuffleNetV1 1.5x	345	2.01M	204	31	93.12	90.32	90.25	83.68	91.50	90.74
3D-ShuffleNetV2 1.5x	290	2.57M	186	34	93.16	89.49	90.46	80.71	91.08	90.60
3D-MobileNetV1 1.5x	427	7.34M	116	19	91.28	88.59	89.70	84.79	90.46	90.95
3D-MobileNetV2 0.7x	324	1.32M	130	13	93.34	90.95	89.08	**86.93**	90.54	90.81
3D-ShuffleNetV1 2.0x	531	3.66M	161	24	93.54	90.95	90.74	83.89	91.78	91.78
3D-ShuffleNetV2 2.0x	436	5.47M	146	26	93.71	**91.52**	91.23	83.67	92.40	91.92
3D-MobileNetV1 2.0x	660	12.93M	88	15	92.56	89.15	89.08	84.59	90.81	90.87
3D-MobileNetV2 1.0x	559	2.39M	93	9	**94.59**	91.36	**91.56**	85.49	**92.88**	**92.47**
3D-SqueezeNet	922	1.85M	682	46	90.77	91.22	85.24	81.10	92.53	92.05

Table B.1: Comparison of different 3D CNN architectures over offline classification accuracy, number of parameters, computation complexity (FLOPs) and speed on two different platforms. Models are trained separately for left and right hands and accuracies are calculated by dividing the number of correctly classified gestures to the total number of gestures. The calculations of parameters and FLOPs are done for the Jester dataset [21] for 16 frames input with 112×112 spatial resolution. For the DriverMHG dataset, 32 frames input with 112×112 spatial resolution are used for trainings. For speed calculations (clips per second - cps), Titan Xp and Jetson TX2 platforms are used with batch size of 8.

achieved with 3D-MobileNetV2 1.0x) since it is invariant to illumination. Although we expect similar results for the depth modality, it performs the worst due to the inferior quality at intensive lighting conditions.

Fig. B.7 shows an example for depth images under three lighting scenarios, which are under heavy solar radiation, medium solar radiation, and light solar radiation. As it is clear from this figure, under intense lighting conditions, the quality of the depth modality drops significantly.

Table B.1 also reports the number of parameters, FLOPs and speed (clip per second - cps). As models get more complex, the number of parameters and FLOPs increase; as a corollary, the speed reduces. It must be noted that 3D-SqueezeNet is comparatively much faster although its number of FLOPs is the highest. This is because it is the only architecture that does not make use of depthwise convolutions, and the CUDNN library is specifically optimized for standard convolutions.

Figure B.7: Depth images under different lighting scenarios: (1) With heavy solar radiation, (2) with medium solar radiation, (3) with light solar radiation.

Modality	Levenshtein Accuracy (%)
RGB	**74.00**
IR	72.90
Depth	56.49

Table B.2: Evaluation of the online detection using model 3D-MobileNetV2 1.0x.

B.4.2 Impact of Modality Fusion on Offline Classification Accuracy

In addition to the mono-modal analysis on RGB, infrared and depth modalities, we have analyzed the impact of fusion of multiple modalities on dynamic micro hand gesture recognition using the score level fusion strategy.

According to the reported results in Table B.1, the fusion of all three modalities enhances the offline classification accuracy with all the applied architectures for all complexity levels. However, another interesting result is that the fusion of RGB and infrared modalities performs better than the fusion of RGB, infrared and depth modalities. The reason is again the poor quality of depth modality for intensive lighting scenarios, which degrades the fusion performance.

B.4.3 Evaluation for Online Classification Accuracy

Here, we use the 3D-MobileNetV2 as the deep CNN model in Fig. B.5 for online evaluation, as its offline performance is in the lead according to the results reported in Table B.1.

Table B.2 shows the online detection results for all RGB, infrared, and depth modalities. The best performance is achieved by modality RGB with 74%. The results with infrared modality are slightly worse than the results with RGB modality. Depth modality provides relatively poor performance, only 56.49%. It is because that the depth images flicker too much, especially under intense lighting. Since the network is trained for recognizing micro gestures, the flickering (in a very small area) can result in false positives.

Appendix C

Driver Anomaly Detection Dataset

Convolutional Neural Networks with 3D kernels (3D CNNs) currently achieve state-of-the-art results in video recognition tasks due to their supremacy in extracting spatiotemporal features within video frames. There have been many successful 3D CNN architectures surpassing state-of-the-art results successively. However, nearly all of them are designed to operate offline creating several serious handicaps during online operation. Firstly, conventional 3D CNNs are not dynamic since their output features represent the complete input clip instead of the most recent frame in the clip. Secondly, they are not temporal resolution-preserving due to their inherent temporal downsampling. Lastly, 3D CNNs are constrained to be used with fixed temporal input size limiting their flexibility. In order to address these drawbacks, we propose dissected 3D CNNs, where the intermediate volumes of the network are dissected and propagated over depth (time) dimension for future calculations, substantially reducing the number of computations at online operation. For action classification, the dissected version of ResNet models performs 77-90% fewer computations at online operation while achieving ∼5% better classification accuracy on the Kinetics-600 dataset than conventional 3D-ResNet models. Moreover, the advantages of dissected 3D CNNs are demonstrated by deploying our approach onto several vision tasks, which consistently improved the performance. We make the DAD dataset and our source code publicly available at https://www.mmk.ei.tum.de/dad/ and https://github.com/okankop/Driver-Anomaly-Detection, respectively.

This section is based on our publication *Driver Anomaly Detection: A Dataset and Contrastive Learning Approach* [27].

C.1 Introduction

Driving has become an indispensable part of modern life providing a high level of convenient mobility. However, this strong dependency on driving also leads to an increased number of road accidents. According to the World Health Organization's estimates, 1.25 million people die in road accidents per year, and up to 50 million people get injured. Human factors are the main contributing cause in almost 90% of the road accidents having distraction as the main factor for around 68% of them [290]. Accordingly, the development of a reliable Driver Monitoring System (DMS), which can supervise a driver's performance, alertness, and driving intention, contains utmost importance to prevent human-related road accidents.

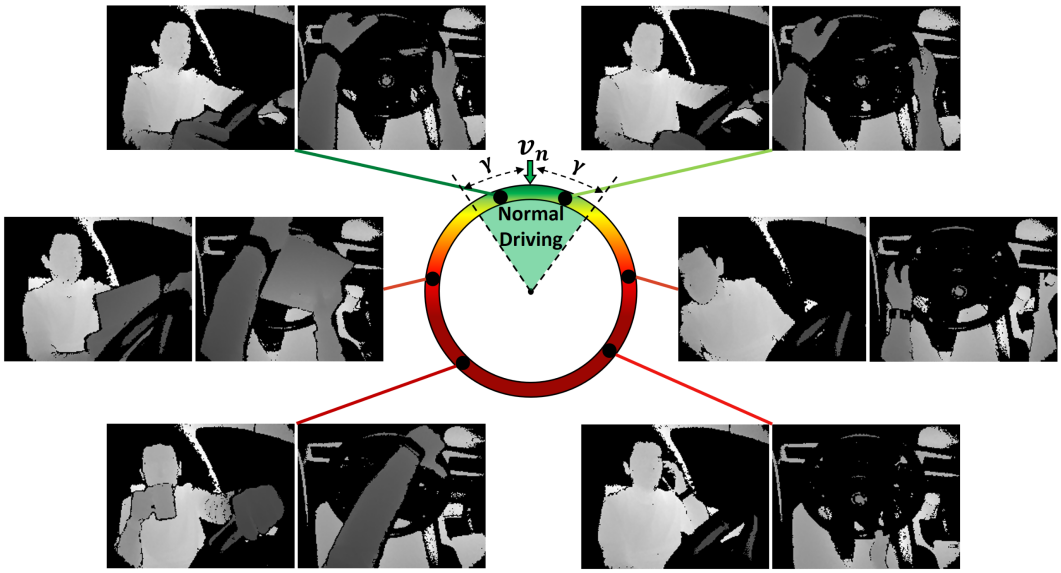


Figure C.1: Using contrastive learning, normal driving template vector $\boldsymbol{v_n}$ is learned during training. At test time, any clip whose embedding is deviating more than threshold γ from normal driving template $\boldsymbol{v_n}$ is considered as anomalous driving. Examples are taken from new introduced Driver Anomaly Detection (DAD) dataset for front (left) and top (right) views on depth modality.

