Technische Universität München
TUM School of Computation, Information and Technology

Exploring Hybrid CTC/Attention End-to-End Speech Recognition: Adversarial Robustness, Sinc Convolutions, and CTC Segmentation

Ludwig Kürzinger

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Norbert Hanik

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Gerhard Rigoll
2. Prof. Dr.-Ing. Hermann Ney

Die Dissertation wurde am 15.11.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 09.04.2024 angenommen.

Abstract

End-to-end speech recognition systems are trained in an end-to-end manner. In contrast to other conventional speech recognition systems, they do not require multiple refinement steps during training or rely on an ensemble of models for acoustic classification and language features. Two structural principles are the most popular for end-to-end speech recognition: (1) Neural networks trained with Connectionist Temporal Classification (CTC) loss provide strong temporal alignments. (2) Attention-based structures sequentially generate their output sequence. Hybrid CTC/attention end-to-end speech recognition combines both powerful concepts that exhibit complementary characteristics.

One key assumption about hybrid CTC/attention is that CTC helps to enforce the attention mechanism to sequential alignments. The first part of this work investigates this key assumption on the basis of various hybrid CTC/Attention hyperparameter configurations. Multiple speech recognition models were trained iteratively and evaluated on the TED-LIUM 2 corpus, whereas each set of parameters was chosen using Gaussian Process optimization. In total, the experiment combined data from 70 models and 590 beam search runs. Summarizing the results, hybrid models exhibit better speech recognition performance when compared to attention-only or CTC-only models.

Deep hybrid CTC/attention neural networks are increasingly complex and prone against specially crafted noise, in particular adversarial noise. Therefore, the second part investigates methods to generate audio adversarial examples for hybrid CTC/attention architectures from CTC loss, based on the attention mechanism or as hybrid CTC/attention adversarial examples. The generated adversarial examples are represented in the feature space and can be converted to audio using feature inversion. Experimental results demonstrate an improvement of speech recognition performance and robustness by augmenting training data with adversarial noise. As adversarial noise is partially inaudible, psycho-acoustic compression methods suppress adversarial noise to some extent. This hypothesis is evaluated for MP3-compression using hybrid CTC/attention models trained on the uncompressed VoxForge corpus in terms of character error rate and SNR of the adversarial noise. Results show improved performance on adversarial examples and a decrease of a portion of the adversarial noise, but at the cost of reduced speech recognition performance on regular audio.

Hybrid CTC/attention models are trained in an end-to-end manner but still require

pre-processed input data. The third part of this work describes how the feature extraction is incorporated into the trainable model in form of Sinc convolutions. Sinc convolutions entail a similar structural information as preprocessed audio features but also provide a trainable flexibility such as regular convolutional network layers. A keyword classification experiment on the Google Speech Commands corpus demonstrates these advantages: The Sinc convolutional network exhibits superior performance, even when compared to larger and more complex convolutional networks. A second experiment incorporates the Sinc convolution into the hybrid CTC/attention architecture for end-to-end decoding, which improved speech recognition performance while at the same time reducing model size.

As these deep neural networks grow increasingly complex, they also require more training data, represented in the form of audio segments annotated with transcription. The final part of this work describes CTC segmentation, a dynamic programming algorithm that aligns transcription text to audio using CTC-based network activations. We use this algorithm to align sentences from audiobooks and evaluate models with different training dataset compositions on the German speech corpus TuDa-DE. Speech recognition performance increases in relation to more training data with more accurate alignments. In comparison to other forced alignment algorithms, CTC segmentation skips unrelated speech segments and yields a confidence score that can be used to filter out utterances with mismatching transcriptions. This CTC-based data cleansing can be an essential building block for automated data acquisition, which is demonstrated with the example of Jtubespeech, the largest publicly available Japanese speech dataset.

Zusammenfassung

Ende-zu-Ende-Spracherkennungssysteme werden mit akustischen Merkmalsdaten direkt auf den Ausgabetext trainiert. Im Gegensatz zu anderen konventionellen Spracherkennungssystemen benötigen sie nicht mehrere Verfeinerungsschritte während des Trainings oder stützen sich auf ein Ensemble von Modellen für die akustische Klassifizierung und Sprachmerkmale. Zwei Techniken sind für die Ende-zu-Ende-Spracherkennung am weitesten verbreitet: (1) Neuronale Netze, die mithilfe von der Connectionist Temporal Classification (CTC) als Zielfunktion trainiert werden, generieren zeitlich gebundene Klassenwahrscheinlichkeiten. (2) Attention-basierte Netze erzeugen den Ausgabetext sequentiell. Die hybride CTC/Attention Ende-zu-Ende-Spracherkennung kombiniert beide Konzepte, welche sich gegenseitig ergänzen.

Eine der wichtigsten Annahmen über die hybride CTC/Attention Architektur ist, dass CTC den Fokus des Attention-Mechanismus lenkt. Der erste Teil dieser Arbeit untersucht diese Annahme auf der Grundlage verschiedener hybrider CTC/Attention-Parameterkonfigurationen. In einem iterativen Verfahren wurden mehrere Modelle auf dem TED-LIUM 2-Sprachkorpus evaluiert, wobei jeder Parametersatz mittels Gaußscher Prozessoptimierung ausgewählt wurde. Insgesamt wurden in dem Experiment Ergebnisse von 70 Modellen und 590 Dekodiervorgängen kombiniert. Die Ergebnisse geben Hinweise auf bewährte Parameterkombinationen und zeigen, dass hybride Modelle im Vergleich zu reinen Attention- oder CTC-Modellen eine bessere Spracherkennungsleistung aufweisen.

Tiefe hybride neuronale CTC/Attention Netze werden immer komplexer und sind anfällig für speziell erzeugtes Rauschen, insbesondere für Adversarial Noise. Daher werden im zweiten Teil Methoden zur Erzeugung von Audio-Adversarial Examples für hybride CTC/Attention Architekturen untersucht, als CTC-basierte, Attention-basierte oder als hybride CTC/Attention Adversarial Examples. Die generierten Gegenbeispiele werden als Merkmalsdaten dargestellt und können mittels Feature Inversion in reguläre Audiodaten umgewandelt werden. Experimentelle Ergebnisse zeigen eine Verbesserung der Spracherkennungsleistung und der Robustheit durch die Anreicherung der Trainingsdaten mit Adversarial Noise. Da Adversarial Noise im Audiobereich teilweise unhörbar ist, können psychoakustische Kompressionsverfahren dieses Rauschen bis zu einem gewissen Grad unterdrücken. Diese Hypothese wird für die MP3-Kompression mit hybriden CTC/Attention-Modellen, die auf dem unkomprimierten VoxForge-Sprachdatenschatz

trainiert wurden, in Bezug auf die Spracherkennungsleistung und das Signal-Rausch-Verhältnis des erzeugten Rauschens evaluiert. Die Ergebnisse zeigen eine verbesserte Leistung bei Adversarial Examples und eine Verringerung von Adversarial Noise, jedoch um den Preis einer geringeren Spracherkennungsleistung bei regulären Sprachaufnahmen.

Hybride CTC/Attention Netze werden zwar Ende-zu-Ende trainiert, benötigen aber immer noch Merkmalsdaten aus einem zusätzlichen Vorverarbeitungsschritt. Der dritte Teil dieser Arbeit beschreibt, wie diese Merkmalsextraktion in Form von Sinc-Faltungen in das trainierbare Modell integriert wird. Sinc-Faltungen beinhalten eine ähnliche strukturelle Information wie vorverarbeitete Audio-Merkmale, bieten aber auch eine trainierbare Flexibilität wie reguläre Faltungen. Ein Experiment zur Klassifizierung von Schlüsselwörtern auf dem Google Speech Commands Datensatz demonstriert die Vorteile dieses Ansatzes: Das Netzwerk mit Sinc-Faltungen zeigt eine überlegene Leistung, selbst im Vergleich zu größeren und komplexeren Faltungsnetzwerken. In einem zweiten Experiment wird die Sinc-Faltung in die hybride CTC/Attention-Architektur integriert, was die Spracherkennungsleistung verbessert und gleichzeitig die Modellgröße reduziert.

Da diese tiefen neuronalen Netze immer komplexer werden, benötigen sie auch mehr Trainingsdaten in Form von Audiosegmenten, die mit Transkriptionen versehen sind. Der letzte Teil dieser Arbeit beschreibt CTC Segmentierung, einen Algorithmus, der Textabschnitte mithilfe von einem CTC-basierten Netz an Audiosegmenten ausrichtet. Wir verwenden diesen Algorithmus zur Ausrichtung von Sätzen aus Hörbüchern und evaluieren Modelle mit verschiedenen Datensatz-Zusammenstellungen auf dem deutschen Sprachkorpus TuDa-DE. Die Spracherkennungsleistung steigt mit zunehmender Anzahl von guten Trainingsdaten und genaueren Abschnittszeiten. Im Vergleich zu anderen Algorithmen kann CTC Segmentierung leere Audiosegmente überspringen und liefert einen Konfidenzwert, der zum Herausfiltern von Audioabschnitten mit nicht übereinstimmenden Transkriptionen verwendet werden kann. Diese CTC-basierte Datenbereinigung kann ein wesentlicher Baustein für die automatische Datenerfassung sein; dies wird am Beispiel von Jtubespeech, dem größten öffentlich verfügbaren japanischen Sprachdatensatz, demonstriert.

Contents

List of Acronyms

ASR	Automatic speech recognition
BLSTM	Bidirectional long short-term memory
BPE	Byte-pair encoding
CER	Character error rate
CPU	Central processing unit
CTC	Connectionist temporal classification
CNN	Convolutional neural network
DNN	Deep neural network
DNN/HMM	Speech recognition architecture with a deep neural network in combination with hidden markov models
DSConv	Depthwise separable convolution
DTW	Dynamic time warping
EI	Expected improvement
EM	Expectation maximization
EOS	End-of-sentence token
FER	Frame error rate
FGSM	Fast gradient sign method
GDSConv	Grouped depthwise separable convolution
GMM	Gaussian mixture model
GP	Gaussian process
GPU	Graphics processing unit
HMM	Hidden markov model
HTK	Hidden markov model toolkit
KN	Kneser-Ney smoothing
LM	Language model
LSC	Lightweight sinc convolutions
LSTM	Long short-term memory
MFCC	Mel frequency cepstral coefficients
MMU	Memory management unit
MAUS	Munich Automatic Segmentation

NLP	Natural language processing
ReLU	Rectified linear unit
RIR	Room Impulse Response
RNN	Recurrent neural network
RNNLM	Recurrent neural network language model
SGD	Stochastic gradient descent
SNR	Signal to noise ratio
WER	Word error rate

List of Symbols

Generic Notation

l	Generic label sequence index
L	Length of label sequence
N	Generic indicator of number of elements in a sequence
n	Position or index indicator within a sequence
$p(y_l)$	Output probability of attention mechanism for label y at sequence index l
$p(y_t)$	Output probability of CTC network for label y at time index t
R	Raw audio data
r_t	Raw audio sample at time t
S	Generic state sequence
s_t	State at time t
T	Generic length of time domain
T_H	Length of time domain for annotated feature sequence
T_R	Length of time domain for raw audio
T_X	Length of time domain for pre-processed audio features
t	Generic time index
X	Pre-processed audio features
x_t	Pre-processed audio feature at time t
$x_{1:T}$	Sequence of features from x_1 to x_T
Y	General label sequence
$\hat{Y}$	Inferred label sequence
Y'	Partial label sequence
$\overline{Y}$	Ground truth label sequence
y_l	Label at sequence index l
$\hat{y}_l$	Inferred label at sequence index l
y'_l	Partial label at sequence index l
$\overline{y}_l$	Ground truth label at sequence index l

Functions

abs	Absolute value function
arg max	Function that selects the argument with the maximum value
arg min	Function that selects the argument with the minimum value
att	Attention
BLSTM	Neural network block for Bidirectional Long Short-Term Memory
BLSTMP	Neural network block for Bidirectional Long Short-Term Memory with Projection
$\mathcal{L}_{\text{CE}}$	Cross-entropy loss
Conv	Convolution
DConvBlock	Depthwise Convolutional Block
dec	Decoder
enc	Encoder
FFN	Feedforward Neural Network
LayerNorm	Layer Normalization
LogCompression	Log Compression
$\mathcal{L}$	Loss function
LSC	Lightweight Sinc convolutions, a preencoder that integrates Sinc convolutions
LSTM	Long Short-Term Memory
MHA	Multi-Head Attention
PE	Positional Encoding
rect	Rect function
sgn	Sign function
sinc	Sinc function
SincBlock	Sinc Convolutional Block
VGG	VGGnet neural network block

Neural Networks

$\cdot$	Dot product in convolution
$*$	Convolution operation
acc	Accuracy of predictions
$\arg\min_\theta \mathcal{L}(\theta)$	Function that selects θ to minimize the loss
b	Bias vector in a linear layer
$\text{Conv}_{k,s,p}(f, x, i)$	Convolution operation at position i with filter f, input x, kernel width k, stride s, and padding p
D	Number of deletions in error rate calculation
η_i	Learning rate at iteration i, for stochastic gradent descent
f	Filter or kernel in convolution operation

$f(x_n; \theta)$	Neural network function with parameters θ
$g_\theta(x)$	Parameterized function of a neural network
$g_\theta(x)$	Output of neural network for input x
I	Number of insertions in error rate calculation
i	Position or index indicator
k	Width or size of the convolutional filter
$\mathcal{L}_{\text{CE}}$	Cross-entropy loss
$\text{LeakyReLU}_\alpha(x)$	Leaky Rectified Linear Unit function, with α as small positive constant, usually 0.01
$\text{Linear}'(x)$	Linear layer transformation without bias
$\text{Linear}(x)$	Linear layer transformation
$\mathcal{L}_n$	Individual loss for training sample n
$\mathcal{L}(\theta)$	Objective function to minimize during training
$\mathbb{R}^C$	Class space of dimension C
$\mathbb{R}^D$	Feature space of dimension D
$\mathcal{B}$	Mini-batch, a small randomly chosen subset of the dataset
$\mathcal{D}$	Dataset containing N training samples
$\mathcal{U}$	Token dictionary
$\max(a, x)$	Maximum of a and input x
p	Padding in the convolutional layer
$p(y_n \vert y_{1:n-1})$	Conditional probability of y_n given previous tokens
$\text{PP}(Y)$	Perplexity of sequence Y
$p(S \vert Y)$	Probability of state sequence S given label sequence Y
$p_\theta(c \vert x)$	Posterior probability of class c given input x
$p(Y)$	Probability of sequence Y
$p(Y \vert X)$	Posterior distribution of Y given X
$\text{ReLU}(x)$	Rectified Linear Unit function
S	Number of substitutions in error rate calculation
s	Stride of the convolutional filter
$\sigma(x)$	Sigmoid function
$\text{Softmax}(x)$	Softmax function
$\tanh(x)$	Hyperbolic tangent function
θ	Parameters of the neural network
θ_i	Parameters at iteration i, for stochastic gradient descent
θ^*	Optimal parameters that minimize the loss
W	Weight matrix in a linear layer
(x_n, y_n)	Training sample consisting of input x_n and output y_n
y_i	Output at position i

Speech Recognition

$\alpha(Y', X)$	CTC score

$a_{l,t}$	Attention weight at output step l and position t
att_{dot}	Scaled dot product attention
$\text{att}_{\text{dot, masked}}(Q, K, V)$	Masked dot product attention mechanism
att_{loc}	Location-aware attention mechanism
C_h	Output of attention head h
c_l	Context vector at output step l (location-aware attention)
d_k	Dimensionality of features
g	Trainable vector for scalar product (location-aware attention)
H	High-level representation of the input sequence
h_l	Vector in the annotated sequence at position l
K	Key matrix in the attention mechanism (Transformer)
K	Trainable convolutional kernel (location-aware attention)
$L_{Y'}$	Length of the prefix Y'
$M_{i,j}$	Masking matrix used in masked multi-head attention
n_k	Number of key elements in the sequence
n_q	Number of query elements in the sequence
p_{ctc}	Conditional probability of the partial hypothesis
PE	Positional encoding function (Transformer)
$\hat{Y}$	Most likely sequence of letters in the hypothesis
Q	Query matrix (Transformer)
S_h	Set of all label sequences that start with the prefix h
V	Value matrix (Transformer)
W^{head}	Weight matrix for combining heads
W_h^k	Weight matrix for keys for head h
W_h^q	Weight matrix for queries for head h
W_h^v	Weight matrix for values for head h
X_i'	Normalized input matrix for self-attention
X_{sub}	Sub-matrix of input features (Transformer)
Y_{emb}	Embedded representation of the token IDs
Y'	Prefix of the full utterance hypothesis
Z_j	Intermediate results from masked multi-head attention
Z_0	Input to the first decoder layer, obtained by concatenating the embedding matrix with positional encoding (Transformer)