Due to the increased popularity of deep learning methods in computer vision applications, there have been several datasets to facilitate video-based driver monitoring [291, 292, 293]. However, all these datasets are partitioned into a finite set of known classes, such as normal driving class and several distraction classes, with equivalent training and testing distribution. In other words, these datasets are designed for *closed set recognition*, where all samples in their test set belong to one of the K known classes that the networks are trained with. This arises a very important question: ***How would the system react if an unknown class is introduced to the network?*** This obscurity is a serious problem since there might be unbounded many distracting actions that a driver can do while driving.

Different from available datasets and majority research on DMS applications, we propose an *open set recognition* approach for video-based driver monitoring. Since the main purpose of a DMS is to ensure that driver drives attentively and safely, which is referred as *normal driving* in this work, we propose a deep contrastive learning approach to learn a metric in order to distinguish normal driving from anomalous driving. Fig. C.1 illustrates the proposed approach.

In order to facilitate further research, we introduce a large-scale, multi-view, multi-modal Driver Anomaly Detection (DAD) dataset. The DAD dataset contains the normal driving class together with a set of anomalous driving actions in its training set. However, there are several unseen anomalous actions in the test set of the DAD dataset that still need to be distinguished from normal driving. We believe that the DAD dataset addresses the true nature of driver monitoring.

Overall, the main contributions of this work can be summarized as:

- We introduce the DAD dataset, which is the first video-based open set recognition dataset for vision-based driver monitoring applications. The DAD dataset is multi-view (front and top views), multi-modal (depth and infrared modalities) and large enough to train deep Convolutional Neural Network (CNN) architectures from scratch.
- We propose a deep contrastive learning approach to distinguish normal driving from anomalous driving. Although contrastive learning has been popular for unsupervised metric learning recently, we prove its effectiveness by achieving 0.9673 AUC in the test set of the DAD dataset.
- We present a detailed ablation study on the DAD dataset and proposed a contrastive learning approach in order to give better insights about them.

C.2 DAD Dataset

There are several vision-based driver monitoring datasets that are publicly available, but for the task of open set recognition such that normal driving should still be distinguished from unseen anomalous actions, there has been none. In order to fill this research gap, we have recorded the Driver Anomaly Detection (DAD) dataset, which contains the following properties:

- The DAD dataset is large enough to train DNN architectures from scratch.
- The DAD dataset is multi-modal containing depth and infrared modalities such that the system is operable at different lighting conditions.
- The DAD dataset is multi-view containing front and top views. These two views are recorded synchronously and complement each other.
- The videos are recorded with 45 frame-per-second providing high temporal resolution.

We have recorded the DAD dataset using a driving simulator that is shown in Fig. C.2. The driving simulator contains a real BMW car cockpit, and the subjects are instructed to drive in a computer game that is projected in front of the car. Two Infineon CamBoard pico flexx cameras are placed on top and in front of the driver. The front camera is installed to record the drivers' head, body and visible part of the hands (left hand is mostly obscured by the driving wheel), while the top camera is installed to focus on the drivers' hand movements. The dataset is recorded in synchronized depth and infrared modalities with the resolution of 224×171 pixels and the frame rate of 45 fps. Example recordings for the two views and two modalities are shown in Fig. C.2.

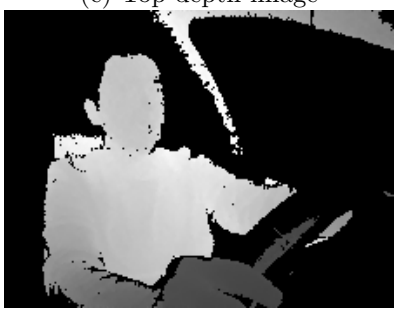
For the dataset recording, 31 subjects are asked to drive in a computer game performing either *normal driving* or *anomalous driving*. Each subject belongs either to

(a) Camera placements in the simulator

(b) Camera

(c) Top depth image

(d) Top infrared image



(e) Front depth image

(f) Front infrared image

Figure C.2: Environment for data collection. (a) Driving simulator with camera placements. (b) Infineon CamBoard pico flexx camera installed for front and top views. Examples of (c) top depth, (d) top infrared, (e) front depth and (f) front infrared recordings.

the training or to the test set. The training set contains recordings of 25 subjects and each subject has 6 normal driving and 8 anomalous driving video recordings. Each normal driving video lasts about 3.5 minutes and each anomalous driving video lasts about 30 seconds containing a different distracting action. The list of distracting actions recorded in the training set can be found in Table C.1. In total, there are around 550 minutes recording for normal driving and 100 minutes recording of anomalous driving in the training set.

The test set contains 6 subjects and each subject has 6 video recordings lasting around 3.5 minutes. Anomalous actions occur randomly during the test video recordings. Most importantly, there are 16 distracting actions in the test set that are

Anomalous Actions in Training Set	Anomalous Actions in Test Set		
Talking on the phone-left	Talking on the phone-left	Adjusting side mirror	Wearing glasses
Talking on the phone-right	Talking on the phone-right	Adjusting clothes	Taking off glasses
Messaging left	Messaging left	Adjusting glasses	Picking up something
Messaging right	Messaging right	Adjusting rear-view mirror	Wiping sweat
Talking with passengers	Talking with passengers	Adjusting sunroof	Touching face/hair
Reaching behind	Reaching behind	Wiping nose	Sneezing
Adjusting radio	Adjusting radio	Head dropping (dozing off)	Coughing
Drinking	Drinking	Eating	Reading

Table C.1: Anomalous actions in the training and test sets. 16 actions in the test set that are not available in the training set are highlighted in red color.

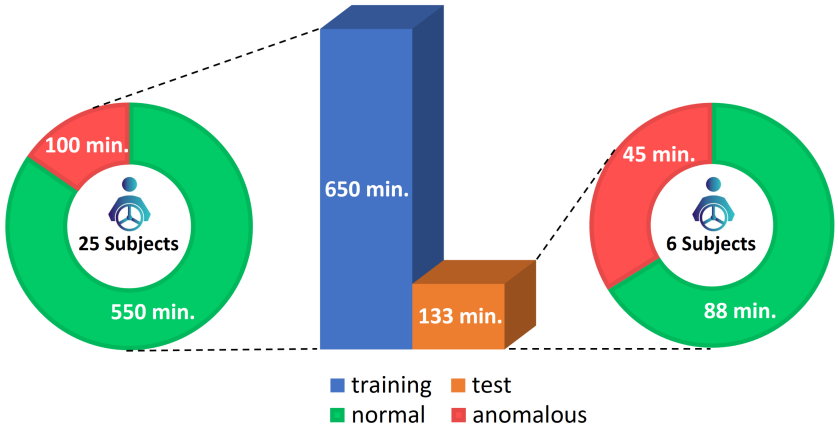


Figure C.3: The DAD dataset statistics.

not available in the training set, which can be found in Table C.1. *Because of these additional distracting actions, the networks need to be trained according to the open set recognition task and distinguish normal driving no matter what the distracting action is.* The complete test consists of 88 minutes recording for normal driving and 45 minutes recording of anomalous driving. The test set constitutes 17% of the complete DAD dataset, which is around 95 GB. The dataset statistics can be found in Fig. C.3.