GP Optimization

β	Language model weight
κ	CTC weight parameter during training
λ	CTC weight during decoding

μ	Mean function of the Gaussian process
ν	Parameter in the Matérn kernel
D	Set of previous observations
$\mathbb{E}$	Expectation operator
f_{EI}	Expected Improvement acquisition function
f_{GP}	Gaussian process modeling the unknown function f
f_{min}	Minimum observed value of f so far
f	Unknown function representing evaluation metric (word or character error rate)
k	Kernel function of the Gaussian process
K_ν	Modified Bessel function
l	Parameter in the Matérn kernel l
N_{att}	Number of attention channels
N_{dec}	Number of decoder layers
N_{enc}	Number of encoder layers
N_{K}	Number of attention filters
$r^{(n)}$	Euclidean distance between parameter sets
W_{att}	Width of attention layer
W_{dec}	Width of decoder layers
W_{enc}	Width of encoder layers
$X^{(n+1)}$	Next optimal point to be evaluated
$\mathcal{X}$	Parameter space
X	Discrete parameter set
$Y^{(i)}$	Target value corresponding to $X^{(i)}$

Adversarial ML

β	Language model weight during decoding
γ	Starting index of the static window for localized adversarial perturbation
$\delta(x)$	Adversarial noise function
ϵ	Factor controlling intensity of adversarial noise
θ	Parameters of the model
κ	CTC weight parameter
λ	CTC weight during decoding
ν	Stride of the moving attention window
ξ	Weighting parameter of the adversarial noise in adversarial training
∇_{x_t}	Gradient with respect to x_t
$\mathcal{L}_{\text{CE}}(X, y; \theta)$	Objective function for cross-entropy
L	Length of reconstructed label sequence
l_w	Width of the window for localized adversarial perturbation
P_a	Signal power of audio signal

$S_b(f)$	Spectral power density of brown noise at frequency f
SNR_{dB}	Signal-to-Noise Ratio in decibels
$S_p(f)$	Spectral power density of pink noise at frequency f
$\hat{x}_t$	Adversarial example at time t

Sinc Convolutions

ϵ	Constant parameter for Adadelta optimizer
κ	CTC weight during training
ρ	Decay rate parameter for Adadelta optimizer
f_1	Lower cutoff frequency of the band-pass filter
f_2	Upper cutoff frequency of the band-pass filter
$G[f, f_1, f_2]$	Fourier transform of the band-pass filter
$g[n, f_1, f_2]$	Sinc convolutional kernel coefficients
$g'[n, f_1, f_2]$	Windowed Sinc convolutional kernel coefficients
L	Length of the convolutional kernel in the Sinc convolution
rect	Rectangular function
$w[n]$	Hamming window function
$w_{\text{Hann}}[n]$	Hann window function
w_1	First learnable parameter related to cutoff frequencies
w_2	Second learnable parameter related to cutoff frequencies

CTC Segmentation

θ	CTC segmentation confidence score threshold for selection of utterances
a_t	Token index that audio frame t is aligned to
c_j	Token at index j
$k_{t,j}$	Maximum joint probability of segment alignments at time t and token index j
L_{ctc}	Length of the CTC output
L_{token}	Length of the aligned token sequence
m_j	Mean value of alignment probabilities within part j
p_{stay}	Probability of staying in the current state
$p_{\text{transition}}$	Probability of transitioning to a new state
ρ_t	Alignment probability at time t
s_{seg}	Confidence score for an utterance

List of Figures

List of Tables

1

Introduction

This thesis contributes to the field of human-machine interaction, in particular on modern methods to transcribe speech to text. In the field of human-machine interaction, automatic speech recognition (ASR) plays the critical role in transcribing speech to text.

ASR is a complex task due to the inherent variability and ambiguity in spoken language. While written language is a structured cultural creation, spoken language often mirrors fluid thought processes, communicates abstract concepts, and conveys nuanced ideas. This complexity is widely acknowledged in related scientific fields. For instance, Ferdinand de Saussure, a prominent linguist, proposed the notion of the arbitrariness of spoken language, asserting that there is no predetermined rule or direct connection between a written word, its pronunciation, and its meaning [50].

Speech is a natural and intuitive means of communication, and ASR aims to convert spoken language into written text through complex machine learning techniques. ASR systems have been under continuous development for decades, with numerous applications ranging from transcription services and voice-controlled devices to language translation and communication aids. As technology advances, there is a growing demand for more accurate and efficient speech recognition systems.

Classical ASR methods often employ a combination of Hidden Markov Models (HMM) and Deep Neural Networks (DNN), resulting in hybrid DNN/HMM systems. However, recent research has demonstrated the potential of large end-to-end neural network models to outperform these conventional approaches. Consequently, the field of ASR is currently experiencing a transition towards end-to-end models.

This dissertation contributes to this transition by proposing replacements and supplements for functionalities of end-to-end models that were previously prevalent among conventional DNN/HMM ASR systems. It investigates the domain of end-to-end speech recognition systems, concentrating on the examination of hybrid Connectionist Temporal Classification (CTC)/attention models and their potential for enhanced performance and robustness.

1.1 Automatic Speech Recognition

The development of ASR has involved addressing two key challenges for traditional hybrid DNN/HMM models: acoustic modeling, which translates the waveform of a spoken utterance into an intermediate representation like phonemes; and language modeling, which generates the corresponding written text based on linguistic rules and grammar.

Recent advancements in ASR have led to the emergence of end-to-end models, which are end-to-end trainable and combine both acoustic and language modeling tasks into a single, unified framework. These modern approaches may even eliminate the need for a distinct language model, as they inherently capture the language-function within their architecture. Throughout this chapter, we will explore traditional approaches to acoustic and language modeling, as well as hybrid DNN/HMM systems, and discuss how the paradigm shift towards end-to-end models has transformed the ASR landscape.

Chapter 2 introduces the terms of speech recognition in more detail and gives an introduction to readers of related fields.

1.2 Hybrid CTC/Attention Speech Recognition

The combination of CTC and attention mechanisms holds considerable promise [196]. Traditional ASR systems often resort to hand-crafted, linguistically informed modules, leading to potential errors and inefficiencies. CTC and attention-based methods, in their unique ways, address these complexities by proposing simplified, unified models.

CTC models offer efficient computation of monotonic alignments, crucial for ASR given the sequential nature of speech. These models adhere to a temporal sequence mapping, ensuring alignment with the progression in speech signals. However, their application often requires additional language models and graph-based decoding, unless large-scale training data sets are available.

Conversely, attention-based models provide a distinct advantage by directly estimating posteriors without necessitating conditional independence assumptions. These models can flexibly form temporal alignments, though this flexibility occasionally results in irregular alignments, especially in ASR scenarios where input and output sequence lengths can vary significantly.

Thus, in an end-to-end ASR context, the fusion of CTC and attention mechanisms seems a particularly potent approach. By utilizing the monotonic alignment from CTC and the direct posterior estimation from the attention mechanism, the proposed hybrid CTC/attention model aims to circumvent the limitations inherent in each standalone approach. The idea is to capitalize on the strengths of both methods, employing CTC-based alignment as a regularization during training and jointly decoding with both attention-based and CTC scores. This design is anticipated to mitigate irregular alignments and enhance the robustness and efficacy of end-to-end ASR systems.

1.3 Scientific Goals of this Work

The transition to end-to-end models demands novel approaches that were solved in the domain of HMM-based systems, but were not solved previously for end-to-end models. Within this context, this work investigates new aspects for of end-to-end models, such as model parameter spaces, feature extraction, adversarial noise and audio-to-text alignment.

1.3.1 Exploration of Hybrid CTC/Attention ASR Models

Hybrid CTC/attention models integrate both the temporal alignment capabilities of CTC and the sequential output generation of attention mechanisms. A critical assumption underpinning this hybrid model is the presumed role of CTC in enforcing the attention mechanism towards sequential alignments. In challenging this assumption, the study presented in Chapter 3 applied Gaussian process optimization to a multitude of hybrid CTC/attention network parameters and language model weights. This approach not only tests the underlying hypothesis but also allows the derivation of general recommendations for model configurations.

This investigation was carried out by training and evaluating 70 hybrid CTC/attention networks with various parameter configurations on the TED-LIUM 2 test set, coupled with 590 beam search runs with a RNNLM. The performance of these models was evaluated on the basis of CER, WER and attention accuracy.

Contrary to prevailing assumptions, the study provides evidence-based argumentation that CTC primarily regularizes the impact of language model feedback in a one-pass beam search, rather than constraining the attention mechanism to sequential alignments. Surprisingly, attention-only models without RNNLM achieved a 22.4% WER, while attention-only decoding combined with an RNNLM significantly underperformed. The best performance was obtained from the combined hybrid CTC/attention model with RNNLM, yielding a 17.6% CER. This research highlights the regularization role of CTC, and identifies specific parameter configurations in combination that can either enhance or hinder performance of language model-supported decoding.

1.3.2 Adversarial Machine Learning

The ascendancy of end-to-end ASR systems has culminated in superior performance in comparison to conventional DNN/HMM models through the utilization of deeper and more complex models. However, these models exhibit increased susceptibility to audio adversarial examples, specialized noise instances crafted to deceive and misguide their processing, yielding misclassifications.

Advancing the understanding and application of adversarial examples, Chapter 4 demonstrates methods to generate audio adversarial examples for hybrid CTC/attention models, in a combination of both CTC and attention techniques into a unified gradient method. This chapter then discusses adversarial training as a method to improve robustness against adversarial noise. Concurrently, a mitigation strategy is explored

through the application of MP3 compression as preprocessing step, employed to reduce the adversarial noise and its impact in audio samples processed by ASR systems.

Experiments were conducted using a hybrid CTC/attention ASR model to generate and apply these new adversarial examples in the training phase, effectively augmenting the robustness of the ASR model. Alongside this, the proposed MP3 compression technique was applied to adversarial examples to assess its capacity to diminish the influence of adversarial noise.

Verification of these methods was undertaken via Character Error Rates (CER) and Signal-to-Noise Ratio (SNR), using a set of ASR models trained on different audio formats and a suite of uncompressed and MP3-compressed adversarial examples reconstructed by feature inversion. Empirical results validate both the effectiveness of the developed audio adversarial examples in strengthening ASR model robustness, and the efficacy of MP3 compression in reducing adversarial noise, as indicated by decreased CER and heightened SNR. However, MP3 compression applied to utterances augmented with regular noise led to an increase in transcription errors, underscoring its specificity in diminishing adversarial noise only.

1.3.3 End-to-End Architecture with Raw Audio Input

The term *end-to-end* in relation to ASR architecture can often imply end-to-end trainability, or it can refer to a simplified structure, distinguishing it from hybrid DNN/HMM systems that necessitate a collection of multiple models for inference and multiple stages of training. This type of architecture, however, still mandates a discrete step for preprocessing audio features, specifically for filter bank features. The elimination of pre-processed frequency-domain features and the pursuit of a fully trainable ASR system becomes the objective of Chapter 5. It accomplishes this by integrating Sinc-Convolutions to extract spectral features directly from raw audio input. As a distinction to previous work, the Sinc convolutions are combined with depthwise separable convolutions and function as an integrated frontend for the ASR system.

This methodology is verified through a keyword spotting architecture. Notably, conventional studies depend on preprocessed features; nevertheless, bypassing feature extraction comes with its own advantages. The keyword spotting task is predominantly used on always-on and battery-operated smart devices, which are bound by hardware resources and power consumption constraints. In this context, classification from raw audio decreases the need for feature extraction, thereby reducing hardware requirements; additionally, low-parameter architectures minimize the quantity of power-consuming memory transfers. As a result, the low-parameter keyword spotting model achieves a commendable accuracy of 96.4% on Google's Speech Commands test set with a mere 62k parameters.

Building on the proof-of-concept end-to-end architecture, this work discusses Lightweight Sinc-Convolutions (LSC). LSC expands on the previously developed end-to-end architectural concept that amalgamates Sinc Convolutions with depthwise separable convolutions and serves as a low-parameter, machine-learnable feature extraction for end-to-end RNN-based ASR systems. Further enhancements, such as data augmentation

in the time-domain using SpecAugment, filter-level improvements, and the application of log-compression as an activation function are also explored. As a result, the model demonstrates a smooth convergence behaviour, achieving a word error rate of 10.7% on the TED-LIUM 2 test dataset. It surpasses the corresponding architecture with regular log-mel filterbank features by an absolute 1.9%, but only constitutes 21% of its model size.

1.3.4 Text-to-Audio Alignment using CTC Segmentation

The recent advancements in end-to-end systems have underscored their potential to surpass traditional hybrid DNN/HMM ASR systems in performance. This evolution is predicated on architectural enhancements and model expansion in terms of depth, parameters, and model capacity. Despite these improvements, these models necessitate an increase in training data to attain comparable performance. The development of large speech corpora is instrumental in improving performance, especially in languages other than English where such resources have remained scarce. However, this process can be complicated by the need for manual segmentation and labeling.

Chapter 6 demonstrates the construction of large and diverse corpora for German and Japanese speech recognition and the utility of segmentation using the posterior probabilities of a CTC network for utterance segmentation. The application of CTC segmentation facilitates bootstrapping of additional training data from unsegmented or unlabeled data, thereby providing an efficient approach to prepare large volumes of speech data for model training.

The discussed methodology for data preparation involves a two-stage approach. To initiate the process, an existing pre-trained hybrid CTC/attention model, specific to the target language and capable of generating CTC output, is utilized. This model is then used to extract utterances from label probabilities obtained from the network trained with CTC to determine segment alignments. Moreover, CTC segmentation streamlines the process of extracting and refining data from YouTube videos and audiobooks, requiring minimal language-specific procedures. It notably improves the accuracy of the transcriptions, which often contain errors, by scoring and correcting discrepancies between the audio and the subtitles. The data was efficiently extracted solely utilizing the output of a CTC-based network, thereby removing the need for alignment tools based on HMMs or Dynamic Time Warping (DTW) that would otherwise necessitate separate hybrid DNN/HMM models.

The resultant corpora amassed over 1700 hours of German speech data and more than 10,000 hours of Japanese speech data from manually subtitled videos. With the German speech data, we trained a hybrid CTC/attention Transformer model that achieved a 12.8% WER on the Tuda-DE test set, surpassing the previous 14.4% WER baseline of conventional hybrid DNN/HMM ASR. Additionally, the Japanese corpus yielded a large-scale ASR benchmark with over 1,300 hours of data. These outcomes underscore the efficacy of CTC segmentation in multilingual corpora construction and its potential in advancing end-to-end ASR systems.

1.4 Mathematical Notation

Before proceeding, it is important to clarify the time indices notation, $t = 1, \cdots, T$. Audio samples are typically sampled at a rate of 16 kHz. After feature extraction, the number of data points in t is reduced, and the neural networks discussed later in this paper employ subsampling techniques, further reducing the data's temporal resolution. Despite being in the same time domain, these indices differ in proportion due to subsampling.

For simplicity and clarity, this thesis uses the index t and length T as generic indicators of time domain. To prevent confusion between similar time domains, we use additional indices. Raw audio data is denoted as $R = \{r_t \in R | t = 1, \cdots, T_R\}$, and pre-processed audio features as $X = \{x_t \in R^D | t = 1, \cdots, T_X\}$, where t indexes the time domain. Thus, T_R and T_X are the lengths of time domain for raw and feature-extracted audio respectively. Similarly, T_H is used for the time domain of the annotated feature sequence.

An equivalent convention also applies to the index notation of sequences $l = 1, \cdots, L$. The general label sequence is annotated as $Y = (y_1, y_2, \ldots, y_L)$. To distinguish the inferred label sequence from the ground truth label sequence when required, we use an overbar to annotate the ground truth sequences as $\overline{Y} = (\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_L)$, and the circumflex accent to denote the inferred sequence as $\hat{Y} = (\hat{y}'_1, \hat{y}_2, \ldots, \hat{y}_L)$. Partial sequences are denoted with an apostrophe as $Y' = (y'_1, y'_2, \ldots, y'_{L'})$.

In concise notation, sequences can also be represented by specifying the range of indices, such as $x_{1:T}$, to denote the sequence $(x_1, x_2, \ldots, x_T)$.

The choice of sequence representation varies with the neural network architecture, highlighting their unique processing characteristics. As discussed in Chapter 2, RNNs operate on data sequentially, leading to the use of x_t to denote sequence data at a specific time step. In contrast, Transformers handle the entire sequence in a single go, representing the full input sequence as X.