C.3 Methodology

C.3.1 Contrastive Learning Framework

Our motivation is to learn a compact representation for normal driving such that any action deviating from normal driving beyond a threshold can be detected as anomalous action. Accordingly, inspired by recent progress in contrastive learning algorithms, we try to maximize the similarity between normal driving samples and minimize the similarity between normal driving and anomalous driving samples in the latent space

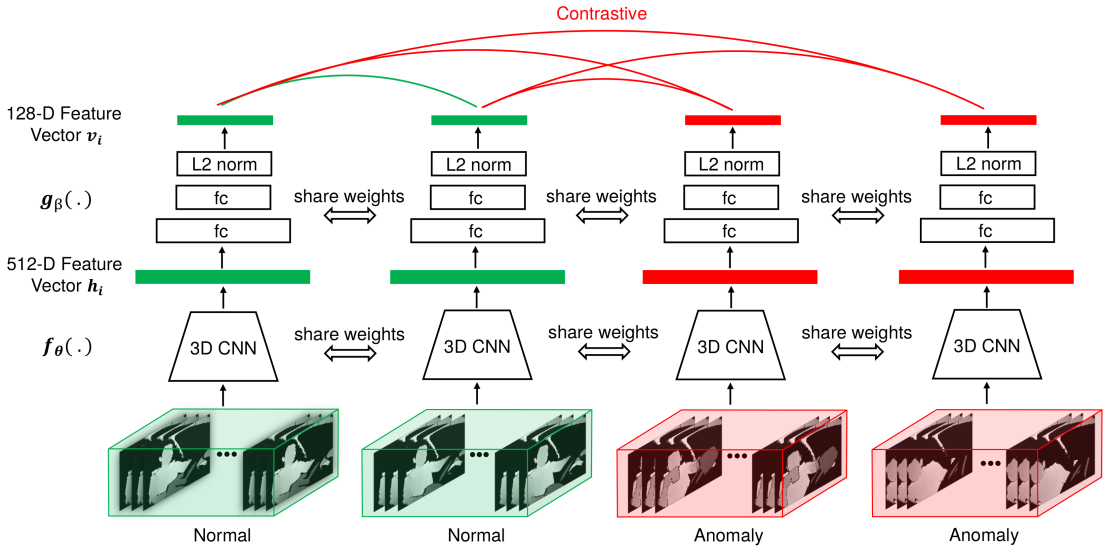


Figure C.4: Contrastive learning framework for driver anomaly detection task. A pair of normal driving clips a number of anomaly driving clips (2 in this example) are fed to a base encoder $f_\theta(.)$ and projection head $g_\beta(.)$ to extract visual representations of $\boldsymbol{h_i}$ and $\boldsymbol{v_i}$, respectively. Once training is completed, projection head is removed, and only the encoder $f_\theta(.)$ is used for test time recognition.

using a contrastive loss. Fig. C.4 illustrates the applied framework, which has three major components:

- **Base encoder $\boldsymbol{f_\theta(.)}$** is used to extract vector representations of input clips. $f_\theta(.)$ refers to a 3D CNN architecture with parameters θ. We performed experiments with ResNet-18 and various resource efficient 3D CNNs to transform input $\boldsymbol{x_i}$ into $\boldsymbol{h_i}\in\mathbb{R}^{512}$ via $\boldsymbol{h_i}=f_\theta(\boldsymbol{x_i})$.
- **Projection head $\boldsymbol{g_\beta(.)}$** is used to map $\boldsymbol{h_i}$ into another latent space $\boldsymbol{v_i}$. According to findings in [294], it beneficial to define the contrastive loss on $\boldsymbol{v_i}$ rather than $\boldsymbol{h_i}$. $\boldsymbol{g_\beta(.)}$ refers to MLP with one hidden layer with ReLU activation and has parameters β to achieve transformation of $\boldsymbol{v_i}=g_\beta(\boldsymbol{h_i})=W^{(2)}max(0,W^{(1)}\boldsymbol{h_i})$, where $\boldsymbol{v_i}\in\mathbb{R}^{128}$. After MLP, $\ell 2$ normalization is applied to the embedding $\boldsymbol{v_i}$.
- **Contrastive loss** is used to impose that normalized embeddings from the normal driving class are closer together than embeddings from different anomalous action classes. For this reason, positive pairs in the contrastive loss are always selected from normal driving clips, whereas anomalous driving clips are used only as negative samples.

We divide our normal and anomalous videos into clips for the training. Within a mini-batch, we have K normal driving clips and M anomalous driving clips with index $i\in\{1,...,K+M\}$. Final embedding of the i^{th} normal and anomalous driving clips are

denoted as $\boldsymbol{v_{ni}}$ and $\boldsymbol{v_{ai}}$, respectively. There are in total $K(K-1)$ positive pairs and KM negative pairs in every mini-batch. For the supervised contrastive learning approach that we have applied for the task of driver anomaly detection task, the loss takes the following final form:

$$\mathcal{L}_{ij} = -\log \frac{exp(\boldsymbol{v}_{\boldsymbol{ni}}^{\mathrm{T}} \boldsymbol{v}_{\boldsymbol{nj}}/\tau)}{exp(\boldsymbol{v}_{\boldsymbol{ni}}^{\mathrm{T}} \boldsymbol{v}_{\boldsymbol{nj}}/\tau) + \sum_{m=1}^{M} exp(\boldsymbol{v}_{\boldsymbol{ni}}^{\mathrm{T}} \boldsymbol{v}_{\boldsymbol{am}}/\tau)}, \tag{C.1}$$

$$\mathcal{L} = \frac{1}{K(K-1)} \sum_{i=1}^{K} \sum_{j=1}^{K} \mathbb{1}_{j \neq i} \mathcal{L}_{ij}, \tag{C.2}$$

where $\mathbb{1} \in \{0, 1\}$ is an indicator function that returns 1 if $j \neq i$ and 0 otherwise, and $\tau \in (0, \infty)$ is a scalar temperature parameter that can control the concentration level of the distribution [295]. Typically, τ is chosen between 0 and 1 to amplify the similarity between samples, which is beneficial for training. The inner product of vectors measures the cosine similarity between encoded feature vectors because they are all $\ell 2$ normalized. By optimizing Eq. (C.2), the encoder is updated to maximize the similarity between the normal driving feature vectors $\boldsymbol{v_{ni}}$ and $\boldsymbol{v_{nj}}$ while minimizing the similarity between the normal driving feature vector $\boldsymbol{v_{ni}}$ and all other anomalous driving feature vectors $\boldsymbol{v_{am}}$ in the same mini-batch.

Noise Contrastive Estimation. The representation learned by Eq. (C.2) can be improved by introducing many more anomaly driving clips (i.e. negative samples). In the extreme case, we can use the complete training samples of anomalous driving. However, this is too expensive considering the limited memory of the used GPU. Noise Contrastive Estimation [296] can be used to approximate the full softmax distribution as in [296, 297]. In our implementation, we have used the m negative samples in our mini-batch and applied $(m+1)$-way softmax classification as also used [298, 299, 300]. Different from these works, we do not use a memory bank and optimize our framework using only the elements in the mini-batch.

C.3.2 Test Time Recognition

The common practice to evaluate learned representations is to train a linear classifier on top of the frozen base network [298, 299, 300, 294]. However, this final training is tricky since representations learned by unsupervised and supervised training can be quite different. For example, training of the final linear classification is performed with the learning rate of 30, although unsupervised learning is performed with the initial learning rate of 0.01. In addition, authors in [297] apply k-nearest neighbours (kNN) classification for the final evaluation. However, kNN also requires distance calculation with all training clips for each test clip, which is computationally expensive.

For the test time recognition, we propose an evaluation protocol that requires neither any further training nor complex computations. After the training phase, we throw away the projection head as in [294] and use the trained 3D CNN model to encode every normal

driving training clips $\boldsymbol{x_i}$, $i \in \{1, ..., N\}$ into a set of $\ell 2$ normalized 512-dimensional feature representations. Afterwards, normal driving template vector $\boldsymbol{v_n}$ can be calculated with:

$$\boldsymbol{v_n} = \frac{1}{N} \sum_{i=1}^{N} \frac{f_\theta(\boldsymbol{x_i})}{\|f_\theta(\boldsymbol{x_i})\|_2}. \tag{C.3}$$

To classify a test video clip $\boldsymbol{x_i}$, we encode it again into a $\ell 2$ normalized 512-dimensional vector and compute the cosine similarity between the encoded clip and $\boldsymbol{v_n}$ by:

$$sim_i = \boldsymbol{v_n}^{\mathrm{T}} \frac{f_\theta(\boldsymbol{x_i})}{\|f_\theta(\boldsymbol{x_i})\|_2}. \tag{C.4}$$

Finally, any clip whose similarity score is below a threshold, $sim_i < \gamma$, is classified as anomalous driving. This way, only a simple vector multiplication is performed for test time evaluation. Moreover, the similarity score of the test clip sim_i gives the severity of the anomalous behavior.

Fusion of Different Views and Modalities. The DAD dataset contains front and top views; and depth and infrared modalities. We have trained a separate model for each view and modality and fused them later with decision level fusion. As an example, the fused similarity score for top view depth and infrared modalities is calculated with:

$$sim^{(top)}_{(DIR)} = \frac{sim^{(top)}_{(D)} + sim^{(top)}_{(IR)}}{2}. \tag{C.5}$$

It must be noted that each applied view and modality increases the required memory and inference time, which would be critical for autonomous driving applications.