Different neural architectures produce outputs in varying domains, which is reflected in their index notation. The output of a CTC network, which is proportionate to the audio duration, is expressed in simplified notation as $p(y_t)$ using the generic index t. Meanwhile, the output of the attention mechanism, being in the sequential domain, is denoted as $p(y_l)$ using the index l.

1.5 Document Structure

The remainder of this thesis is grouped into three parts: Chapter 2 gives an overview over the core concepts of ASR, Chapters 3 is mostly experimental, and Chapters 4, 5 and 6 introduce new methods.

Chapter 2 introduces the ASR system used throughout this thesis, i.e., the hybrid CTC/attention neural network architecture, and its common building blocks. Chapter 3 presents experimental results for parameter exploration and optimization using Gaussian processes. Chapter 4 describes adversarial examples that are constructed to mislead classification, and then are used to train a more robust ASR model. Further experiments also demonstrate how psycho-acoustic MP3 compression reduces adversarial noise. Chap-

ter 6 introduces CTC segmentation which uses the output of a CTC-based network to align utterances in audio files. This new method can be used for corpus construction and data clean-up, as demonstrated on a German and a Japanese dataset. Chapter 5 describes Sinc convolutions as a method how the hybrid CTC/attention system can directly classify from raw audio, instead of relying on pre-processed filterbank features. Chapter 7 concludes this thesis.

2

Fundamental Concepts of Speech Recognition

This chapter explores the core principles of speech recognition, transitioning from basic neural network notations to modern end-to-end methods. It offers insights into the scientific objectives and challenges addressed in this research, highlighting both handcrafted feature extraction and data-driven neural network approaches. This exploration offers an understanding of the scientific challenges that have emerged with the rise of deep end-to-end neural networks in ASR.

2.1 The Speech Recognition Objective

The input for ASR systems consists of spoken sentences. Spoken language is an innate human ability that has developed in conjunction with our auditory system. Statistical modeling of speech mimics the psychoacoustic properties of human hearing [218]. The most effective ASR systems today utilize frequency filters for feature preprocessing, which are grounded in psychoacoustic insights. As one of the five senses, the auditory system is estimated to contribute approximately 10% of perceived information [161].

The output of ASR systems is always text, representing written language. Converting speech into written text involves addressing challenges such as differences in speaking styles such as accents and dialects, out-of-vocabulary words, background noise, and speech disfluencies like hesitations and repetitions. The arbitrariness of spoken language, which lacks a direct link between written words, pronunciation, and meaning, adds to this complexity. ASR systems must also handle the intricate structure and rules of written language that can vary across languages and domains. To effectively model and process speech data, ASR systems rely on sophisticated mathematical frameworks and large amounts of training data to tackle these challenges and generate accurate written text.

In formal terms, ASR maps a raw audio recording of speech $R = \{r_t \in R | t = 1, \cdots, T_R\}$ via an intermediate sequence in feature space $X = \{x_t \in R^D | t = 1, \cdots, T_X\}$ to a ground truth character or token sequence $\overline{Y} = \{\overline{y}_l \in \mathcal{U} | l = 1, \cdots, L\}$ of the token

dictionary $\mathcal{U}$. Bayesian decision theory is employed to mathematically establish an optimization objective to the most likely character sequence, represented as $\hat{Y}$. This objective can be expressed as

$$\hat{Y} = \arg\max_{Y} p(Y|X). \tag{2.1}$$

Thus, the primary challenge in ASR lies in calculating the posterior distribution, $p(Y|X)$.

2.1.1 Measurement of Performance

The quality or performance of speech recognition systems is assessed using two primary metrics, namely the Word Error Rate (WER) and the Character Error Rate (CER). These error rates are calculated based on the normalized Levenshtein Distance, also known as Edit Distance, that quantifies the minimum number of single-word or single-character edits (insertions, deletions, or substitutions) required to change the inferred utterance into the ground truth [116, 130]. Then, error rates are expressed as $(S + D + I)/N$, where S represents the number of substitutions, D denotes deletions, I stands for insertions, and N indicates the number of words in the reference sentence.

The quality of sequence classification or of token-level prediction tasks is evaluated based on accuracy. Accuracy measures the proportion of total correct predictions in a classification problem. It is defined as the proportion of correct predictions $y_i = \overline{y}_i$ divided by the total number of predictions N.

$$\text{acc} = \frac{1}{N} \sum_{i=1}^{N} 1_{(y_i = \hat{y}_i)} \tag{2.2}$$

Attention-based sequence generative models may also be evaluated based on their accuracy, but only during training. Here, accuracy specifically refers to how accurately the model predicts each subsequent token in the sequence, using a portion of the ground truth sequence as input, a technique known as teacher forcing.

In the context of neural network language models, perplexity is a measure of how well a probability model predicts a sample. Given a sequence of N tokens $Y = [y_1, y_2, \cdots, y_N]$, the perplexity $\text{PP}(Y)$ is:

$$\text{PP}(Y) = p(Y)^{-\frac{1}{N}} = \left[\prod_{n=1}^{N} p(y_n | y_1, \cdots, y_{n-1}) \right]^{-\frac{1}{N}} \tag{2.3}$$

Perplexity is computed over the entire dataset and can be interpreted as the average number of choices to continue the word sequence Y at any position n. A lower perplexity indicates that the model is better at predicting the sample.

2.1.2 Feature Extraction

In the initial stage of speech recognition, digital audio is represented as a continuous sequence of numerical samples, obtained from the sound wave of the audio signal.

Typically, the sampling rate is set to 16 kHz, which is necessary to accommodate the significant frequencies of speech signals, which can be up to 8 kHz. According to the Nyquist-Shannon sampling theorem, the sampling rate should be at least twice the highest frequency of interest.

Feature extraction is the process of transforming the sampled audio into a sequence of feature vectors. This process is inspired by models of the human auditory system and aims to mimic the way humans perceive sound. To enhance the accuracy of speech recognition, a normalization step is employed to eliminate irrelevant information. From an information-theoretic perspective, feature extraction serves as a lossy compression method.

To calculate the feature vectors, windowed samples of the audio signal are used, typically consisting of 25 ms of audio, as it is assumed that the speech signal remains static during this time frame. Spectral analysis, commonly based on the Fast Fourier Transform (FFT), is a widely used technique in speech recognition [151]. However, most systems disregard the phase information after performing the Fourier transformation [1]. Triangular filter functions are then applied in the spectral domain, enabling the computation of Mel-frequency Cepstral Coefficients (MFCCs) [49] or Perceptual Linear Prediction (PLP) coefficients [80].

In hybrid DNN/HMM systems, additional techniques may be employed to enhance speech recognition performance, such as speaker adaptation. One common method for speaker adaptation is vocal tract length normalization, which accounts for the differences in vocal tract length among speakers. Furthermore, speaker adaptation can be incorporated into the training process through data augmentation, in which vocal tract length perturbation is utilized.

Neural networks often necessitate fewer processing steps, with filter banks frequently being used directly [128]. In some cases, pitch features are incorporated, offering benefits for tonal languages predominantly spoken in Asia. However, as discussed in later chapters, a separate feature extraction step may not always be essential, as some neural networks can directly classify speech from raw audio.

In hybrid DNN/HMM speech recognition systems, the acoustic model can directly classify raw speech using various techniques, such as convolutional neural networks (CNNs) or Sinc convolutions, as implemented in an architecture known as SincNet. Conventional CNNs may not converge as effectively, which is one of the advantages of Sinc convolutions. SincNet employs Sinc convolutions that fulfill a similar purpose as feature extraction: instead of performing an FFT, these convolutions directly apply a filter function in the time domain, which can be parameterized using learnable parameters.

End-to-end speech recognition systems often employ feature extraction techniques grounded in psycho-acoustics, primarily utilizing filter bank features. Alternatively, some systems classify from a speech spectrogram [76], leveraging convolutional neural network architectures employed in the field of computer vision. Furthermore, end-to-end systems may directly classify raw audio using Sinc convolutions, a technique that will be elaborated in Chapter 5.

2.2 Neural Networks

A thorough discussion of neural networks at large is beyond the scope of this work. This section serves to introduce fundamental concepts and pertinent formalism related to the neural networks discussed in this thesis. For a more in-depth introduction, readers are referred to textbooks such as [17] and [65].

2.2.1 Basic Notation of Neural Networks

Within the scope of this work, neural networks are employed for classification tasks, that is, they serve as models for class-posterior probabilities. Given an observation space and a finite set of C classes, the neural network can be described as a parameterized function:

$$g_\theta : \mathbb{R}^D \to \mathbb{R}^C, \quad x \mapsto g_\theta(x) \tag{2.4}$$

The parameters θ are learned using labeled data points from a given dataset. Instead of directly outputting a class decision, the function produces class posterior probabilities. For any input x, the function $g(x)$ generates a C-dimensional vector containing pseudo-probabilities, leading to the following decision rule:

$$r : \mathbb{R}^D \to c \in \{1, \cdots, C\}, \quad x \mapsto \arg\max_c \, p_\theta(c|x) \tag{2.5}$$

Single-layer neural networks, also known as perceptrons, consist of a single layer of input nodes connected to an output layer. While they can effectively model linearly separable problems, they are limited in their ability to learn complex, non-linear patterns in the data. The introduction of multi-layer neural networks, specifically feedforward networks with one or more hidden layers, marked a critical step in the evolution of neural networks. These hidden layers allow the network to learn more complex, non-linear relationships between inputs and outputs by transforming the input data through a series of non-linear activation functions. In mathematical terms, the network is composed of N layers and the organization of these layers defines the architecture of the network. Each layer is represented as a basic function, i.e.,

$$g_\theta = g^{(N)}_{\theta^N} \circ g^{(N-1)}_{\theta^{N-1}} \circ \cdots \circ g^{(1)}_{\theta^1}, \tag{2.6}$$

where each layer consumes the output of the previous layer of dimension D_{n-1}, and produces a tensor of dimension D_n, i.e.,

$$g^{(n)}_{\theta^n} : \mathbb{R}^{D_{n-1}} \to \mathbb{R}^{D_n}. \tag{2.7}$$

The first layer is termed as the input layer and responsible for receiving the raw data, typically in the form of feature vectors. Hidden layers are the intermediate layers of a neural network that lie between the input and output layers. A neural network with multiple hidden layers is known as a deep neural network, giving rise to the term “deep learning.” The output layer is the final layer in a neural network that produces the network’s predictions.

The last layer of multi-class classification networks typically uses a softmax to match the distribution of the posterior probabilities $p_\theta(c|x)$ of Equation 2.5. The softmax function [25] converts the output layer's raw scores into probability distributions over the possible classes, and is defined as

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{2.8}$$

where x_i is the input vector's i-th element, and n is the number of elements in the input vector. This value is computed for all i elements of x, resulting in an output vector that has the same size of x.

2.2.2 Feed-Forward Neural Network Layers

A single linear layer, also known as a fully connected or dense layer, connects each scalar of the input tensor to every scalar of the output tensor through a set of learnable weights, represented by the matrix W, and biases, denoted as b, thereby transforming the input data through a linear operation.

$$\text{Linear}(x) = Wx + b \tag{2.9}$$

A variant of the linear layer, denoted as $\text{Linear}'(x)$, excludes the bias term, resulting in a transformation determined solely by the weight matrix W and the input x.

$$\text{Linear}'(x) = Wx \tag{2.10}$$

One-dimensional Convolutional Neural Networks (1D CNNs) are a type of neural network that applies a convolution operation over a one-dimensional input, typically used for processing time-series data or sequences. The convolution operation in CNNs is commonly described by a number of key parameters: The filter, or kernel, f, is convolved over the input. The width or size of this filter is represented by k. As the filter scans the input, it moves in steps defined by the stride s. Another important parameter is the padding p which designates the number of zeros added to the input's edges. After the convolution is computed, a bias term b is added to the result. Additionally, i refers to the specific position in the input where the convolution is executed. The mathematical representation of a single-channel 1D convolutional layer using filter f and input x is

$$y_i = (f * x)_i = b + \sum_{j=1}^{k} f_j \cdot x_{i+j-1}. \tag{2.11}$$

where y_i is the output at position i, and $*$ denotes the convolution operation. To incorporate the padding p, the formula can be expanded to the more comprehensive form

$$y_i = \text{Conv}_{k,s,p}(f, x, i) = b + \sum_{j=1}^{k} f_j \cdot x_{i \cdot s+j-1+p}. \tag{2.12}$$

In Equation 2.12, $\text{Conv}_{k,s,p}(f, x, i)$ denotes a convolution operation at position i with filter f and input x with kernel width k, stride s, and padding p. The output at position i, y_i, is then the result of this convolution operation plus a bias term b. Padding p typically adds zeros around the input data before any convolution operation begins, and doesn't influence the position where the operation is applied directly. However, it does affect the size of the output and thus the range of i.

2.2.3 Recurrent Network Layers

Recurrent Neural Networks (RNNs) are widely employed in speech recognition tasks due to their proficiency in modeling sequential data and identifying temporal dependencies. RNNs excel at processing sequences by maintaining a hidden state that updates at each time step, allowing them to store information from previous time steps and capture context within the input data. Their ability to handle input and output sequences of varying lengths makes RNNs particularly well-suited for speech recognition tasks. Moreover, RNNs are trained end-to-end, processing input tokens and directly predicting output posteriors sequentially, eliminating the need for intermediate representations. RNNs can be defined as

$$h_t = \phi(W_h \cdot [h_{t-1}, x_t] + b_h) \tag{2.13}$$
$$y_t = W_y \cdot h_t + b_y. \tag{2.14}$$

Here, x_t represents the input at time step t, and h_{t-1} denotes the hidden state from the previous time step. The updated hidden state at the current time step is given by h_t, and the activation function, typically a non-linear function like tanh or sigmoid, is represented by ϕ. The weight matrices for the hidden state and output are represented by W_h and W_y, respectively, while the bias terms for the hidden state and output are given by b_h and b_y, respectively. Finally, y_t represents the output at time step t.

While training RNNs, the gradients of the loss function concerning the model parameters can occasionally diminish, resulting in a significantly slowed learning process or even a complete halt. To address this issue, Long Short-Term Memory (LSTM) networks were developed, incorporating gating mechanisms that effectively regulate the flow of information throughout the network. The LSTM cell consists of several gating mechanisms: the input gate, forget gate, output gate, and a memory cell. LSTM units [69] are defined as

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \odot c_{t-1} + b_f) \tag{2.15}$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \odot c_{t-1} + b_i) \tag{2.16}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.17}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \odot c_t + b_o) \tag{2.18}$$
$$h_t = o_t \odot \tanh(c_t), \tag{2.19}$$

where x_t represents the input at time step t, while h_{t-1} and c_{t-1} denote the hidden state and memory cell state from the previous time step, respectively. The forget gate

activation is given by f_t, and the input gate activation is represented by i_t. The updated memory cell state is given by c_t. The output gate activation is represented by o_t, and the updated hidden state is denoted as h_t. The weight matrices for each gate are represented by $W_{xf}, W_{xi}, W_{xc}, W_{xo}, W_{hf}, W_{hi}, W_{hc}, W_{ho}, W_{cf}$, and W_{co}, while the bias terms for each gate are given by b_f, b_i, b_c, and b_o. The sigmoid activation function is denoted by σ, the hyperbolic tangent activation function is represented by tanh, and the element-wise (Hadamard) product is denoted by $\odot$.

Bidirectional RNNs [166] process input sequences using two separate RNN layers, one moving forward and the other moving backward, enabling the capture of information from both past and future time steps. The hybrid CTC/attention architecture employed in this thesis utilizes bidirectional LSTMs (BLSTM). To differentiate between the forward and backward directions, the hidden states of the forward LSTM are denoted as $\overrightarrow{h_t}$, while the hidden states of the backward LSTM are represented as $\overleftarrow{h_t}$. Similarly, the weights for the forward and backward directions are distinguished as $W^{\rightarrow}$ and $W^{\leftarrow}$, respectively. The outputs from the forward and backward LSTMs are combined at each time step through concatenation:

$$\text{BLSTM}(x_t) = h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}] \tag{2.20}$$

Here, the function $\text{BLSTM}(x_t)$ serves as abstraction for bidirectional LSTM layers that are used in the RNN-based ASR architecture described in a later section.

2.2.4 Activation Functions

Activation functions are used in neural networks to introduce non-linearity, enabling the network to learn complex patterns in the data. These activation functions are applied element-wise to the output of a linear layer or other layers in the neural network, transforming the output values to introduce non-linear behavior.

The sigmoid function maps input values to a range between 0 and 1, often used as an activation function in binary classification tasks or as a gate function in LSTMs to control information flow between cells:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.21}$$

The Hyperbolic tangent (tanh) function scales input values to a range of $(-1, 1)$, commonly applied as an activation function in hidden layers of RNNs and LSTMs due to its zero-centered output, which can help mitigate the vanishing gradient problem:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.22}$$

The Rectified Linear Unit (ReLU) function sets all negative input values to zero and keeps positive values unchanged, widely used in the hidden layers of feedforward neural networks and CNNs, but less common in sequence-to-sequence models like RNNs, LSTMs, and Transformers due to its non-zero-centered output and potential for dead neurons.

$$\text{ReLU}(x) = \max(0, x) \tag{2.23}$$

The Leaky Rectified Linear Unit (Leaky ReLU) function is a variation of the ReLU function that allows for small negative values when the input is less than zero, helping to mitigate the vanishing gradient problem in deep neural networks and improve learning performance:

$$\text{LeakyReLU}_\alpha(x) = \max(\alpha x, x), \tag{2.24}$$

where α is a small positive constant, usually 0.01.

2.2.5 Supervised Training of Neural Networks

Supervised training of neural networks relies on training samples, which are input-output pairs. Inputs are data or features given to the network, and outputs are the desired outcomes the network should predict or generate. Let $\mathcal{D}$ be a dataset containing N training samples; then, each training sample (x_n, y_n) consists of an input x_n and its corresponding output (or target) y_n.

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\} \tag{2.25}$$

For end-to-end speech recognition networks, training samples (x_n, y_n) include input audio vectors and corresponding ground truth label sequences. While this section primarily addresses the general principles of supervised training for neural networks, the training of sequence discriminative and sequence generative architectures, as applied to speech recognition, are fundamentally based on the methodologies presented in this section. These specific techniques are examined in greater detail when discussing their respective architectures.

The training criterion, also termed loss function or objective function, evaluates a neural network's performance during training by quantifying the difference between its predictions and actual target values. The objective is to minimize the loss function by adjusting the model's parameters θ.

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(y_n; f(x_n; \theta)) \tag{2.26}$$

Here, $\mathcal{L}_i$ is the individual loss for the training sample, comparing the ground truth y_n with the predicted value $g_\theta(x_n)$, where g is the neural network function with parameters θ. The objective function $\mathcal{L}(\theta)$ is the average of the individual losses over all N training samples in the dataset. The goal during the training process is to find the optimal parameters θ^* that minimize this loss:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta) \tag{2.27}$$

Commonly used for classification tasks, cross-entropy loss measures the difference between the predicted probability distribution and the actual probability distribution for the target classes [25]. Cross entropy loss is defined as

$$\mathcal{L}_{\text{CE}}(x_n, y_n; \theta) = -\log p_\theta(y_n | x_n), \tag{2.28}$$

which, when minimized, leads to high probability for the correct class and low probability for other classes. The loss approaches zero when the correct class is predicted,

$$\mathcal{L}_{\text{CE}}(x_n, y_n; \theta) \approx 0 \quad \text{when} \quad p_\theta(y_n|x_n) \to 1.0. \tag{2.29}$$

The Softmax function, as in Equation 2.8, is used as output layer to normalize the output of a neural network towards a probability function, so that $p_\theta(y_n|x_n) \equiv g_\theta(x_n)$.

Neural network training is a non-convex optimization problem and the loss function's minimum cannot be analytically determined. Thus, iterative numerical optimization techniques are employed, utilizing the local gradient. The gradient is calculated using the backpropagation algorithm [26, 160, 163], which is a widely accepted method for computing the gradients of the objective function concerning each neural network parameter. These gradients are then utilized to update the weights and biases, ultimately minimizing the loss function. The backpropagation algorithm exploits the chain rule of calculus to compute the gradient of the loss function with respect to each weight within the network by propagating the error in reverse, from the output layer to the input layer.

The most prevalent iterative numerical optimization method is Stochastic Gradient Descent (SGD). It computes the gradient based on a small, randomly chosen subset, denoted as a mini-batch $\mathcal{B} \subset \mathcal{D}$. In each iteration i, the parameters θ are updated accordingly with the learning rate η_i:

$$\nabla\mathcal{L}(\theta, \mathcal{B}) = \sum_{n \in \mathcal{B}} \mathcal{L}_n(\theta) \tag{2.30}$$

$$\theta_i = \theta_{i-1} - \eta_i \nabla\mathcal{L}(\theta, \mathcal{B}) \tag{2.31}$$

The learning rate is not maintained at a constant value; instead, learning rate scheduling adapts the learning rate according to the specific architecture being used. SGD facilitates enhanced optimization and better convergence across various model architectures. In the context of this work, plain SGD is mainly used for the training of RNN language models and for the generation of adversarial noise; architecture-specific strategies will be discussed in the respective sections.

The networks trained in this work utilize several regularization techniques. One method is *dropout* [81, 174] that "drops out" the output of each neuron with probability p at each training step. As a result, the network becomes less sensitive to specific weights of neurons, promoting a more generalized model. *Weight decay* [84] adds a penalty to the loss function proportional to the magnitude of the weights, often through L_2 regularization; weight decay ensures that the model weights remain small, preventing overfitting and improving generalization. Finally, *early stopping* serves as a regularization method by monitoring the model's performance on a separate validation dataset. If the validation performance stops improving after several epochs, indicating possible overfitting on the training set, the training is halted early.

2.3 Hybrid DNN/HMM Approaches to ASR

Traditional ASR systems necessitate pretraining and several refinement stages. They consist of partially hand-crafted components for feature extraction, phoneme probability inference, and decoding linguistic priors. Training labels for a DNN are acquired only after estimating phoneme or state alignments via a Gaussian mixture model. The decoding process involves a global search over numerous possible word sequences using weighted finite state transducers.

Various open-source toolkits offer DNN/HMM speech recognizers. An overview is provided by Gaida *et al.* [61]. Notably, Kaldi [148] is the primary tool utilized for evaluation in Chapter 6.

Hybrid DNN/HMM approaches [21] employ Bayes' theorem and introduce the HMM state sequence $S = \{s_t \in \{1..J\} | t = 1, \cdots, T\}$ to decompose $p(Y|X)$ into three separately modeled parts,

$$\arg\max_Y p(Y|X) = \arg\max_Y \sum_S p(X|S,Y)p(S|Y)p(Y), \tag{2.32}$$

where the acoustic, lexicon, and language models are represented as $p(X|S,Y)$, $p(S|Y)$, and $p(Y)$, respectively. To make the acoustic model more manageable, the conditional independence assumption suggests that, given the corresponding word sequence, the observed acoustic features are independent of each other. Thus, $p(X|S,Y) \approx p(X|S)$, resulting in a simplified expression:

$$\arg\max_Y p(Y|X) \approx \arg\max_Y \sum_S p(X|S)p(S|Y)p(Y) \tag{2.33}$$

This assumption reduces dependencies between acoustic features, making it more straightforward to model and calculate probabilities.

2.3.1 Hybrid DNN/HMM Acoustic Model

In the acoustic model, $p(X|S)$, the probability is further factorized using the probabilistic chain rule and a further conditional independence assumption.

$$p(X|S) = \prod_{t=1}^{T} p(x_t|x_1, \cdots, x_{t-1}, S) \tag{2.34}$$

$$\approx \prod_{t=1}^{T} p(x_t|s_t) \tag{2.35}$$

$$\equiv \prod_{t=1}^{T} \frac{p(s_t|x_t)}{p(s_t)} \tag{2.36}$$

Equation 2.36 states the hybrid approach to speech recognition [21], which combines posterior probabilities of a neural network with HMM states. To use the neural network as

emission model, the frame-wise likelihood function $p(x_t|s_t)$ is replaced with the frame-wise posterior distribution $p(s_t|x_t)/p(s_t)$ computed by DNN classifiers. This rearrangement of the conditional probability is done by applying Bayes' rule, $p(a, b|c) = p(a|b, c)p(b|c)$.

The assumed conditional independence in the acoustic model does not take into account any input and hidden state contexts. To address the limitations arising from the conditional independence assumption, DNNs with long context features or recurrent neural networks are often employed, as incorporating additional context improves the system's accuracy.

To train the frame-wise posteriors, a frame-wise state alignment, s_t, is needed as a target. This alignment is typically provided by an HMM/GMM system, which offers the necessary contextual information for the acoustic model.

2.3.2 Lexicon Model

The lexicon model $p(S|Y)$ serves as an intermediary between the language model and the acoustic model, by providing a mapping between words and their corresponding HMM states, phonemes or sub-word units through a pronunciation dictionary. The pronunciation dictionary comprises a list of words, each accompanied by their phonetic representation in the form of phonemes, which are the most basic sound units that can distinguish words in a language. In some cases, a word might have multiple pronunciations, which can be due to dialectal variations, speaker-specific accents, or coarticulation effects. The lexicon model is factorized and simplified using a first-order Markov assumption:

$$p(S|Y) = \prod_{t=1}^{T} p(s_t|s_1, \cdots, s_{t-1}, Y) \tag{2.37}$$

$$= \prod_{t=1}^{T} p(s_t|s_{t-1}, Y) \tag{2.38}$$

The first-order Markov assumption states that the probability of the current state s_t depends only on its immediate predecessor s_{t-1}, and it is conditionally independent of all previous HMM states given the immediate predecessor.

2.3.3 Language Model

The language model, as expressed in equation 2.39, is designed to predict a sequence of words or symbols Y by learning the probability distribution of sequences. It does this by sequentially predicting each subsequent word or symbol y_l based on the preceding words or symbols $y_1, \cdots, y_{l-1}$.

$$p(Y) = \prod_{l=1}^{L} p(y_l|y_1, \cdots, y_{l-1}) \tag{2.39}$$

This can be factorized using the $(m-1)$st-order Markov assumption for m-gram models:

$$p(Y) \approx \prod_{l=1}^{L} p(y_l | y_{l-m}, \cdots, y_{l-1}) \tag{2.40}$$

Contemporary hybrid DNN/HMM systems utilize neural network language models that circumvent the conditional independence assumption issue, allowing for the modeling of long-term contexts. Architectures used for language models include Recurrent Neural Network Language Models (RNNLMs [126]) and attention-based Transformers [91]. These models and integration strategies into end-to-end ASR systems are discussed in Section 2.4.9.

2.4 Contemporary End-to-End Techniques for ASR

End-to-end ASR aims to simplify the training and decoding procedure by directly inferring sequential letter probabilities given a speech signal [31, 196]. Such systems usually transcribe speech features to letters or word fragments without any intermediate representations. It is also possible to train these networks in an end-to-end manner without previous refinement steps.

Conventional hybrid DNN/HMM speech recognition systems rely on these advances in technology: Mel Frequency Cepstral Coefficients reflect the spectral characteristics of the human auditory system. Hidden Markov Models that are excellent for statistical modeling of time sequences. DNNs provide a statistical model for classification.

In hybrid DNN/HMM systems, DNNs usually classify the hidden states within HMMs. These systems rely on handcrafted linguistic information. Training labels are not derived directly from text but rather in multiple refinement steps; with respect to their phonetic equivalents, statistical models determine the hidden states that serve as labels for the DNN. A global search over many possible word sequences then extracts the most probable word sequence by combining the output of the network, as well as linguistic and probabilistic prior knowledge.

The inferred classes can be either sequential or temporal. Modern large end-to-end networks achieve language modeling by using extensive text corpora for training. As described in the architecture chapter, certain parts of the end-to-end network already function as a language model. Training data typically comprises a set of utterances, where each utterance consists of an audio segment accompanied by its corresponding textual transcription.

2.4.1 Definition of End-to-End Speech Recognition

In the context of speech recognition, the term *end-to-end* is used to describe models that are *end-to-end trainable* [71]. An end-to-end model is trained directly from the input data to output the final transcription, i.e., from audio features to text output.

One could argue that this is not genuinely *end-to-end*. Firstly, these so-called end-to-end networks arguably still don't infer directly from raw audio; instead, they still

necessitate a feature extraction stage. Secondly, CTC-based end-to-end networks provide per-frame probabilities, which subsequently necessitate a forward algorithm for decoding. Finally, although attention models can generate correct sentences without a supporting language model, the quality of their transcription is typically enhanced by employing an additional model. The term *end-to-end* doesn't necessarily apply to single-model architectures that handle input data and produce the desired output without requiring distinct intermediate steps or modules as in other fields; for example, end-to-end object detection means that the object detection pipeline is without any non-differentiable component [175].

For speech recognition, early works on end-to-end architectures used the term *end-to-end* to emphasize the contrast from hybrid DNN/HMM systems, that require several stages during the training process [71, 196].

2.4.2 Types of End-to-End Architectures

Approaches utilized in modern end-to-end speech recognition systems can be classified into two main categories:

- **Sequence-generative** neural networks are designed to generate sequences of data, by predicting each next item based on previous items in the sequence.
- **Frame-discriminative** neural networks aim to classify or discriminate each individual input item in a sequence independently.

Sequence-generative neural networks typically employ attention-based encoder-decoder architectures, which offer flexibility to handle variable-length input but also require additional information about the sequential arrangement of the input features. In the literature, sequence-generative architectures are commonly labeled as sequence-to-sequence (*seq2seq*) architecture, a terminology introduced by Sutskever *et al.* [178]. Alternatively, attention-based architectures are also known as *Listen-Attend-Spell* (LAS), a term established by Chan *et al.* [31]. While Section 2.4.3 describes general properties of sequence-generative architectures using attention, the two fundamental attention architectures used in this work, namely RNN-based attention models and Transformers, are detailed in Section 2.4.4 and Section 2.4.5, respectively.

Section 2.4.6 discusses the general properties of CTC introduced by Graves *et al.* [70], which is the primary technique in modern end-to-end trained frame-discriminative neural networks. These employ frame-based classification via CTC, with the HMM-like structure of the CTC loss imposing strong temporal dependencies during decoding.

Some publications also mention RNN-Transducers [67] as a third category. Their architecture resembles the encoder-decoder structure of attention-based models. A transcription network (the encoder) extracts an intermediate representation, and a prediction network (the decoder) sequentially generates output tokens or output token posteriors. The prediction network's dictionary also includes a blank token, similar to CTC-based architectures. In a distinction to sequence-generative models, these output token posteriors do not directly represent the output sequence, but are conditioned as

probability lattices, and are decoded using the forward-backward algorithm. Unlike CTC-based models, RNN-Transducers have the advantage of generating output sequences longer than their input. However, this feature might not be as relevant for speech recognition tasks and is not used in this work.

2.4.3 Sequence-generative Models

In their simplest form, sequence-generative networks use an RNN to establish a probability distribution over the next-in-sequence token, essentially predicting the subsequent token in the sequence [69]. In an early work on sequence to sequence models, RNNs encoded the input into a fixed-size context vector, as in the models of Cho *et al.* [38, 39] and Kalchbrenner *et al.* [96]. The model of Sutskever *et al.* is depicted in Figure 2.1. In a similar approach by Sutskever *et al.* [178], the information of the input sequence was encoded into the hidden state of an LSTM, after which the end-of-sequence (EOS) token provides a signal for the LSTM to generate its output sequence. These approaches already employ an encoder-decoder architecture, as the encoder processes a variable-length input X and generates a fixed-size intermediate representation c, from which the decoder generates the variable-length output sequence Y.

$$c = \text{enc}(X) \tag{2.41}$$

$$Y = \text{dec}(c) \tag{2.42}$$

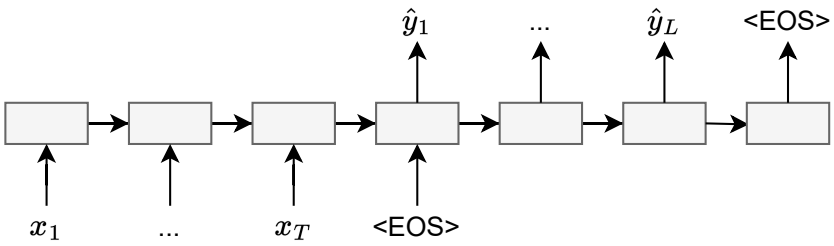


Figure 2.1: The sequence-generative architecture by Kalchbrenner *et al.* [96] that encodes the input sequence with an LSTM, generating an intermediate representation of the audio sequence. The neural network was then given the signal to generate output tokens with an additional end-of-sequence (EOS) token, as proposed by Sutskever *et al.* [178].

The attention-based encoder-decoder sequence transformation was a subsequent improvement of this principle, that was first proposed as a method for machine language translation by Bahdanau *et al.* in [10]. The attention mechanism performs a weighting for a variable-length input and aggregates all values into a single vector. Although such aggregation can be achieved using fixed-size convolutional kernels, as demonstrated by Elbayad *et al.* [56], the attention mechanism provides a flexible window size. In combination with a sequential decoder, the attention mechanism establishes alignments between the input and output sequences [42].