C.4 Experiments

Baseline Results. We have used ResNet-18 as the base encoder for the baseline results. All the models in the experiments are trained from scratch unless otherwise specified. For every view and modality, a separate model is trained and individual results, as well as fusion results, are reported in Table C.2. The thresholds that are achieving the highest classification accuracy are reported in Table C.2. However, true positive rates and false positive rates change according to the applied threshold value. Therefore, we have also reported the AUC of the ROC curve for baseline evaluation.

For all different views, the fusion of different modalities always achieves better performance compared to single modalities and views. This shows that different views/modalities in the dataset contain complementary information. Fusion of top/front views and depth/infrared modalities achieves the best performance with 0.9655 AUC. Using this fusion network, the visualization for a continuous video stream is illustrated in Fig. C.5.

Resource Efficient Base Encoders. For autonomous applications, it is critical that the deployed systems should be designed considering resource efficiency. Therefore, we have experimented with different resource efficient 3D CNNs [21] as the base encoder.

Metric	Thresholds γ	Acc. (%)	AUC
Top(D)	0.89	89.13	0.9128
Top(IR)	0.65	83.63	0.8804
Top(DIR)	0.76	87.75	0.9166
Front(D)	0.75	87.21	0.8996
Front(IR)	0.82	83.68	0.8695
Front(DIR)	0.81	88.68	0.9196
Top+Front(D)	0.83	91.60	0.9609
Top+Front(IR)	0.80	87.06	0.9311
Top+Front(DIR)	0.81	**92.34**	**0.9655**

Table C.2: Results obtained by using a ResNet-18 as base encoder. Thresholds that result in highest classification accuracy are reported.

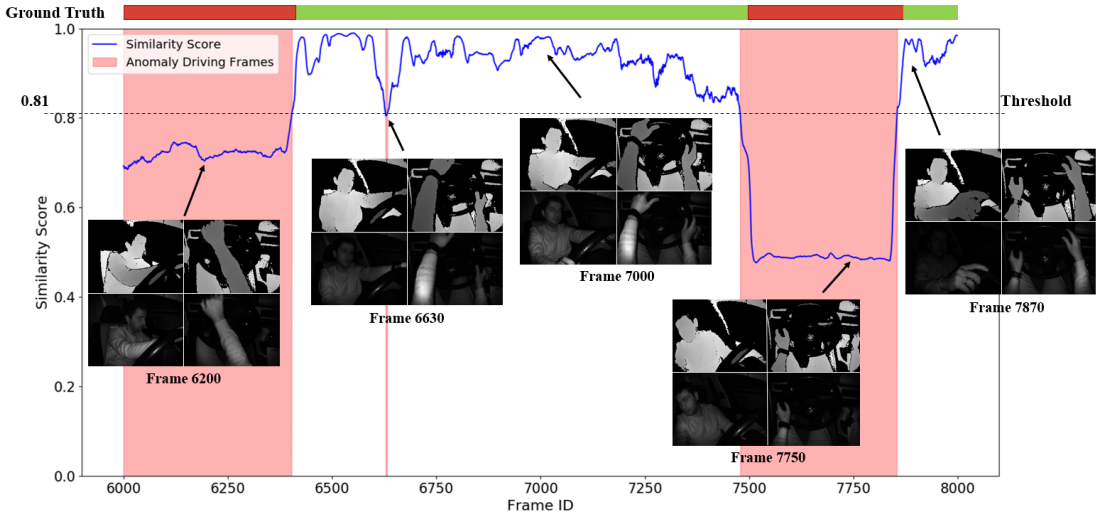


Figure C.5: Illustration of recognition for a continuous video stream using fusion of both views and modalities. Similarity score refers to cosine similarity between the normal driving template vector and base encoder embedding of input clip. The frames are classified as anomalous driving if the similarity score is blow the preset threshold.

Comparative results are reported in Table C.3. Out of all resource efficient 3D CNNs, MobileNetV2 stands out with its performance achieving close to ResNet-18 architecture. More importantly, MobileNetV2 has around 11 times fewer parameters and requires 15 times less computation compared to ResNet-18.

With or Without Pre-training? Transfer learning is a common and effective strategy to improve generalization in small-scale datasets by pretraining network initially with a large-scale dataset [301]. Therefore, in order to investigate the effect of pretraining, we have pretrained our ResNet-18 base encoder on Kinetics-600 for 100 epochs with

Model	Params	MFLOPS	AUC								
			Top			Front			Top+Front		
			Depth	IR	D+IR	Depth	IR	D+IR	Depth	IR	D+IR
MobileNetV1 2.0x	13.92M	707	0.9125	0.8381	0.9097	0.9018	0.8374	0.9057	0.9474	0.9059	0.9533
MobileNetV2 1.0x	3.01M	585	0.9124	0.8531	0.9146	0.8899	0.8355	0.8984	0.9641	0.9154	0.9608
ShuffleNetV1 2.0x	4.59M	558	0.8884	0.8567	0.8926	0.8869	0.8398	0.9000	0.9358	0.9023	0.9480
ShuffleNetV2 2.0x	6.46M	461	0.8959	0.8570	0.9066	0.9002	0.8371	0.9054	0.9490	0.9131	0.9531
ResNet-18 (from scratch)	32.99M	8870	0.9128	0.8804	0.9166	0.8996	0.8695	0.9196	0.9609	0.9311	0.9655
ResNet-18 (pre-trained)	32.99M	8870	**0.9200**	**0.8857**	**0.9228**	**0.9020**	0.8666	0.9128	**0.9646**	0.9227	0.9620
ResNet-18 (post-processed)	32.99M	8870	0.9143	0.8827	0.9182	**0.9020**	**0.8737**	**0.9223**	0.9628	**0.9335**	**0.9673**

Table C.3: Comparison of different network architectures over AUC, number of parameters and MFLOPS. All architectures takes 16 frames input with 112×112 spatial resolution.

contrastive loss similar to our contrastive learning approach described in Section C.3. We have not applied CE loss that is common for training classification tasks since feature representations learned by CE loss and contrastive loss would be quite different, hence can hinder the transfer learning performance. Before fine-tuning, we have modified the initial convolution layer of the pretrained network to accommodate single channel input by averaging weights of 3 channels. Afterwards, we fine-tune the network using the DAD dataset. Comparative results are reported in Table C.3 that pretrained base encoder does not show apparent advantages compared to base encoder trained from scratch. We infer that our DAD dataset is large enough and the networks that are trained from scratch can already learn all distinctive features without the need of transfer learning.

Post-processing. It is a common approach to apply post-processing in order to prevent fluctuation of detected scores [22]. For instance, the misclassification between frames 6500 and 6750 in Fig. C.5 can be prevented by such post-processing. Therefore, we have applied a simple low pass filtering (i.e. averaging) on the predicted scores. Instead of making score predictions considering only the current clip, we have applied a running averaging on the k-previous scores. We have experimented with different k values and the best results are achieved when k=6. Comparative results with and without post-processing are reported in Table C.3, where post-processing slightly improves the performance.

Closed set and open set anomalies. We have compared the performance of the proposed architecture over closed set and open set anomalies separately. We achieve a closed set specificity score of 0.8713, an open set specificity score of 0.8252, and an average specificity score of 0.8565. This result verifies that the proposed architecture successfully detects open set anomalies, although closed set performance is still better than open set.

How Training Data Affects the Performance? The quality and the amount of training data is one of the most important factors in the performance of deep learning applications. Therefore we have investigated the impact of different amounts of training data. First, we have created 5 equal folds each containing training data of 5 subjects. Then, keeping all the anomalous driving in the training set, we have gradually increased

Ratio		**AUC**		
$\boldsymbol{\lambda_n}$	$\boldsymbol{\lambda_a}$	**Top**	**Front**	**Top+Front**
20%	100%	0.7956	0.7639	0.8513
40%	100%	0.7795	0.8111	0.8561
60%	100%	0.8599	0.8166	0.8802
80%	100%	0.8998	0.8601	0.9382
100%	20%	0.8025	0.7873	0.8545
100%	40%	0.8103	0.8577	0.9070
100%	60%	0.8694	0.8911	0.9335
100%	80%	0.8854	0.8921	0.9484
100%	100%	**0.9128**	**0.8996**	**0.9609**

Table C.4: Performance comparison using different amount of normal and anomalous driving data in the training. Results are reported for ResNet-18 base encoder on depth modality.

the used folds for normal driving data. We have applied the same procedure by switching the normal and anomalous driving subsets. The comparative results are reported in Table C.4, where λ_n and λ_a refer to the proportion of the used training data for normal driving and anomalous driving subsets, respectively.