The LAS architecture was originally proposed with two main components: the *listener*, or encoder network, and the *speller*, or decoder network. The listener processes the input speech signal into a higher-level intermediate representation $H = \{h_t \in H | t = 1, \cdots, T\}$ that contains the information of the input sequence with annotations by the encoder. The speller incorporates an attention mechanism that scores the listener's output sequence for each output symbol, thereby providing an alignment function. Raw attention scores are combined with a Softmax function to obtain normalized attention weights $a_{l,t}$. The context vector c_l is generated from the attention weights for each sequential output step l that depends on a state q_{l-1} of the decoder, the decoding decision of the previous state, and the high-level representation H. The speller network then generates the output text sequence Y, one symbol y_l at a time.

$$H = \text{enc}(X) \tag{2.43}$$

$$c_l = \text{att}(H, y_{l-1}, q_{l-1}) \tag{2.44}$$

$$y_l = \text{dec}(c_l, y_{l-1}, q_{l-1}) \tag{2.45}$$

In this model, RNN-based networks use the *encoder-decoder attention* mechanism, which focuses on different parts of the input sequence for each step in the output sequence. A novel feature of the LAS model is its use of a pyramidal encoder, that reduces the temporal resolution of the speech signal, thereby enabling more efficient handling of long sequences.

Attention-based systems, such as LAS, are characterized by their lack of independence assumptions between sequence elements, enabling the capture of complex dependencies. The attention mechanism, integral to the decoder, dynamically shifts focus across different segments of the input, allocating attention, based on the relevance of the input components to the output being generated.

While encoder-decoder attention models like LAS employ attention mechanisms within the decoder to link output characters to relevant parts of the input sequence, Transformers models take a distinct approach with *self-attention*. In the Transformer architecture, self-attention operates within the encoder, establishing connections between each feature or intermediate value in the input sequence and every other value, capturing complex internal structures within the data.

2.4.4 Location-aware Encoder-decoder Attention

Similar to an alignment model in conventional ASR, RNN-based attention architectures employ a recurrent attention mechanism applied to the hidden values. The attention weights $a_{l,t}$ are used in this mechanism to influence the annotated sequence H for each vector h_t during the l-th decoding step. This work employs the location-aware attention that was described by Chorowski *et al.* in [43]. The weights depend on the input sequence that was encoded in H and on the carried-over state of the decoder q_{l-1}. Location-aware attention, denoted as att_{loc}, includes the attention weights of the previous sequential

output $a_{l-1,t}$.

$$a_{l,t} = \text{att}_{\text{loc}}(q_{l-1}, h_t, a_{l-1,t}) \tag{2.46}$$

$$= \text{Softmax}(g^T \cdot \tanh(\ \text{Linear}'(q_{l-1}) + \text{Linear}'(h_t) + \text{Linear}(K * a_{l-1,t})\ )) \tag{2.47}$$

In this context, $*$ represents the convolution operator applied with the trainable convolutional kernel K. A scalar product operation with the trainable vector g condenses the activations of the attention network's internal linear layers into a single scalar value. The attended parts of the sequence are then aggregated into the context vector c_l using a weighted sum, expressed as

$$c_l = \sum_{t=1}^{T} a_{l,t} h_t. \tag{2.48}$$

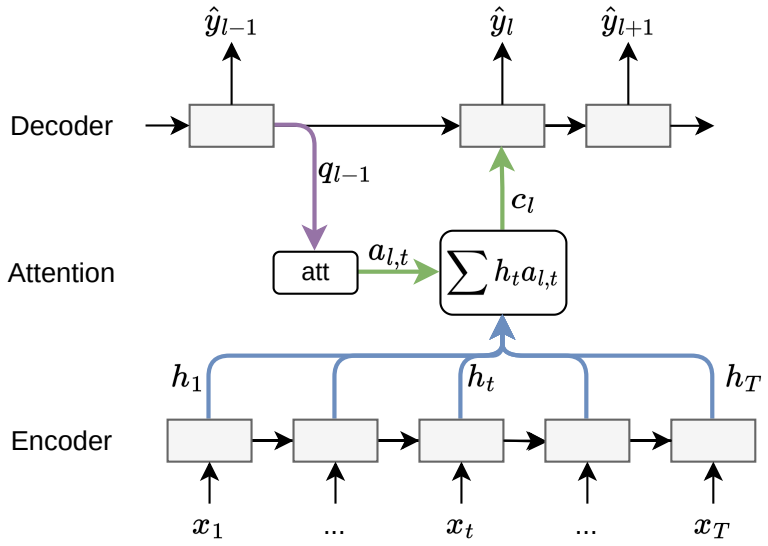


Figure 2.2: Encoder-decoder attention. An input sequence is processed by the encoder to produce an intermediate representation in the form of vectors h_t. These vectors are passed to the attention block which assigns weights $a_{l,t}$ to each h_t and computes a weighted sum to form the context vector c_l for each decoding step l. The context vector c_l is then used by the decoder to generate the output sequence one symbol at a time. A characteristic of the encoder-decoder attention mechanism is its dependence on the previous decoder state q_{l-1} when assigning weights $a_{l,t}$.

2.4.5 Attention in the Transformer Architecture

In RNN-based networks, such as LSTM encoders, feature vectors are provided as input in a sequential manner, with each vector being processed individually and in order. Both the encoder and decoder components of Transformer models use a dot-product attention mechanism that processes given sequences concurrently, as opposed to the recurrent nature of RNNs.

Transformer models were introduced by Vaswani *et al.* [189] for machine translation. Early adopters of Transformers to speech recognition were Dong *et al.* [52] for attention-based speech recognition, and Karita *et al.* [102] for the hybrid CTC/attention architecture. This work utilizes the implementation of Karita *et al.* for experiments.

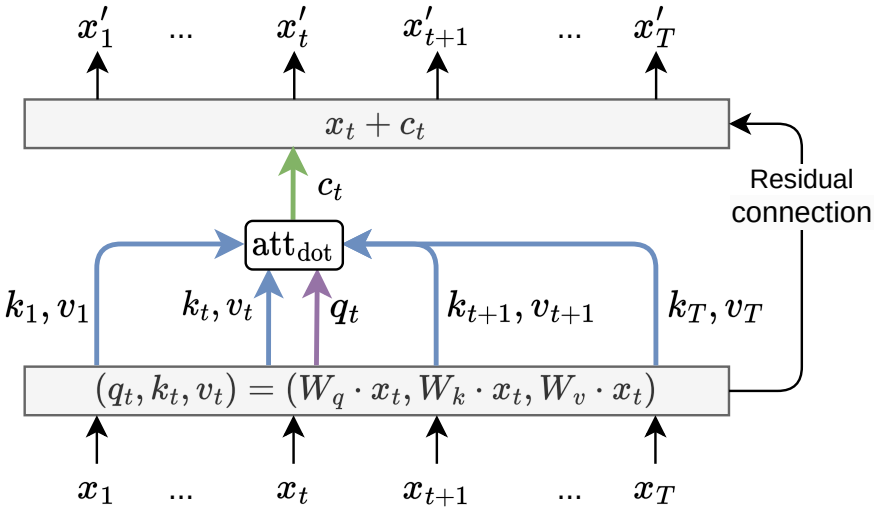


Figure 2.3: The self-attention sublayer of a Transformer with a single head. The input sequence vectors of $X = [x_1, \cdots, x_T]$ are transformed into query, key and value vectors. To demonstrate the information flow and data dependencies, the matrices Q, K, and V are represented as single-input slices q_t, k_t and v_t. Scaled dot product attention calculates the attention weights for each q_t with all key vectors, as in Equation 2.51. In the weighted summation step, each value vector is multiplied by its corresponding attention weight; the resulting weighted values are then summed across all sequence positions, thereby producing the output c_t of the self-attention layer for each query. Here, c_t corresponds to the slice of matrix C_h (the output of a single attention head, Equation 2.53) at position t. The original input is then added back to this output, forming a residual connection. In contrast to encoder-decoder attention, the query vector q_t in self-attention is derived directly from the input sequence.

The attention mechanism in the Transformer uses query (Q), key (K), and value (V) matrices as input to the attention layer. The K and V matrices are generally derived from the same input matrix and are analogous to the intermediate representation H in encoder-decoder attention models, both containing encoded information about the input sequence. Similarly, the query matrix Q carries information that is comparable to the decoder state q_l in encoder-decoder attention models, as both guide the attention mechanism. K and V share the same dimensions, i.e., $K, V \in \mathbb{R}^{n_k \times d_k}$ and $Q \in \mathbb{R}^{n_q \times d_k}$. Here, n_k and n_q represent the number of elements in the sequence and d_k is the dimensionality of features.

How the query, key, and value matrices are derived differs depending on whether the MHA is utilized as self-attention or as encoder-decoder attention. In the standard self-attention mechanisms, all three matrices (Q, K, V) are derived from the same input

matrix,

$$(Q, K, V)_{\text{self-attention}} = (X'_i, X'_i, X'_i) \text{ with } X'_i = \text{LayerNorm}(X_i). \tag{2.49}$$

This is typically true for both the encoder and decoder portions of transformer models that employ self-attention. For self-attention, the number of query elements equals the number of key-value pairs and thus, n_k and n_q are equal.

On the other hand, in encoder-decoder attention settings, as only used in the decoder, the query matrix usually comes from the decoder's state, while the key K and value V matrices are generated from the encoder's output H.

$$(Q, K, V)_{\text{encoder-decoder attention}} = (X'_i, X_e, X_e) \text{ with } X'_i = \text{LayerNorm}(X_i). \tag{2.50}$$

Transformers use scaled dot product attention that enables highly parallelized computation on GPUs. With QK^T as the attention matrix, the scaled dot product attention can be written as

$$\text{att}_{\text{dot}}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \tag{2.51}$$

To improve the stability of the numerical computation of the Softmax, the attention matrix is scaled by $1/\sqrt{d_k}$.

Transformers also employ multi-head attention in each self-attention sublayer, that allows the model to capture different types of information from different parts of the input. The concept of multi-head attention was previously integrated into end-to-end speech recognition using standard encoder-decoder attention as presented in [167] and explored in [37]. The model generates multiple queries, keys, and values from the input, processes each set independently as *heads*, then combines the results. The multi-head attention mechanism, which incorporates H heads[i], functions as a wrapper around the dot product attention sublayer. It utilizes the same input matrices Q, K and V, multiplied with learnable weight matrices $W_h^q, W_h^k, W_h^v \in \mathbb{R}^{d_k \times d_k}$ for each head h, and delivers an output with identical dimensions by multiplying with the weight matrix $W^{\text{head}} \in \mathbb{R}^{(Hd_k) \times d_k}$.

$$\text{MHA}(Q, K, V) = [C_1, C_2, \cdots, C_H] W^{\text{head}} \tag{2.52}$$

$$C_h = \text{att}_{\text{dot}}(QW_h^q,\, KW_h^k,\, VW_h^v) \quad \text{with } h = 1, \cdots, H \tag{2.53}$$

After the attention sublayers, the output is layer-normalized [9] and passed through a feed-forward neural network $\text{FFN}(x)$ that is applied to each position. This network has two linear layers, as in Equation 2.9, and a ReLU activation in between. Its output is then added to the input of the normalization layer, forming a residual connection. Layer

[i] In the notation presented by Vaswani *et al.* [189], the number of attention heads is represented by C and the outcome of the dot product attention is labeled as H_c. In this explanation, we've altered the symbol notation to highlight a parallel: The context vector c_l of Equation 2.48 signifies the result of the encoder-decoder attention, capturing annotated context information. Similarly, C_h symbolizes the output of an individual attention head, containing its own annotated context information, as defined in Equation 2.53.

normalization is also applied before the multi-head attention [203]. A single multi-head attention block with input X_i and output X_{i+1} is determined by:

$$\mathrm{FFN}(x) = \mathrm{Linear}(\,\mathrm{ReLU}(\,\mathrm{Linear}(x)\,)\,) \tag{2.54}$$

$$X_i' = \mathrm{MHA}(Q, K, V) + X_i \qquad \text{with } K, Q, V \text{ from Equation 2.49} \tag{2.55}$$

$$X_{i+1} = \mathrm{FFN}(\mathrm{LayerNorm}(X_i')) + X_i' \tag{2.56}$$

Figure 2.3 shows a simplified structure of the self-attention sublayer. While vectors are represented separately for illustrative purposes in this figure, during actual GPU-based computation, they are referred to as matrices Q, K, and V for each layer or attention head. Given that all computations in a single layer can be viewed as matrix-matrix multiplications and additions, these operations are highly parallelizable and efficient on modern GPUs.

2.4.6 Frame-Discriminative Connectionist Temporal Classification (CTC)

CTC enables temporal classification using end-to-end neural networks, but does not require the network to be combined with hidden Markov models [68, 70]. With CTC, RNNs or Transformers can be trained directly for temporal classification tasks. CTC achieves this by allowing the network to make label predictions at any point in the input sequence, under the condition that the overall sequence of labels is correct. Pre-segmentation of speech data is no longer necessary, since the alignment of the labels with the input is no longer important. CTC directly estimates the probabilities of the complete label sequences, effectively acting as a temporal classifier.

In a preparation step at training, the L-length token sequence $\overline{Y} = \{\overline{y}_l \in \mathcal{U} | l = 1, \cdots, L\}$ of the token dictionary $\mathcal{U}$ is extended to an $(2L+1)$-length letter sequence Y^* that includes the blank token "ϵ":

$$Y^* = \{\epsilon, y_1, \epsilon, y_2, \epsilon, \cdots, \epsilon, y_{L-1}, \epsilon, y_L, \epsilon\} \tag{2.57}$$

This resembles a left-right HMM with three states for each token with these transitions, i.e., $[\epsilon, y_l, \epsilon]$ for each token in the sequence. The function $\mathcal{B}$ as a many-to-one mapping collapses sequences, removing redundant symbols, i.e., removing the ϵ as well as any repeated tokens of a sequence, for example:

$$\mathcal{B}(a\epsilon ab\epsilon) = \mathcal{B}(\epsilon aa\epsilon\epsilon abb) = aab \tag{2.58}$$

Let $S_\pi = \{\pi | \mathcal{B}(\pi) = Y\}$ be the set of all sequences with length T that map onto Y after reduction. The conditional probability of the output sequence Y is determined by summing the probabilities of all the possible alignments in S_π,

$$p_{\mathrm{CTC}}(Y|X) = \sum_{\pi' \in S_\pi} p(\pi'|X). \tag{2.59}$$

The alignment probability $p(\pi'|X)$ of a single path given the matrix of token probabilities $C_{\text{ctc}} \in \mathbb{R}^{T\times(|\mathcal{U}|+1)}$ is

$$p(\pi'|X) = \prod_{t=1}^{T} C_{\text{ctc}}[t, \pi[t]]. \tag{2.60}$$

Here, the matrix C_{ctc} is the output of the network itself. Section 2.4.7 describes the CTC modules used in the RNN and Transformer architectures.

During the training process, the CTC loss function tunes the network for predicting frame-wise token probabilities. This CTC loss function determines the likeliest path through the label sequence by computing the probability of a specific target sequence through aggregating the frame-based letter probabilities across all potential paths in the modified letter sequence, using the forward-backward algorithm. This computation of path probabilities is tractable, i.e., its gradient can be efficiently computed, and thus can be used as objective function in neural network training.

With this loss function, the neural network converges towards classifying token occurrences, i.e., the network aims to activate each token at only a single time step, while classifying the blank token at the other time steps. Figure 2.4 shows an equivalent HMM state diagram, showing the transitions of the CTC state sequence for the word "CAT".

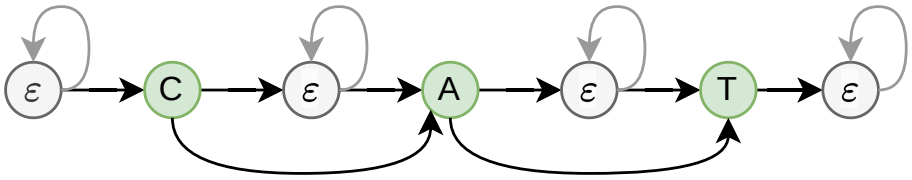


Figure 2.4: An equivalent HMM state diagram, illustrating the transitions of the CTC state sequence for the word "CAT".

Kanda *et al.* [98] partitioned the probability $p_{\text{CTC}}(Y|X)$ into an acoustic model and a letter model, shedding more light on the similarities between HMM-based ASR systems and CTC. Watanabe *et al.* [196] also used this formulation when outlining the hybrid CTC/attention architecture.

This work employs the CTC loss function provided by the PyTorch Toolkit [142]. It's worth noting that, although not utilized in this research, there exist other variants of this loss function which could be used as suitable alternatives. Of particular interest are recent advancements in computationally efficient and GPU-executable weighted finite state transducers, like GTN [77] and k2 [103], which have the capability to mimic the HMM structure of CTC.

2.4.7 Inference of CTC Posterior Probabilities

The matrix C_{ctc} with frame-wise token probabilities $p(y_t)$, that is used in Equation 2.60, is derived directly from the encoded values $H = [h_1, \cdots, h_T]$. Both RNN and Transformer architectures only use a single fully connected layer to generate the token posterior

probabilities from each h_t:

$$C_{\text{ctc}} = \text{Softmax}(\text{Linear}(H)) \tag{2.61}$$
$$= [\text{Softmax}(\text{Linear}(h_1)), \cdots, \text{Softmax}(\text{Linear}(h_T))] \tag{2.62}$$
$$= [p(y_1|h_1), p(y_2|h_2), \cdots, p(y_T|h_T)] \tag{2.63}$$

Here, Linear(H) represents the application of the linear layer to each encoded value h_t, producing a logit vector for each frame t. The Softmax function then converts these logits into a probability distribution over the possible tokens for each frame.

The encoded representation h_t at each time step t is transformed into a vector of token probabilities $p(y_t|h_t)$, capturing the likelihood of each token occurring at that specific time step. This probabilistic output C_{ctc} also serves as the input to the subsequent CTC loss computation, enabling the model to align the predicted token sequences with the ground truth sequences during training.

2.4.8 Text Tokenization

The transcription is represented as a sequence of tokens. During the training process, the text is tokenized, meaning it is encoded as a series of integer token IDs that correspond to the original text. At tokenization, out-of-vocabulary characters cannot be represented as tokens and are therefore excluded from the process.

The architectures discussed in this work employ the following tokenization methods: (1) Byte-Pair Encoding (BPE), (2) unigram tokens and (3) simple character-level tokenization. Both BPE and unigram tokens are determined using subword-based tokenization with SentencePiece.

Character-level tokenization uses one token per character in the text, providing a straightforward and simple representation. Subword-based tokenization with SentencePiece [107] incorporates either unigram or BPE encodings to generate token representations. In this approach, frequently occurring shorter words are represented as individual tokens, while single characters are also tokenized, allowing for the construction of more complex and infrequent words. Notably, SentencePiece can handle languages that do not separate words with spaces, as it processes text as a continuous input stream. BPE [60] determines tokens by beginning with individual characters and successively merging the most frequently occurring character combinations throughout the text. The unigram [106] method starts with an extensive vocabulary and gradually reduces it to the desired number of tokens, while consistently retaining the base characters in the token dictionary.

2.4.9 Language Models for End-to-end Networks

While attention models are capable of generating coherent sentences, they are typically trained on relatively smaller text corpora compared to standard language models. They may occasionally produce ungrammatical or nonsensical sequences.

In the context of attention-based models, Gulcehre *et al.* [73] introduced shallow fusion; further integration of beam search into the seq2seq framework was described by Wiseman *et al.* [198]. Shallow fusion incorporates a pre-trained language model into the beam search decoding. Shallow fusion applies a language model rescoring, i.e., it merges the attention model's log probabilities with those from the language model during the beam search decoding phase using a weight parameter β:

$$\log p(\hat{y}_l) = \log p_{\text{Att}}(\hat{y}_l) + \beta \cdot \log p_{\text{LM}}(\hat{y}_l) \tag{2.64}$$

2.5 Hybrid CTC/Attention End-to-End Architectures

End-to-end models, as opposed to hybrid DNN/HMM systems, offer a simplified ASR pipeline and eliminate the need for complex legacy architectures. Attention-based frameworks consolidate acoustic, lexicon, and language models into a unified encoder-decoder network. CTC networks skip hidden states and directly deduce frame-wise probabilities of characters or tokens, eliminating the need for Markov or conditional independence assumptions. End-to-end models typically utilize letter, or BPE/unigram token outputs instead of word representations, that enables them to manage out-of-vocabulary situations and enhances generalization. These models can be jointly optimized as a single network for enhanced coherence. Also, end-to-end models do not require further adaptive techniques, such as speaker adaption or vocal tract length normalization.

Watanabe *et al.* [195, 196] amalgamated CTC and attention approaches in the hybrid CTC/Attention architecture, which optimizes both networks simultaneously for improved performance over either model used independently. This hybrid model efficiently manages irregular alignments frequently seen with standalone attention models. The pairing of CTC's monotonic alignment and the attention model's ability to tackle complex mappings accelerates convergence during training. In the decoding phase, the hybrid model merges attention-based and CTC scores using a one-pass beam search algorithm. This section introduces the main components to combine CTC and attention in a single architecture.

Hybrid CTC/attention with RNNs is discussed in Section 2.6 and with Transformers in Section 2.7. Karita *et al.* [101] provide an extensive comparison between these RNNs and Transformers with this architecture.

2.5.1 Model Training based on Multiobjective Learning

The application of a multi-objective training function [104, 119] merges CTC and attention loss through the introduction of a multi-objective training factor κ, which ranges from 0 to 1. Thus, the combined loss, $\mathcal{L}_{\text{hybrid}}$, is expressed as

$$\mathcal{L}_{\text{hybrid}} = \kappa \mathcal{L}_{\text{CTC}} + (1 - \kappa)\mathcal{L}_{\text{Att}}. \tag{2.65}$$

A network solely comprising an attention network results from training with $\kappa = 0.0$, while a network solely consisting of a CTC network arises from training with $\kappa = 1.0$. Networks

trained with κ values between 0.0 and 1.0 (excluding the extremes) are denominated as hybrid models in this work, whereas networks with $\kappa = 0.0$ and $\kappa = 1.0$ are referred to as attention-only and CTC-only models, respectively.

The CTC loss is used as discussed in Section 2.4.6. The loss function for the attention decoder is also calculated using cross entropy, given as $\mathcal{L}_{\text{Att}} = -\log p_{\text{Att}}(Y|X)$. The Adadelta optimizer is typically employed for RNN networks [213] and the Adam optimizer [46] for Transformers. Popel *et al.* [147] discusses suitable training parameters for Transformers.

For the purpose of data augmentation, spectral augmentation techniques are applied on top of the extracted features using SpecAugment [141]. The spectral augmentation involves two techniques: time warping and dropout-like masking. Time warping distorts a portion of the input features to simulate variations in speech speed, while the dropout-like masking is applied to blocks in both the feature and time dimensions.

2.5.2 CTC/Attention Joint Decoding

The Hybrid CTC/attention architecture employs a one-pass decoding beam search technique that melds frame-based and sequential letter probabilities [196]. This algorithm seeks the most likely sequence of letters $\hat{Y}$ for the hypothesis Y, represented as:

$$p(\hat{Y}) = \arg\max_{Y} p(Y|X) \tag{2.66}$$

In the context of beam search, letters are joined together based on their posterior probabilities, forming a rebuilt letter sequence [67]. This study utilizes the commonly adopted joint decoding approach which straightforwardly combines the sum of log probabilities from both the attention and CTC models [10, 87]. In employing beam search decoding, attention networks directly yield the next-in-sequence token probabilities, or partial hypotheses Y', and these are arranged in order of their likelihood [31]. This continues until the sequence conclusion is identified.

CTC generates probabilities for tokens on a frame-by-frame basis. To modify this to fit the sequential output probabilities, the calculation of the forward probability becomes an extra step needed to determine the next token probability in the sequence as produced by CTC. This probability within the CTC network is estimated using a prefix probability [68, 78]. Let $S_h = \{h \cdot \nu | \nu \in (\mathcal{U} \cup \text{EOS})^+\}$ be the set of all label sequences that start with the prefix h and continue with any non-empty label sequence ν from the set of all labels $\mathcal{U}$ and end with the end-of-sequence symbol (EOS). The prefix Y' includes the first $L_{Y'}$ tokens or characters of the full utterance hypothesis. The prefix probability represents the conditional probability of the partial hypothesis Y' by summing over the probabilities of all label sequences in $S_{Y'}$,

$$p_{\text{ctc}}(Y', \ldots | X) = \sum_{\pi \in S_{Y'}} p(\pi|X). \tag{2.67}$$

Furthermore, the CTC score [196] is then defined from the probability $p_{\text{ctc}}(Y', \ldots | X)$ as

$$\alpha(Y', X) \triangleq p_{\text{ctc}}(Y', \ldots | X). \tag{2.68}$$

Within the attention network, the probability of a partial hypothesis Y' is determined by applying the probabilistic chain rule, hence

$$p_{\text{att}}(Y'|X) \approx \prod_{l=1}^{L_{Y'}} p(y_l|y_i, \ldots, y_{l-1}, X). \tag{2.69}$$

The probability approximation for the attention model $p_{\text{att}}(Y|X)$ is computed as the product of conditional probabilities for each token given previous tokens and input X. For the CTC model, $p_{\text{CTC}}(Y|X)$ is approximated by the forward function $\alpha(Y, X)$ on the sequence and the input X. The CTC posterior, unlike the attention posteriors, does not have a recursive calculation method; however, it can be computed in an efficient manner by maintaining the forward probabilities across the input frames for every individual partial hypothesis [196].

The probability of a full hypothesis is estimated as:

$$p_{\text{att}}(Y|X) \approx \prod_{l=1}^{L} p(y_l|y_i, \ldots, y_{l-1}, X). \tag{2.70}$$

$$p_{\text{ctc}}(Y|X) \approx \alpha(Y, X) \tag{2.71}$$

The joint decoding further fuses the language model using shallow fusion [73]. Shallow fusion combines the log probabilities of the attention model and CTC model with the log probabilities of the language model using a weight β. The hybrid probability is then computed in a multi-objective style, using the weight parameters λ as CTC probabilities and β for language model probabilities. The calculation is then represented as:

$$p_{\text{hybrid}}(Y|X) = \lambda\, p_{\text{CTC}}(Y|X) + (1-\lambda) p_{\text{att}}(Y|X) + \beta\, p_{\text{LM}}(Y|X). \tag{2.72}$$

2.6 The Hybrid CTC/Attention RNN Architecture

The RNN-based architecture was initially introduced as the inaugural Hybrid CTC/Attention architecture, which is thoroughly explored in Chapter 3. The RNN architecture, as discussed, comprises the encoder, attention network, and decoder. This builds on the location-aware attention mechanism outlined in Section 2.4.4.

2.6.1 BLSTM Encoder

The architecture takes as input 80-dimensional log-Mel filterbank features, with the option to include an additional three pitch features. These pitch features can enhance ASR performance especially for tonal languages, like Mandarin or Vietnamese [63, 125]. Starting with an input sequence X of feature vectors, it is converted by the encoder into a hidden representation $h_1, \cdots, h_T$. The encoder network, shown in Figure 2.5, integrates a convolutional VGGnet [170] and a stacked BLSTM layer, topped with a fully connected layer of projection neurons, represented by Linear().

$$h_t = \text{enc}_{\text{RNN}}(X) = [\text{Linear}(\text{BLSTM})]^{N_{\text{elayers}}}(\text{VGG}_2(X)) \tag{2.73}$$

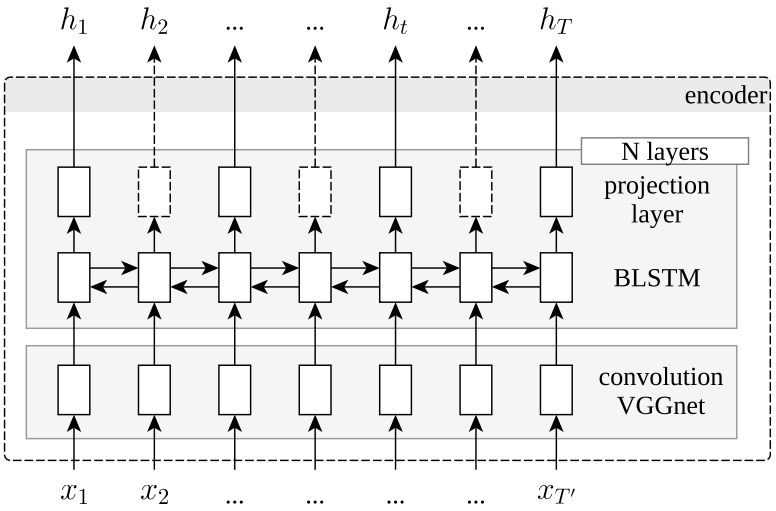


Figure 2.5: Encoder of the hybrid CTC/attention RNN-based architecture.

The encoder uses VGG2L [195], a simplified version of the VGGnet architecture. This VGG2L variant consists of two blocks, each containing two convolutional layers followed by a ReLU activation function and a max-pooling layer. The convolutional layers in the first block have 64 filters and the ones in the second block have 128 filters. After these convolution and pooling blocks, the features are flattened and passed through a linear layer to produce the output[ii] to the LSTM layers of the encoder.

The projection neurons within the BLSTM blocks apply subsampling to the features to reduce the computational complexity of the model. Resembling the behavior in other attention encoder-decoder models, they implement a pyramid-like structure, similar to the pyramidal encoder of the LAS architecture [31], meaning they systematically diminish the temporal resolution of the representation, for instance, in a 2-to-1 manner.

2.6.2 Decoder with Location-aware Attention

In this work, the RNN-based model incorporates the location-aware attention mechanism, denoted as att_{RNN}, in accordance with Equation 2.46. This mechanism operates by accepting the decoder's internal state vector and the annotated speech sequence $h_1, \cdots, h_T$ as its input. Subsequently, it yields the context vector c_l for each sequential output step l, as defined in Equation 2.48.

The attention decoder network, denoted as dec_{RNN}, uses the context vector to generate posterior probabilities for each letter in the sequence, including an extra label *EOS* signifying the sequence's end. The decoder is composed of several LSTM layers, without

[ii] Its output dimension is determined by the feature dimension D_{feat}, number of feature channels D_{fc}, the number output filters and the two max-pooling layers, $D_{\text{vgg}} = 128 \left\lfloor \left\lfloor \frac{D_{\text{feat}}}{D_{\text{fc}}} \div 2 \right\rfloor \div 2 \right\rfloor$. With 80-dim log-Mel filterbank features in a single channel, the output dimension to the BLSTM layer is 2560.

any intermediate projection layers or subsampling. The decoder operates recursively, using its previous internal state vector q_{l-1} and the predicted previous letter $\hat{y}_{l-1}$ from the beam search.

$$q_l = \text{dec}_{\text{RNN}}(c_t, \hat{y}_{l-1}) = [\text{LSTM}]^{N_{\text{dlayers}}}(c_t, q_{l-1}, \hat{y}_{l-1}) \tag{2.74}$$

$$p(y_l) = \text{Softmax}(\text{Linear}(q_l)) \tag{2.75}$$

Notably, q_l serves a dual role: it acts as the internal state guiding the attention mechanism and the next-in-sequence output token is classified based on it. Utilizing the hidden values produced by the encoder, the attention and decoder networks cooperatively generate a sequence in a recurrent fashion, as depicted in Fig. 2.6. With the calculated letter posteriors $p(y_l)$, the beam search selects a likely letter hypothesis $\hat{y}_l$. The process then cycles back to the attention network and continues this way until it encounters the label signifying the end of the sequence. The configuration of the CTC output layer is

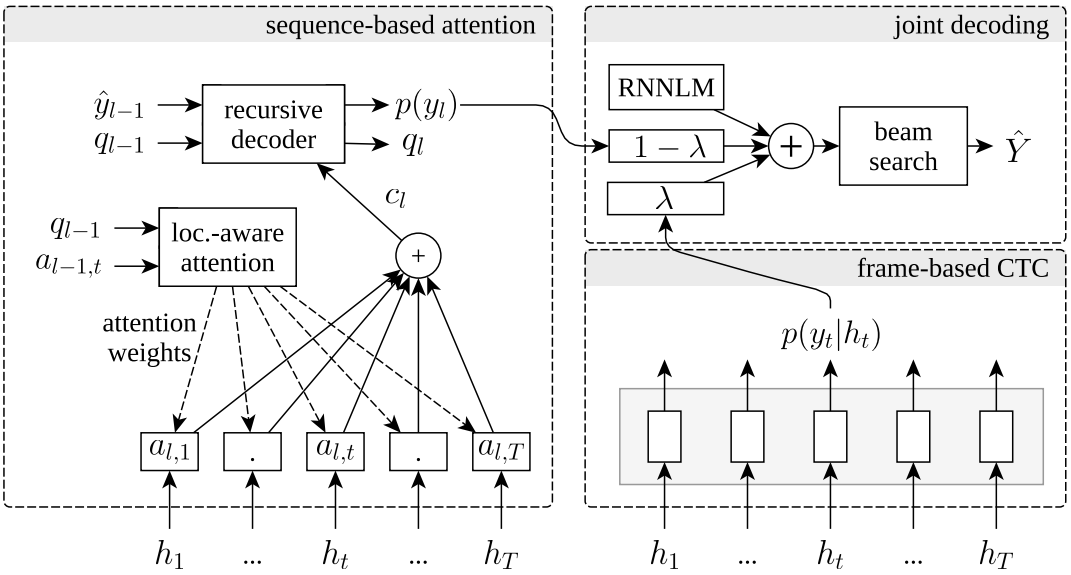


Figure 2.6: Decoder of the hybrid CTC/attention architecture.

depicted in Figure 2.6, adjacent to the attention mechanism. The encoder's pre-computed hidden values inherently incorporate a short-term context drawn encoded in the hidden representation h_t. A linear layer then calculates the CTC posterior probabilities on each frame using a linear layer as in Equation 2.63.