The results in Table C.4 show that as we increase the amount of normal and anomalous driving videos, achieved performance also increases accordingly. This is natural since we need more normal driving data in order to increase the generalization strength of the learned embeddings. We also need enough anomalous driving data in the training set to draw the boundary of the normal driving embedding and increase the compactness of the learned representation.

List of Figures

List of Tables

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. 2016.

[2] M. Vrigkas, C. Nikou, and I. A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2:28, 2015.

[3] C. Gu, C. Sun, D. A. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, et al. Ava: A video dataset of spatio-temporally localized atomic visual actions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6047–6056, 2018.

[4] J. Materzynska, G. Berger, I. Bax, and R. Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[5] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[6] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[7] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1010–1019, 2016.

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, S. R., and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[9] F. Caba Heilbron, V. Escorcia, B. Ghanem, and J. Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[10] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, T. Yan, L. Brown, Q. Fan, D. Gutfreund, C. Vondrick, et al. Moments in time dataset: one million videos for event understanding. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 42(2):502–508, 2019.

[11] R. G. Barker and H. F. Wright. Midwest and its children: The psychological ecology of an american town. 1955.

[12] J. Gonzalez i Sabaté. *Human sequence evaluation the key-frame approach.* Universitat Autònoma de Barcelona,, 2005.

[13] A. Fardinpour. *Taxonomy of Human Actions for Action-based Learning Assessment in Virtual Training Environments.* PhD thesis, Curtin University, 2016.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[15] Z. Dai, H. Liu, Q. V. Le, and M. Tan. Coatnet: Marrying convolution and attention for all data sizes. *arXiv preprint arXiv:2106.04803*, 2021.

[16] V. I. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (7):677–695, 1997.

[17] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.

[18] O. Köpüklü, F. Herzog, and G. Rigoll. Comparative analysis of cnn-based spatiotemporal reasoning in videos. In *Proceedings of the International Conference on Pattern Recognition*, pages 186–202, 2021.

[19] O. Köpüklü, N. Kose, and G. Rigoll. Motion fused frames: Data level fusion strategy for hand gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2103–2111, 2018.

[20] O. Köpüklü, X. Wei, and G. Rigoll. You only watch once: A unified cnn architecture for real-time spatiotemporal action localization. *arXiv preprint arXiv:1911.06644*, 2019.

[21] O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll. Resource efficient 3d convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[22] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, pages 1–8, 2019.

[23] O. Köpüklü, S. Hörmann, F. Herzog, H. Cevikalp, and G. Rigoll. Dissected 3d cnns: Temporal skip connections for efficient online video processing. *Computer Vision and Image Understanding*, 215:103318, 2022.

[24] O. Köpüklü, M. Taseska, and G. Rigoll. How to design a three-stage architecture for audio-visual active speaker detection in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1193–1203, 2021.

[25] O. Köpüklü, Y. Rong, and G. Rigoll. Talking with your hands: Scaling hand gestures and recognition with cnns. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[26] O. Köpüklü, T. Ledwon, Y. Rong, N. Kose, and G. Rigoll. Drivermhg: A multi-modal dataset for dynamic recognition of driver micro hand gestures and a real-time recognition framework. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, pages 77–84, 2020.

[27] O. Köpüklü, J. Zheng, H. Xu, and G. Rigoll. Driver anomaly detection: A dataset and contrastive learning approach. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 91–100, 2021.

[28] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Online dynamic hand gesture recognition including efficiency analysis. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2(2):85–97, 2020.

[29] O. Köpüklü, M. Babaee, S. Hörmann, and G. Rigoll. Convolutional neural networks with layer reuse. In *Proceedings of the IEEE International Conference on Image Processing*, pages 345–349, 2019.

[30] O. Köpüklü and G. Rigoll. Analysis on temporal dimension of inputs for 3d convolutional neural networks. In *Proceedings of the IEEE International Conference on Image Processing, Applications and Systems*, pages 79–84, 2018.

[31] H. Cevikalp, B. Uzun, O. Köpüklü, and G. Ozturk. Deep compact polyhedral conic classifier for open and closed set recognition. *Pattern Recognition*, 119:108080, 2021.

[32] N. Kose, O. Köpüklü, A. Unnervik, and G. Rigoll. Real-time driver state monitoring using a cnn based spatio-temporal approach. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3236–3242. IEEE, 2019.

[33] M. Babaee, Y. Zhu, O. Köpüklü, S. Hörmann, and G. Rigoll. Gait energy image restoration using generative adversarial networks. In *Proceedings of the IEEE International Conference on Image Processing*, pages 2596–2600, 2019.

[34] S. Hörmann, M. Knoche, M. Babaee, O. Köpüklü, and G. Rigoll. Outlier-robust neural aggregation network for video face identification. In *Proceedings of the IEEE International Conference on Image Processing*, pages 1675–1679, 2019.

[35] M. Kayhan, O. Köpüklü, M. H. Sarhan, M. Yigitsoy, A. Eslami, and G. Rigoll. Deep attention based semi-supervised 2d-pose estimation for surgical instruments. In *Proceedings of the International Conference on Pattern Recognition*, pages 444–460, 2021.

[36] F. Kälber., O. Köpüklü., N. Lehment., and G. Rigoll. U-net based zero-hour defect inspection of electronic components and semiconductors. In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 593–601, 2021.

[37] H. Saribas, H. Cevikalp, O. Köpüklü, and B. Uzun. Trat: Tracking by attention using spatio-temporal features. *Neurocomputing*, 492:150–161, 2022.

[38] Y. Feng, S. Wu, O. Köpüklü, X. Kang, and F. Tombari. Unsupervised monocular depth prediction for indoor continuous video streams. *arXiv preprint arXiv:1911.08995*, 2019.

[39] R. Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[40] S. Herath, M. Harandi, and F. Porikli. Going deeper into action recognition: A survey. *Image and vision computing*, 60:4–21, 2017.

[41] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 23(3):257–267, 2001.

[42] J. Han and B. Bhanu. Individual recognition using gait energy image. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 28(2):316–322, 2005.

[43] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1402, 2005.

[44] D. Weinland, R. Ronfard, and E. Boyer. Free viewpoint action recognition using motion history volumes. *Computer vision and image understanding*, 104(2-3):249–257, 2006.

[45] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. Vancouver, British Columbia, 1981.

[46] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[47] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[48] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proceedings of the European Conference on Computer Vision*, pages 25–36, 2004.

[49] A. A. Efros, A. C. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *Computer Vision, IEEE International Conference on*, volume 3, pages 726–726. IEEE Computer Society, 2003.

[50] I. Laptev. On space-time interest points. *International journal of computer vision*, 64(2-3):107–123, 2005.

[51] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 65–72. IEEE, 2005.

[52] M. Bregonzio, S. Gong, and T. Xiang. Recognising action as clouds of space-time interest points. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1948–1955, 2009.

[53] R. Messing, C. Pal, and H. Kautz. Activity recognition using the velocity histories of tracked keypoints. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 104–111, 2009.

[54] P. Matikainen, M. Hebert, and R. Sukthankar. Trajectons: Action recognition through the motion analysis of tracked features. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 514–521, 2009.

[55] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[56] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3551–3558, 2013.

[57] C. Harris, M. Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[58] G. Willems, T. Tuytelaars, and L. Van Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In *Proceedings of the European Conference on Computer Vision*, pages 650–663, 2008.

[59] A. Klaser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3d-gradients. In *BMVC 2008-19th British Machine Vision Conference*, pages 275–1. British Machine Vision Association, 2008.

[60] L. Yeffet and L. Wolf. Local trinary patterns for human action recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 492–497, 2009.

[61] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[62] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 2005.

[63] I. Laptev, B. Caputo, C. Schüldt, and T. Lindeberg. Local velocity-adapted motion events for spatio-temporal recognition. *Computer vision and image understanding*, 108(3):207–229, 2007.

[64] J. Sun, X. Wu, S. Yan, L.-F. Cheong, T.-S. Chua, and J. Li. Hierarchical spatio-temporal context modeling for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2004–2011, 2009.

[65] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *Bmvc 2009-british machine vision conference*, pages 124–1. BMVA Press, 2009.

[66] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[67] M. Jain, H. Jégou, and P. Bouthemy. Better exploiting motion for better action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2555–2562, 2013.

[68] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3169–3176, 2011.

[69] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *Proceedings of the European Conference on Computer Vision*, pages 428–441, 2006.