2.7 The Hybrid CTC/Attention Transformer Architecture

The Transformer encoder-decoder architecture uses attention in three different ways. The decoder attends over all items in the encoded input using encoder-decoder attention,

i.e., the queries derive from the preceding decoder layer, while the memory keys and values originate from the encoder output. Self-attention layers in both the encoder and decoder contain regular self-attention layers that derive its keys, values and queries from the previous layer, with the decoder implementing a mask to preserve the auto-regressive property by preventing future information from influencing the current or past positions. Figure 2.7 provides a visual overview of the Transformer architecture, which is further detailed in the section that follows. The formulae in this section are primarily based on the notation of Karita *et al.* [102].

2.7.1 Transformer Encoders

At the input layer, the model uses 80-dim log-Mel filterbank features; the extracted features $x_1, \cdots, x_T$ are concatenated in the feature matrix $X \in \mathbb{R}^{T \times 80}$. The features in X are then subsampled to X_{sub} using a convolutional neural network that has stride size of 2 and a kernel size of 3. Within transformer encoders, positional encodings are added to the acoustic feature vectors to convey positional information for each token in the sequence, ensuring the model captures the contextual relationships between tokens effectively. Speech recognition Transformer architectures typically use sinusoidal positional encoding as positional encoding (PE) function [47, 102, 189]. The input X_0 to the first encoder layer is the matrix of features stacked on top of positional encodings:

$$\text{PE}_t = \begin{cases} \sin\left(\frac{t}{10000^{\frac{t}{d_k}}}\right) & \text{if } t \text{ is even} \\ \cos\left(\frac{t}{10000^{\frac{t}{d_k}}}\right) & \text{if } t \text{ is odd.} \end{cases} \tag{2.76}$$

$$X_0 = \begin{bmatrix} x_{\text{sub},1} & \cdots & x_{\text{sub},T'} \\ \text{PE}(1) & \cdots & \text{PE}(T') \end{bmatrix} \text{ with } X_{\text{sub}} = [x_{\text{sub},1}, \ldots, x_{\text{sub},T'}] \tag{2.77}$$

Each encoder layer leverages the multi-head self-attention mechanism described in Equation 2.52, complemented by the feed-forward network from Equation 2.54.

$$X_i' = X_i + \text{MHA}(X_i, X_i, X_i) \tag{2.78}$$

$$X_{i+1} = X_i' + \text{FFN}(X_i') \tag{2.79}$$

The final output from the last encoder layer is represented as the annotated sequence X_e.

2.7.2 Transformer Decoders

Similar as its RNN counterpart, the Transformer decoder serves as a next-in-sequence predictor. For this, the decoder consumes the encoded sequence X_e and the sequence of preceding tokens. The following paragraphs describe the inner workings of the decoder that has d layers.

The prefix sequence of the already decoded partial sequence $Y' = [y_0, \cdots, y_{l-1}]$ is determined as a sequence of token IDs. To convert it into a learnable format, the token

IDs are encoded using an embedding layer.

$$Y_{\text{emb}} = \text{Embed}(Y') \tag{2.80}$$

The embedding matrix is concatenated with positional encoding to obtain the input to the first decoder layer Z_0.

$$Z_0 = \begin{bmatrix} y_{\text{emb},0} & \cdots & y_{\text{emb},l-1} \\ \text{PE}(1) & \cdots & \text{PE}(l-1) \end{bmatrix} \tag{2.81}$$

The decoder uses multi-head attention differently compared to the pure self-attention in the encoder, in two ways: First, the decoder applies masked multi-head attention onto the layer input. Second, multi-head attention is used as regular encoder-decoder attention include the input of the encoded audio sequence.

In masked multi-head attention, the decoder attends only to earlier sequence positions, inhibiting leftward information propagation. Unlike standard multi-head attention, its dot product attention, as in Equation 2.51, omits the lower triangular section of the QK^T matrix. The masked attention is thus determined by

$$M_{i,j} = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{otherwise} \end{cases} \tag{2.82}$$

$$\text{att}_{\text{dot, masked}}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d_k}}\right) \cdot V, \tag{2.83}$$

which is integrated into the multi-head mechanism $\text{MHA}^{\text{masked}}(Q, K, V)$ and applied to obtain the first intermediate result

$$Z'_j = Z_j + \text{MHA}^{\text{masked}}(Z_j, Z_j, Z_j). \tag{2.84}$$

Following the masked multi-head attention step, the encoder-decoder attention focuses on the encoded sequence. In this encoder-decoder attention mechanism, while the computation is analogous to standard self-attention, the distinctions arise from the source of its inputs. Specifically, the keys and values for attention are derived from the encoded sequence, whereas the queries are sourced from the encoder. This mechanism is termed as encoder-decoder attention, drawing parallels with the flow of information seen in the listen-attend-spell attention architecture.

$$Z''_j = Z_j + \text{MHA}^{\text{encoder-decoder}}(Z_j, X_e, X_e) \tag{2.85}$$

As a finalizing step for each layer, the output is propagated through the feed forward network of Equation 2.54.

$$Z_{j+1} = \text{FFN}(\text{LayerNorm}(Z''_j)) + Z''_j \tag{2.86}$$

Finally, a fully connected layer with a Softmax determines the probability of the next-in-sequence token $p(y_l)$ from the previous-in-sequence output of the last layer $Z_d[l-1]$.

$$p(y_l) = \text{Softmax}(\text{Linear}(Z_d[l-1])) \tag{2.87}$$

Figure 2.7 gives an overview of the Transformer encoder-decoder architecture.

2.7.3 Conformer Architecture

The Conformer, as introduced by Gulati *et al.* [72], is an augmentation of the Transformer architecture, embedding a convolutional layer after each attention layer. This architecture replaces standard Transformer blocks with Conformer blocks, by integrating Transformer's self-attention with CNNs for ASR, which helps to also capture local patterns using convolutions. The combination of CNNs and attention layers in the Conformer follows a sequential processing approach, inspired by Macaron-Net [120]. The Conformer adopts relative sinusoidal positional encodings, as introduced in the Transformer-XL [47] architecture.

While this architecture is only used for a comparative experiment in Section 6.4, an in-depth discussion of this architecture is beyond the scope of this introductory segment. Readers seeking an in-depth understanding are directed to the seminal work by Gulati *et al.* [72]. This work uses the implementation for ESPnet by Guo *et al.* [74].

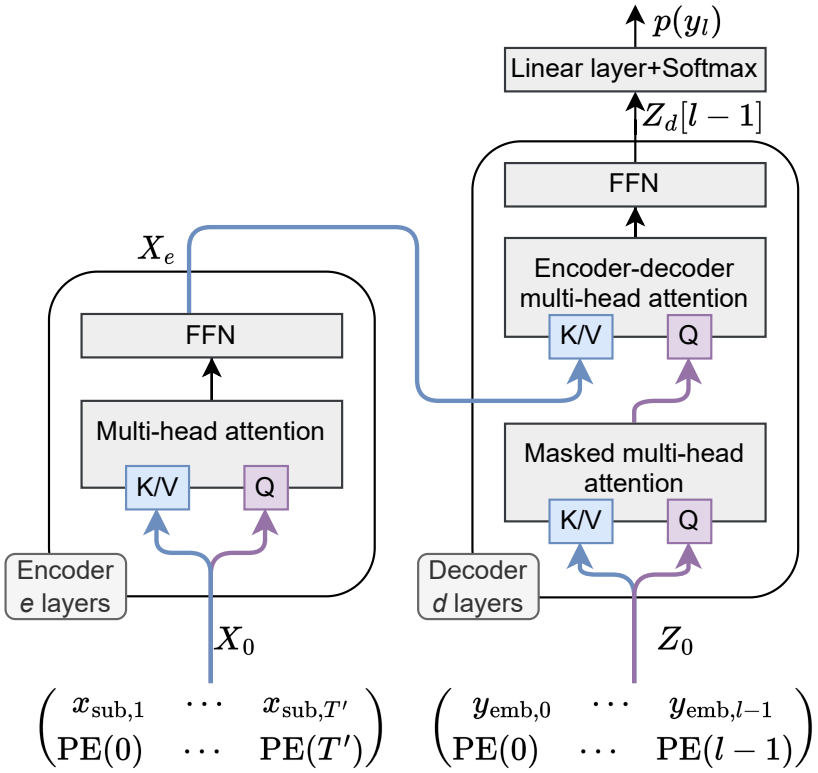


Figure 2.7: Encoder and decoder of the Transformer architecture.
On the left side, the encoder is depicted. The first encoder layer consumes the subsampled input features $x_{\text{sub},t}$ combined with positional encoding $\text{PE}(t)$. Within each layer, the data is passed through a regular self-attention block and a feed-forward neural network block.
On the right side, the decoder is shown. The already decoded partial sequence, $y_0, \cdots, y_{l-1}$, is first embedded and concatenated with its positional encoding to produce the inptu to the first decoder layer Z_0. Each layer of the decoder processes its input using masked multi-head attention, ensuring that future information doesn't influence current or past positions. Following this block, encoder-decoder multi-head attention attends to the encoded sequence H, before a feed-forward network refines the output for the current decoder layer.
Finally, a fully connected layer with a Softmax determines the probability of the next-in-sequence token $p(y_l)$ from the previous-in-sequence output of the last layer $Z_d[l-1]$.

3

Exploration of Hybrid CTC/Attention Speech Recognition

This chapter explores Gaussian process optimization performed on the speech recognition model detailed in Chapter 2. For this, we trained in total 70 different hybrid CTC/attention models and performed 595 beam search decoding iterations. Its aim is two-fold, parameter space exploration and optimization. The results of this exploration help to derive general training recommendations, and also revisit the key assumption of the hybrid CTC/attention approach stating that CTC provides alignments to the attention mechanism. This chapter summarizes the work published in [8†].

3.1 Gaussian Processes Optimization

Under the assumption that few parameters are more influential than other parameters, sequential optimization methods perform better than random search, by weighting the importance of each dimension [15]. The following investigation uses Gaussian processes as tool for parameter optimization and exploration. Gaussian processes capture function uncertainty at all input points, enabling informed decisions in optimization by balancing exploitation and exploration [3]. In comparison, random brute-force searches don't consider this uncertainty, resulting in possible inefficient evaluations. Gaussian process optimization is a sequential model-based optimization method that was shown to outperform random brute-force search and human performance for many algorithms [79, 172]. Gaussian processes were first applied to optimize hyperparameters in [16].

In the experiment, each training or decoding run was evaluated by the word or character error rate on a test dataset. This evaluation metric is represented by an unknown function f that samples from the discrete parameter set X. Given that continuous functions simplify optimization tasks compared to discrete samples, the following steps demonstrate how these discrete samples are recast to a continuous function. Gaussian processes help to generate continuous functions in a probabilistic manner that estimate f. Optimization is then applied to find the optima, i.e., good parameter configurations.

To formalize this, Gaussian processes estimate the unknown function $f : \mathcal{X} \to \mathbb{R}$. The finite set of points $\{X^{(n)} \in \mathcal{X}\}$ induces a joint Gaussian distribution in $\mathbb{R}^N$. A Gaussian process is described by two functions, the mean-function $\mu : \mathcal{X} \to \mathbb{R}$ and the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We additionally need a suitable kernel and acquisition function for good optimization results [172]. The Matérn-kernel [154] is a generalization of the radial-basis function kernel, defined by

$$k_{\text{Matérn}}(r^{(n)}) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\frac{\sqrt{2\nu}r^{(n)}}{l})^{\nu}K_{\nu}(\frac{\sqrt{2\nu}r^{(n)}}{l}). \tag{3.1}$$

Here, K_ν is the modified Bessel function and $\Gamma(\nu)$ is the Gamma-function. Using $r^{(n)} = ||X^{(n)} - X'^{(n)}||$ as the Euclidean distance between two parameter sets, the kernel characterizes a statistical covariance between sample points. The kernel has two positive parameters, $\nu > 0$ and $l > 0$. Gaussian noise in the target value $f(X^{(n)})$ is modeled by adding a small noise constant ϵ onto the kernel in all sample points.

Hypotheses modeled as a Gaussian process $f_{\text{GP}} \propto \text{GP}(\mu, k)$ are generated based on the previous observations $D = \{(X^{(i)}, Y^{(i)})\}$, $i = \{1, 2, \ldots, n\}$. The acquisition function assists in finding the next optimal point $X^{(n+1)}$ by weighting the mean-function and the kernel. Using the so far minimum observed value f_{min}, the Expected Improvement (EI) metric helps to find a suitable new parameter configuration from the hypotheses:

$$f_{\text{EI}}(X^{(n+1)}) = \mathbb{E}[max(0, f_{min} - f_{\text{GP}}(X^{(n+1)}))|X^{(n+1)}, D], \tag{3.2}$$

This acquisition function is continuous and can be optimized, e.g., by a grid search over its input space [15] in combination with quasi-Newton optimization methods. To avoid local optima, points are randomly sampled as optimization starting point. From those, the quasi-Newton L-BFGS-B optimization [27] is applied on the EI function to find the next optimal point.

3.2 Experiment Setup

Gaussian process hyperparameter optimization is applied on the hybrid CTC/attention network provided in the ESPnet toolkit[i]. The TED-LIUM 2 dataset serves as training recipe and testing benchmark. The experiment was done in two stages: First, optimizing the hyperparameters of the end-to-end model, and second, optimizing the decoding parameters. For each iteration, observations and parameters were passed to the Gaussian process optimizer to obtain the next parameter configuration.

The first stage optimized the parameter configuration of the hybrid CTC/attention architecture. This experiment is based on the standard BLSTMP architecture, as described in Section 1.2, that uses a bidirectional LSTM encoder with projection neurons and an LSTM decoder with location-aware attention. Subsampling was applied to the second and third layer of the encoder, i.e., only every second hidden value is forwarded in these projection layers to the subsequent layer. Parameters along with their upper and

[i] ESPnet version 0.3.0, on git commit hash `716ff54`.

lower bounds are listed in Table 3.1; this includes the network depth and width, and the multi-objective training parameter κ. The parameter κ determines network structure, as CTC-only models did not contain any attention mechanism, while attention-only models omitted the CTC output layer. In total, 70 models were trained in the first stage:

- 20 models were trained beforehand to provide the starting observations.
- Then, 40 models were trained with optimized parameters. A decoding run without language model with each model was performed. Here, the model CER served as optimization target.
- This stage included ten further models, optimizing towards the accuracy of the attention mechanism on the validation set. This performance metric is already available during training, which is an advantage, as no additional decoding run is needed to derive it.

Table 3.1: Optimized hyperparameters of the model training and decoding stages for the Gaussian Process optimization.

Stage	**Parameter**	**Symbol**	**Range**
Model training	Encoder layers	N_{enc}	$[1; 6]$
	Decoder layers	N_{dec}	$[1; 6]$
	Encoder layer width	W_{enc}	$[25; 400]$
	Projection units	N_{dec}	$[25; 400]$
	Decoder layer width	W_{dec}	$[25; 400]$
	Attention layer width	W_{att}	$[25; 400]$
	CTC weight	κ	$[0.0; 1.0]$
	Attention channels	N_{att}	$[1; 20]$
	Attention filters	N_{K}	$[30; 150]$
Beam search decoding	Language model weight	β	$[0.0; 1.0]$
	CTC weight at decoding	λ	$[0.0; 1.0]$
	Language model index		$[1; 4]$

The second stage optimized beam search parameters, such as CTC weight and language model weight, as in Tab. 3.1. Additionally, the optimizer also could choose from four different RNNLM language models; each with 2-layers to 650 LSTM units and a perplexity on the text corpus ranging from 3.32 to 4.02. We set the CER on the TED-LIUM 2 dev set as target value. This stage included the hybrid models of the previous stage. To obtain the first decoding results, decoding was done with and without language models, with the same CTC weight during the training, i.e., $\lambda = \kappa$.