[70] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proceedings of the European Conference on Computer Vision Workshops*, pages 1–2, 2004.

[71] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Proceedings of the European Conference on Computer Vision*, pages 143–156, 2010.

[72] M. Marszalek, I. Laptev, and C. Schmid. Actions in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2929–2936, 2009.

[73] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked fisher vectors. In *Proceedings of the European Conference on Computer Vision*, pages 581–595, 2014.

[74] D. Oneata, J. Verbeek, and C. Schmid. Action and event recognition with fisher vectors on a compact feature set. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1817–1824, 2013.

[75] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[76] J. D. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289, 2001.

[77] T. V. Duong, H. H. Bui, D. Q. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 838–845, 2005.

[78] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1521–1527, 2006.

[79] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[80] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4681–4690, 2017.

[81] P. Liu, H. Zhang, K. Zhang, L. Lin, and W. Zuo. Multi-level wavelet-cnn for image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[82] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[83] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. C Van Esesn, A. Awwal, and V. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. 03 2018.

[84] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[85] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4489–4497, 2015.

[86] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang. Real-time action recognition with enhanced motion vector cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2718–2726, 2016.

[87] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[88] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[89] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[90] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[91] B. Zhou, A. Andonian, A. Oliva, and A. Torralba. Temporal relational reasoning in videos. In *Proceedings of the European Conference on Computer Vision*, pages 803–818, 2018.

[92] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. In *Proceedings of the European Conference on Computer Vision*, pages 20–36, 2016.

[93] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

[94] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[95] J. Lin, C. Gan, and S. Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7083–7093, 2019.

[96] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[97] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. pages 2625–2634, 2015.

[98] J. Liu, A. Shahroudy, D. Xu, and G. Wang. Spatio-temporal lstm with trust gates for 3d human action recognition. In *Proceedings of the European Conference on Computer Vision*, pages 816–833, 2016.

[99] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. pages 103–111, 2014.

[100] D. Dwibedi, P. Sermanet, and J. Tompson. Temporal Reasoning in Videos using Convolutional Gated Recurrent Units. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1111–1116, 2018.

[101] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[102] V. Veeriah, N. Zhuang, and G.-J. Qi. Differential recurrent neural networks for action recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4041–4049, 2015.

[103] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[104] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[105] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[106] H. Song, W. Wang, S. Zhao, J. Shen, and K.-M. Lam. Pyramid dilated deeper convlstm for video salient object detection. In *Proceedings of the European Conference on Computer Vision*, pages 715–731, 2018.

[107] L. Zhang, L. Lu, X. Wang, R. M. Zhu, M. Bagheri, R. M. Summers, and J. Yao. Spatio-temporal convolutional lstms for tumor growth prediction by learning 4d longitudinal patient data. *IEEE Transactions on Medical Imaging*, 39(4):1114–1126, 2019.

[108] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[109] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[110] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[111] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[112] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations*, 2020.

[113] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the International Conference on Machine Learning*, pages 10347–10357, 2021.

[114] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision*, pages 213–229, 2020.

[115] Y. Wang, Z. Xu, X. Wang, C. Shen, B. Cheng, H. Shen, and H. Xia. End-to-end video instance segmentation with transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8741–8750, 2021.

[116] C. Plizzari, M. Cannici, and M. Matteucci. Spatial temporal transformer network for skeleton-based action recognition. In *Proceedings of the International Conference on Pattern Recognition*, pages 694–701, 2021.

[117] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman. Video action transformer network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2019.

[118] G. Bertasius, H. Wang, and L. Torresani. Is space-time attention all you need for video understanding? In *Proceedings of the International Conference on Machine Learning*, July 2021.

[119] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. Vivit: A video vision transformer. *arXiv preprint arXiv:2103.15691*, 2021.

[120] H. Fan, B. Xiong, K. Mangalam, Y. Li, Z. Yan, J. Malik, and C. Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021.

[121] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[122] D. Neimark, O. Bar, M. Zohar, and D. Asselmann. Video transformer network. *arXiv preprint arXiv:2102.00719*, 2021.

[123] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 35(1):221–231, 2012.

[124] J. Carreira, E. Noland, A. Banki-Horvath, C. Hillier, and A. Zisserman. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*, 2018.

[125] J. Carreira, E. Noland, C. Hillier, and A. Zisserman. A short note on the kinetics-700 human action dataset. *arXiv preprint arXiv:1907.06987*, 2019.

[126] H. Zhao, A. Torralba, L. Torresani, and Z. Yan. Hacs: Human action clips and segments dataset for recognition and temporal localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8668–8678, 2019.

[127] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[128] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[129] Z. Qiu, T. Yao, and T. Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5533–5541, 2017.

[130] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6202–6211, 2019.

[131] K. Hara, H. Kataoka, and Y. Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[132] C. Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 203–213, 2020.

[133] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4207–4215, 2016.

[134] L. Hedegaard and A. Iosifidis. Continual 3d convolutional neural networks for real-time processing of videos. *arXiv preprint arXiv:2106.00050*, 2021.

[135] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu. Video swin transformer. *arXiv preprint arXiv:2106.13230*, 2021.

[136] J. Wan, Q. Ruan, W. Li, and S. Deng. One-shot learning gesture recognition from rgb-d data using bag of features. *The Journal of Machine Learning Research*, 14(1):2549–2582, 2013.

[137] J. Wan, G. Guo, and S. Z. Li. Explore efficient local features from rgb-d data for one-shot learning gesture recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 38(8):1626–1639, 2015.

[138] Z. Liu, X. Chai, Z. Liu, and X. Chen. Continuous gesture recognition with hand-oriented spatiotemporal feature. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 3056–3064, 2017.

[139] H. Wang, P. Wang, Z. Song, and W. Li. Large-scale multimodal gesture recognition using heterogeneous networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3129–3137, 2017.

[140] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. A. Shah, and M. Bennamoun. Attention in convolutional lstm for gesture recognition. In *Advances in Neural Information Processing Systems*, pages 1957–1966, 2018.

[141] P. Narayana, J. R. Beveridge, and B. A. Draper. Gesture recognition: Focus on the hands. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5235–5244, 2018.

[142] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 369–376, 2006.

[143] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[144] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*, pages 21–37, 2016.

[145] G. Gkioxari and J. Malik. Finding action tubes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 759–768, 2015.

[146] X. Peng and C. Schmid. Multi-region two-stream r-cnn for action detection. In *Proceedings of the European Conference on Computer Vision*, pages 744–759, 2016.

[147] R. Hou, C. Chen, and M. Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5822–5831, 2017.

[148] S. Saha, G. Singh, M. Sapienza, P. Torr, and F. Cuzzolin. Deep learning for detecting multiple space-time action tubes in videos. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 58.1–58.13. BMVA Press, September 2016.

[149] G. Singh, S. Saha, M. Sapienza, P. H. Torr, and F. Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3637–3646, 2017.

[150] K. Duarte, Y. Rawat, and M. Shah. Videocapsulenet: A simplified network for action detection. In *Advances in Neural Information Processing Systems*, pages 7610–7619, 2018.

[151] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[152] A. Ephrat, I. Mosseri, O. Lang, T. Dekel, K. Wilson, A. Hassidim, W. T. Freeman, and M. Rubinstein. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *ACM Trans. Graph.*, July 2018.

[153] J. L. Alcázar, F. Caba, A. K. Thabet, and B. Ghanem. Maas: Multi-modal assignation for active speaker detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 265–274, 2021.

[154] J. L. Alcázar, F. Caba, L. Mai, F. Perazzi, J.-Y. Lee, P. Arbeláez, and B. Ghanem. Active speakers in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12465–12474, 2020.

[155] S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6964–6968, 2014.

[156] J. Lee, J. Park, T. Kim, and J. Nam. Raw waveform-based audio classification using sample-level cnn architectures. In *Advances in Neural Information Processing Systems Workshops*, 2017.

[157] I. Ariav and I. Cohen. An end-to-end multimodal voice activity detection using wavenet encoder and residual networks. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):265–274, 2019.

[158] M. Ravanelli and Y. Bengio. Speaker recognition from raw waveform with sincnet. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 1021–1028. IEEE, 2018.

[159] L. Kürzinger, N. Lindae, P. Klewitz, and G. Rigoll. Lightweight end-to-end speech recognition from raw audio data using sinc-convolutions. *Proceedings of Interspeech*, pages 1659–1663, 2020.

[160] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll. Small-footprint keyword spotting on raw audio data with sinc-convolutions. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7454–7458, 2020.

[161] C.-L. Liu, S.-W. Fu, Y.-J. Li, J.-W. Huang, H.-M. Wang, and Y. Tsao. Multichannel speech enhancement by raw waveform-mapping using fully convolutional networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:1888–1900, 2020.

[162] Q. Miao, Y. Li, W. Ouyang, Z. Ma, X. Xu, W. Shi, X. Cao, Z. Liu, X. Chai, Z. Liu, et al. Multimodal gesture recognition based on the resc3d network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3047–3055, 2017.

[163] Q. Wang, H. Muckenhirn, K. Wilson, P. Sridhar, Z. Wu, J. R. Hershey, R. A. Saurous, R. J. Weiss, Y. Jia, and I. L. Moreno. Voicefilter: Targeted voice separation by speaker-conditioned spectrogram masking. *Proceedings of Interspeech*, pages 2728–2732, 2019.

[164] P. Chakravarty, S. Mirzaei, T. Tuytelaars, and H. Van hamme. Who's speaking? audio-supervised classification of active speakers in video. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 87–90, 2015.

[165] R. Cutler and L. Davis. Look who's talking: Speaker detection using video and audio correlation. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No. 00TH8532)*, volume 3, pages 1589–1592. IEEE, 2000.

[166] T. Darrell, J. W. Fisher, and P. Viola. Audio-visual segmentation and "the cocktail party effect". In *International Conference on Multimodal Interfaces*, pages 32–40. Springer, 2000.

[167] J. Roth, S. Chaudhuri, O. Klejch, R. Marvin, A. Gallagher, L. Kaver, S. Ramaswamy, A. Stopczynski, C. Schmid, Z. Xi, et al. Ava active speaker: An audio-visual dataset for active speaker detection. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4492–4496, 2020.

[168] J. S. Chung. Naver at activitynet challenge 2019–task b active speaker detection (ava). *arXiv preprint arXiv:1906.10555*, 2019.

[169] Y.-H. Zhang, J. Xiao, S. Yang, and S. Shan. Multi-task learning for audio-visual active speaker detection.

[170] J. Wan, Y. Zhao, S. Zhou, I. Guyon, S. Escalera, and S. Z. Li. Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 56–64, 2016.

[171] Y. Zhang, C. Cao, J. Cheng, and H. Lu. Egogesture: a new dataset and benchmark for egocentric hand gesture recognition. *IEEE Transactions on Multimedia*, 20(5):1038–1050, 2018.

[172] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, et al. The" something

something" video database for learning and evaluating visual common sense. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5842–5850, 2017.

[173] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2556–2563, 2011.

[174] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[175] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[176] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision*, pages 116–131, 2018.

[177] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[178] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, pages 448–456, 2015.

[179] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems*, pages 4967–4976, 2017.

[180] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[181] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[182] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[183] M. Zolfaghari, K. Singh, and T. Brox. Eco: Efficient convolutional network for online video understanding. In *Proceedings of the European Conference on Computer Vision*, pages 695–712, 2018.

[184] G. Varol, I. Laptev, and C. Schmid. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2017.

[185] V. Kantorov and I. Laptev. Efficient feature extraction, encoding and classification for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2593–2600, 2014.

[186] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *Proceedings of the European Conference on Computer Vision*, pages 20–36, 2016.

[187] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[188] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[189] P. Wang, W. Li, Z. Gao, Y. Zhang, C. Tang, and P. Ogunbona. Scene flow to action map: A new representation for rgb-d based action recognition with convolutional neural networks. In *Proc. Comput. Vis. Pattern Recognit.*, pages 1–10, 2017.

[190] P. Wang, W. Li, S. Liu, Z. Gao, C. Tang, and P. Ogunbona. Large-scale isolated gesture recognition using convolutional neural networks. In *Proceedings of the International Conference on Pattern Recognition*, pages 7–12, 2016.

[191] G. Zhu, L. Zhang, L. Mei, J. Shao, J. Song, and P. Shen. Large-scale isolated gesture recognition using pyramidal 3d convolutional networks. In *Proceedings of the International Conference on Pattern Recognition*, pages 19–24, 2016.

[192] J. Duan, J. Wan, S. Zhou, X. Guo, S. Li, et al. A unified framework for multi-modal isolated gesture recognition. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM),(Accept)*, 3, 2017.

[193] Y. Li, Q. Miao, K. Tian, Y. Fan, X. Xu, R. Li, and J. Song. Large-scale gesture recognition with a fusion of rgb-d data based on the c3d model. In *Proceedings of the International Conference on Pattern Recognition*, pages 25–30, 2016.

[194] G. Zhu, L. Zhang, P. Shen, and J. Song. Multimodal gesture recognition using 3-d convolution and convolutional lstm. *IEEE Access*, 5:4517–4524, 2017.

[195] L. Zhang, G. Zhu, P. Shen, J. Song, S. A. Shah, and M. Bennamoun. Learning spatiotemporal features using 3dcnn and convolutional lstm for gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3120–3128, 2017.

[196] E. Ohn-Bar and M. M. Trivedi. Hand gesture recognition in real time for automotive interfaces: A multimodal vision-based approach and evaluations. *IEEE transactions on intelligent transportation systems*, 15(6):2368–2377, 2014.

[197] H. Wang, D. Oneata, J. Verbeek, and C. Schmid. A robust and efficient video representation for action recognition. *International Journal of Computer Vision*, 119(3):219–238, 2016.

[198] R. Qian, T. Meng, B. Gong, M.-H. Yang, H. Wang, S. Belongie, and Y. Cui. Spatiotemporal contrastive video representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6964–6974, 2021.

[199] C.-F. R. Chen, R. Panda, K. Ramakrishnan, R. Feris, J. Cohn, A. Oliva, and Q. Fan. Deep analysis of cnn-based spatio-temporal representations for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6165–6175, 2021.

[200] W. Wu, D. He, T. Lin, F. Li, C. Gan, and E. Ding. Mvfnet: Multi-view fusion network for efficient video recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2943–2951, 2021.

[201] D. Kondratyuk, L. Yuan, Y. Li, L. Zhang, M. Tan, M. Brown, and B. Gong. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 16020–16030, 2021.

[202] C.-Y. Wu, C. Feichtenhofer, H. Fan, K. He, P. Krahenbuhl, and R. Girshick. Long-term feature banks for detailed video understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 284–293, 2019.

[203] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang. Residual attention network for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2017.

[204] L. Chen, H. Zhang, J. Xiao, L. Nie, J. Shao, W. Liu, and T.-S. Chua. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5659–5667, 2017.

[205] S. Woo, J. Park, J.-Y. Lee, and I. So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision*, pages 3–19, 2018.

[206] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu. Dual attention network for scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3146–3154, 2019.

[207] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[208] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[209] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7263–7271, 2017.

[210] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. *Journal of Vision*, 16(12):326–326, 2016.

[211] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[212] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.

[213] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, pages 740–755, 2014.

[214] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[215] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016.

[216] V. Kalogeiton, P. Weinzaepfel, V. Ferrari, and C. Schmid. Action tubelet detector for spatio-temporal action localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4405–4413, 2017.

[217] J. Wei, H. Wang, Y. Yi, Q. Li, and D. Huang. P3d-ctn: Pseudo-3d convolutional tube network for spatio-temporal action detection in videos. In *Proceedings of the IEEE International Conference on Image Processing*, pages 300–304, 2019.

[218] G. Singh, S. Saha, and F. Cuzzolin. Predicting action tubes. In *Proceedings of the European Conference on Computer Vision Workshops*, 2018.

[219] E. H. P. Alwando, Y.-T. Chen, and W.-H. Fang. Cnn-based multiple path search for action tube detection in videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[220] X. Yang, X. Yang, M.-Y. Liu, F. Xiao, L. S. Davis, and J. Kautz. Step: Spatio-temporal progressive learning for video action detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 264–272, 2019.

[221] C. Sun, A. Shrivastava, C. Vondrick, K. Murphy, R. Sukthankar, and C. Schmid. Actor-centric relation network. In *Proceedings of the European Conference on Computer Vision*, pages 318–334, 2018.

[222] D. Li, Z. Qiu, Q. Dai, T. Yao, and T. Mei. Recurrent tubelet proposal and recognition networks for action detection. In *Proceedings of the European Conference on Computer Vision*, pages 303–318, 2018.

[223] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*, 2015.

[224] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[225] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri. Convnet architecture search for spatiotemporal feature learning. *arXiv preprint arXiv:1708.05038*, 2017.

[226] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, pages 525–542, 2016.

[227] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[228] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Proceedings of the International Conference on Learning Representations*, 2016.

[229] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[230] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[231] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the International Conference on Learning Representations*, 2017.

[232] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.

[233] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *Proceedings of the International Conference on Learning Representations*, 2015.

[234] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*, 2014.

[235] J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[236] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[237] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[238] K. S. Abhishek, L. C. F. Qubeley, and D. Ho. Glove-based hand gesture recognition sign language translator using capacitive touch sensor. In *2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 334–337. IEEE, 2016.

[239] R. Wen, L. Yang, C.-K. Chui, K.-B. Lim, and S. Chang. *Intraoperative visual guidance and control interface for augmented reality robotic surgery*. IEEE, 2010.

[240] D. McNeill and E. Levy. *Conceptual representations in language activity and gesture*. ERIC Clearinghouse Columbus, 1980.

[241] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Advances in Neural Information Processing Systems Workshops*, 2014.

[242] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3435–3444, 2019.

[243] S. Li, S. Bak, P. Carr, and X. Wang. Diversity regularized spatiotemporal attention for video-based person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 369–378, 2018.

[244] M. Li, X. Zhu, and S. Gong. Unsupervised tracklet person re-identification. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2019.

[245] C. Song, Y. Huang, W. Ouyang, and L. Wang. Mask-guided contrastive attention model for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1179–1188, 2018.

[246] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[247] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang, and Q. Tian. Mars: A video benchmark for large-scale person re-identification. In *Proceedings of the European Conference on Computer Vision*, 2016.

[248] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 212–220, 2017.

[249] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018.

[250] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

[251] J. Yang, P. Ren, D. Zhang, D. Chen, F. Wen, H. Li, and G. Hua. Neural aggregation network for video face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4362–4371, 2017.

[252] W. Xie and A. Zisserman. Multicolumn networks for face recognition. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, page 111. BMVA Press, 2018.

[253] W. Xie, L. Shen, and A. Zisserman. Comparator networks. In *Proceedings of the European Conference on Computer Vision*, pages 782–797, 2018.

[254] Y. Zhong, R. Arandjelović, and A. Zisserman. Ghostvlad for set-based face recognition. In *Asian Conference on Computer Vision*, pages 35–50. Springer, 2018.

[255] J. S. Chung, A. Nagrani, and A. Zisserman. Voxceleb2: Deep speaker recognition. In *Proceedings of Interspeech*, 2018.

[256] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 529–534, 2011.

[257] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[258] A. Owens and A. A. Efros. Audio-visual scene analysis with self-supervised multisensory features. In *Proceedings of the European Conference on Computer Vision*, pages 631–648, 2018.

[259] B. Korbar, D. Tran, and L. Torresani. Cooperative learning of audio and video models from self-supervised synchronization. In *Advances in Neural Information Processing Systems*, pages 7774–7785, 2018.

[260] A. Rouditchenko, H. Zhao, C. Gan, J. McDermott, and A. Torralba. Self-supervised audio-visual co-segmentation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2357–2361, 2019.

[261] H. Zhao, C. Gan, A. Rouditchenko, C. Vondrick, J. McDermott, and A. Torralba. The sound of pixels. In *Proceedings of the European Conference on Computer Vision*, pages 570–586, 2018.

[262] T. Afouras, A. Owens, J. S. Chung, and A. Zisserman. Self-supervised learning of audio-visual objects from video. In *Proceedings of the European Conference on Computer Vision*, pages 208–224, 2020.

[263] C. Gan, D. Huang, H. Zhao, J. B. Tenenbaum, and A. Torralba. Music gesture for visual sound separation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10478–10487, 2020.

[264] Z. Wu, Y.-G. Jiang, X. Wang, H. Ye, and X. Xue. Multi-stream multi-class fusion of deep networks for video classification. In *Proceedings of the ACM international conference on Multimedia*, pages 791–800, 2016.

[265] F. Xiao, Y. J. Lee, K. Grauman, J. Malik, and C. Feichtenhofer. Audiovisual slowfast networks for video recognition. *arXiv preprint arXiv:2001.08740*, 2020.

[266] S. Ebrahimi Kahou, V. Michalski, K. Konda, R. Memisevic, and C. Pal. Recurrent neural networks for emotion recognition in video. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 467–474, 2015.

[267] F. Noroozi, M. Marjanovic, A. Njegus, S. Escalera, and G. Anbarjafari. Audio-visual emotion recognition in video clips. *IEEE Transactions on Affective Computing*, 10(1):60–75, 2017.

[268] D. Hu, X. Li, et al. Temporal multimodal learning in audiovisual speech recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3574–3582, 2016.

[269] S. Petridis, T. Stafylakis, P. Ma, F. Cai, G. Tzimiropoulos, and M. Pantic. End-to-end audiovisual speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6548–6552, 2018.

[270] F. Tao and C. Busso. Aligning audiovisual features for audiovisual speech recognition. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018.

[271] I. D. Gebru, S. Ba, X. Li, and R. Horaud. Audio-visual speaker diarization based on spatiotemporal bayesian fusion. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 40(5):1086–1099, 2017.

[272] A. Nagrani, C. Sun, D. Ross, R. Sukthankar, C. Schmid, and A. Zisserman. Speech2action: Cross-modal supervision for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10317–10326, 2020.

[273] R. Gao, T.-H. Oh, K. Grauman, and L. Torresani. Listen to look: Action recognition by previewing audio. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10457–10467, 2020.

[274] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 31–35, 2016.

[275] D. Stoller, S. Ewert, and S. Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pages 334–340, 2018.

[276] N. Zeghidour, N. Usunier, I. Kokkinos, T. Schaiz, G. Synnaeve, and E. Dupoux. Learning filterbanks from raw speech for phone recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5509–5513, 2018.

[277] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[278] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[279] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[280] R. Sharma, K. Somandepalli, and S. Narayanan. Crossmodal learning for audio-visual speech event localization. *arXiv preprint arXiv:2003.04358*, 2020.

[281] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 109–117, 2017.

[282] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[283] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3234–3243, 2016.

[284] J. Tremblay, T. To, and S. Birchfield. Falling things: A synthetic dataset for 3d object detection and pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2038–2041, 2018.

[285] G. Varol, I. Laptev, C. Schmid, and A. Zisserman. Synthetic humans for action recognition from unseen viewpoints. *International Journal of Computer Vision*, 129(7):2264–2287, 2021.

[286] H. Hwang, C. Jang, G. Park, J. Cho, and I.-J. Kim. Eldersim: A synthetic data generation platform for human action recognition in eldercare applications. *IEEE Access*, 2021.

[287] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[288] J.-L. Nespoulous, P. Perron, and A. R. Lecours. *The biological foundations of gesture: Motor and semiotic aspects*. Psychology Press, 2014.

[289] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer vision and image understanding*, 73(1):82–98, 1999.

[290] T. A. Dingus, F. Guo, S. Lee, J. F. Antin, M. Perez, M. Buchanan-King, and J. Hankey. Driver crash risk factors and prevalence evaluation using naturalistic driving data. *Proceedings of the National Academy of Sciences*, 113(10):2636–2641, 2016.

[291] M. Martin, A. Roitberg, M. Haurilet, M. Horne, S. Reiß, M. Voit, and R. Stiefelhagen. Drive&act: A multi-modal dataset for fine-grained driver behavior recognition in autonomous vehicles. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2801–2810, 2019.

[292] J. D. Ortega, N. Kose, P. Cañas, M.-A. Chao, A. Unnervik, M. Nieto, O. Otaegui, and L. Salgado. Dmd: A large-scale multi-modal driver monitoring dataset for attention and alertness analysis. In *Proceedings of the European Conference on Computer Vision Workshops*, pages 387–405, 2020.

[293] Y. Abouelnaga, H. M. Eraqi, and M. N. Moustafa. Real-time distracted driver posture classification. In *Advances in Neural Information Processing Systems Workshops*, 2018.

[294] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the International Conference on Machine Learning*, pages 1597–1607, 2020.

[295] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

[296] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, 2010.

[297] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[298] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. In *Proceedings of the European Conference on Computer Vision*, pages 776–794, 2020.

[299] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[300] P. Bachman, R. D. Hjelm, and W. Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15535–15545, 2019.

[301] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.