3.3 Experimental Results

Overall, 70 models were trained and results were gathered from 590 decoding runs. Compared with the baseline model that achieved 10.1% CER on the TED-LIUM 2 test set [195], our best model achieved 8.9% CER, an absolute improvement of 1.2% CER. As non-hybrid models have a κ value at the borders of its interval, and starting values were already provided at these values, the algorithm exclusively chose hybrid model and decoding configurations.

A few parameter groups emerged from the results, distinguishable by their CTC weight at decoding and language model weight. Figure 3.1a displays these groups on a CER-WER plot. Most beam search results are between 10% and 20% CER. From inspection of decoding runs with higher error rates, two parameter groups were clearly distinguishable from the other results:

Group 1: High WER but relatively low CER. This group mostly contained results from CTC-only decoding without a language model, i.e. $\lambda > 0.8; \beta < 0.1$, that may be explained by the properties of CTC: CTC-based networks generate temporal probabilities for each character. When those network activations are seen as a sequence, sometimes characters are switched or omitted. This group characteristically exhibits spelling errors within words, or sometimes shifted word boundaries. Each wrongly spelled word causes one word error, each shifted space token two word errors. Temporal information of word writing may be distributed unevenly within a word, thus it is usually combined with a sequence-based language model.

Group 2: Low WER but relatively high CER. Attention-only decoding in combination with an RNNLM language model, i.e. $\lambda < 0.05; \beta > 0.3$, exhibits word loops and dropped utterance parts, but mostly correct spelling. By intuition, attention-based decoding that is supported by a language model should improve performance, however, in the experiment this was not the case and adding the RNNLM in the beam search deteriorated results. Possible causes for this effect are discussed in section 3.4.

By manual inspection, there was no single parameter that strongly influenced the CER. Still, larger networks have a tendency to perform better, resulting in a weak inverse correlation of the number of learnable parameters with the CER, as pictured in Figure 3.1b.

Figure 3.1c separates these results into further categories by distinguishing hybrid, CTC-only, and attention-only models and beam search runs. (1) includes all decoding runs, and (2) and (3) show the performance of hybrid models with and without language model. Categories (4) and (5) show attention-only beam search results of hybrid models; their performance was clearly better without RNNLM (4) than with it (5). A similar effect was observed on models that were only trained with attention mechanism, again, with a better performance without RNNLM (6) than with it (7). Categories (8) and (9) show the CER of hybrid models that were decoded solely with the CTC output of the network. Again, CTC-only beam search runs with hybrid models yielded better CER with RNNLM (8) than without it (9). A similar result was observed with CTC-only models in categories (10) and (11).

The best performance in the experiment, however, was only achieved in a hybrid

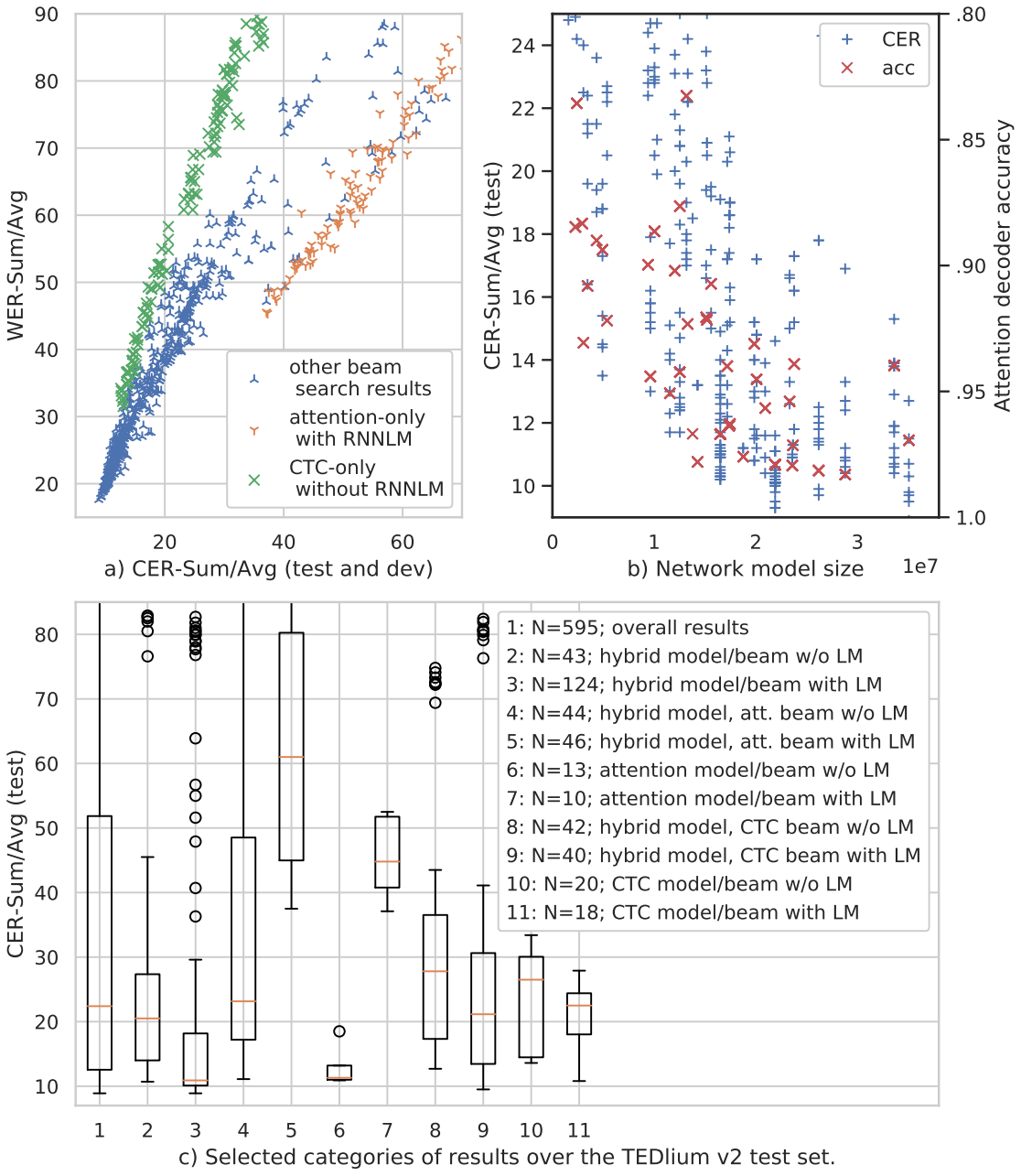


Figure 3.1: TED-LIUM 2 exploration results overview as discussed in Sec. 3.4.
(a) shows three parameter groups in the WER-CER scatter graph: Attention-only beam search with RNNLM $\rightarrow \lambda < 0.05; \beta > 0.3$. CTC-only beam search without RNNLM $\rightarrow \lambda > 0.8; \beta < 0.1$; (b) demonstrates how larger networks tend to have a higher performance; manifesting in a weak inverse correlation between network size and the CER; (c) lists parameter categories[a] and their CER distributions.

[a] In detail: Hybrid models $\rightarrow \kappa \in]0.0; 1.0[$, attention-only models $\rightarrow \kappa = 0.0$, att.-only beam search $\rightarrow \lambda = 0.0$, CTC-only models $\rightarrow \kappa = 1.0$, CTC-only beam search $\rightarrow \lambda = 1.0$, 'w/o LM' $\rightarrow \beta = 0.0$, 'with LM' $\rightarrow \beta = 1.0$.

Table 3.2: Comparison of best results from the parameter space exploration in selected categories. The columns 'without LM' or 'with LM' indicate that the language model weight is set to either $\lambda = 0.0$ or $\lambda > 0.0$.

	baseline	with LM			without LM		
Parameter	[195]	hybrid	att.-only	CTC-only	hybrid	att.-only	CTC-only
Encoder layers	6	6	6	6	4	6	6
Encoder BLSTM cells	320	485	400	400	497	400	400
Projection units	320	292	320	320	377	320	320
Decoder Layers	1	5	2	2	3	2	2
Decoder LSTM cells	300	352	100	400	364	100	400
Attention neurons	320	379	100	350	172	100	350
Att. channels in K	10	2	10	10	10	10	10
Att. filters in K	100	122	100	100	128	100	100
Multi-obj. (training) κ	0.5	0.69	0.00	1.00	0.15	0.00	1.00
Model size ($1e6$)	18.7	35.1	23.6	26.6	28.8	23.6	26.6
RNNLM weight β	1.0	0.73	0.41	1.00	0.00	0.00	0.00
Multi-obj. (beam) λ	0.3	0.62	*0.08*	1.00	0.15	0.00	1.00
TED-LIUM 2 dev/CER	10.8	**9.2**	37.1	11.3	10.4	10.4	13.7
TED-LIUM 2 dev/WER	19.8	**18.5**	47.1	23.3	22.5	22.4	36.5
TED-LIUM 2 test/CER	10.1	**8.9**	40.2	11.3	10.6	10.9	14.4
TED-LIUM 2 test/WER	18.6	**17.6**	49.3	22.6	22.1	22.4	36.9

decoding approach with a language model, as shown in Figure 3.1c, item (3). The results of this figure are further quantified in Tab. 3.2. Notably, the columns of attention-only or CTC-only models contain parameter configurations that were handpicked, due to the algorithmic preference of hybrid models and beam search runs.

3.4 On Word Loops and Dropped Sentences

In the experiment, attention-based shallow fusion decoding deteriorated performance when using a language model. This is not intuitive and also stands in contrast to previous publications; those successfully combined attention-only models with a language model [31, 183]. There may be possible explanations of this deteriorated performance:

Different token distributions in training and decoding: Due to train-test-mismatches in sequential training, the ASR network may be less prepared for unseen letter sequences. In the experiment, only ground truth letters are fed into the network as previous letter hypothesis y_{l-1}; this method is called teacher forcing [197]. Teacher forcing trains the model only with the ground truth sequence, and thus, the model will generate sequences at inference that were not seen during training. This leads to a discrepancy between training and inference distributions and to an accumulation of errors. Scheduled sampling [14] solves this by randomly selecting the previous letter hypothesis y_{l-1}. For example, the *listen-attend-spell* architecture [31] uses scheduled sampling; that architecture was discussed in Section 2.4.3 as a precursor to the investigated RNN model and is based on a content-based attention mechanism. Attention-only decoding with RNNLM of the test set introduces such unexpected letter hypotheses. This effect may explain an amplification of small errors in the beginning of the generated sequence, however, does not yet explain how word loops and missing sequence parts are generated. Other factors may also contribute to this issue, as explained in the following paragraphs.

Delayed information propagation in the attention mechanism: Architectural differences of the attention mechanism may impact the decoding. As a general observation, the attention mechanism uses a weighted sum to extract information about the next-in-sequence token, which depends on the temporal location within the encoded sequence of the next possible character. Attention does not have a strong temporal alignment such as CTC, and thus, the attention focus does not always progress linearly in time, depending on spreading of intra-word alignments or pronunciation.

The investigated CTC/attention architecture uses the location-aware attention mechanism, as discussed in Section 2.4.4. While the letter hypotheses, i.e., which character was chosen for y_{l-1}, are given as feedback to the network, the next-in-sequence attention weights may be shifted wrongly due to delayed sequential processing: The next-state q_l depends on the previous state vector q_{l-1}, and also previous attention weights $a_{l-1,t}$. The attention focus may need readjustment for different hypotheses, e.g., for an unexpected letter hypothesis y_{l-1}. But the information of the chosen hypothesis y_{l-1} arrives at the attention state not at the current l but with an additional delay $l+1$, because the context vector c_l at position l can not consider the information of the previous letter y_{l-1}. Instead, information of the chosen hypothesis propagates to the state vector only

at time step $l+1$, and only then impacts the attention weights at time step $l+2$. These delays of information flow may accumulate to a feedback loop due to a cyclic progression: Hypotheses with a feedback loop may then have higher posterior probabilities $p(y_l)$ based on the estimated c_l and the preceding letter hypothesis. Wrong estimations of the letter probabilities then may accumulate to a wrong decoded sequence with a wrongly estimated high confidence.

At the time of the experiment, the beam search did not contain any regulating mechanisms such as a penalty to decoded sequence length. Later, the default parameters of the hybrid CTC/attention architecture were extended with two regulating parameters[ii], setting an interval for the ratio of input length to output length. A sensibly chosen interval significantly reduces word loops or dropped sentence parts.

Low-perplexity language models: The language models itself may not be performant enough. In a later inspection of RNN language models, many of them produced word loops themselves without any influence of an ASR model. Experiments on the LibriSpeech dev set [138] by Karita [99] demonstrated how language model perplexity can be improved from 56.5 to roughly 30.6 when switching from an RNNLM (similar to ours, 2 layers with 1024 LSTM units) to a large Transformer language model (18 layers, 2048 units).

Even large language models can exhibit repetitive loops, a phenomenon termed as neural text degeneration [36, 85, 204]. Repetitive loops also occur in ASR Transformers, but RNN networks seem to be more prone to it [100].

3.5 Revisiting the Hybrid CTC/Attention Hypothesis

The motivation to jointly train and decode the CTC network with an attention mechanism was introduced by Watanabe *et al.* [196], to provide temporal alignments to the attention mechanism. Using the experimental results of the parameter exploration, we can evaluate this notion.

For this, we compare the hybrid model with joint beam search in Figure 3.1c item (2) with the attention-only approach (6). Transcriptions of both classes had mostly correct alignments, and errors were dominantly introduced by similar-word mistakes. In item (6), attention-only beam search, lost alignments were rare - without needing CTC as alignment regularization. Another indication of this is that attention-only beam search with hybrid models (4) did not yield better results than with attention-only models (6).

Here, the results did not prove the advantage of combining CTC with an attention architecture, but rather that attention models alone already achieve acceptable performance. Still, the best performance in the experiment was achieved in a hybrid decoding approach with a language model with a CTC weighting of 0.63 and a language model weighting of 0.73. Even with larger attention models, misspellings and similar-word substitutions can only be reduced to a minimum by joint decoding with a language model.

[ii] The two parameters are *minlenratio* and *maxlenratio* in ESPnet 1.

Nevertheless, in combination with a language model, adding CTC to the attention for the beam search regularizes and provides temporal alignments.

4

Adversarial Machine Learning

There is a general trend from ASR systems with handcrafted features towards deeper neural networks that are trained in an end-to-end manner. Deeper neural networks are also more complex and, in many cases, susceptible to adversarial noise: Adversarial noise is a specially crafted disturbance that is optimized to interfere with the classification of a neural network. In some cases, even a small amount of such noise is sufficient to cause the network to falsely classify its input [180, 202]. Samples with adversarial noise are called adversarial examples.

From a security perspective, misleading neural networks into falsely classifying their inputs has direct implications for the security of systems with ASR functionality. Personal assistants that are prone to adversarial examples constitute a potential attack surface [201]. A TV show or a sound may play a barely noticeable adversarial example to issue hidden voice commands [29, 30], e.g., to unlock a door or to order a product.

Most importantly, the performance of speech recognition systems should closely align with human hearing and comprehension. ASR systems should be robust against types of noise that are nearly imperceptible to humans. To solve this, we will describe later in this chapter how to improve the robustness of the model, by using adversarial training that infuses adversarial examples during the training process.

This chapter combines the work published in two publications [$1^{\dagger}$, $7^{\dagger}$], building on top of two master's theses by Andronic [1^{+}] and Chavez Rosas [5^{+}], and focuses on adversarial examples in hybrid CTC/attention ASR, adversarial training and corresponding countermeasures. All experiments and benchmarks are conducted using the RNN-based architecture discussed in Section 2.6. The first section gives an overview of related work of adversarial examples, and countermeasures such as adversarial training. Then, the following section presents several methods to generate untargeted adversarial examples in the feature domain for the hybrid CTC/attention ASR system. The third section demonstrates how these adversarial examples can be incorporated back into the training process through a technique called adversarial training, which serves as a countermeasure against adversarial noise. At the same time, adversarial training improves generalization in the conducted experiment and improves the speech recognition performance by a relative 10% compared to the baseline model. While audio adversarial examples are commonly generated in the feature domain, the fourth section discusses a method to

directly generate adversarial examples in the time domain. The last section investigates MP3 compression as a successful countermeasure against time domain adversarial examples. Adversarial noise often includes inaudible components, which are reduced by MP3 compression.

4.1 Related Work

Before examining the effects of adversarial machine learning on the end-to-end ASR architecture, this section conducts a review of the literature in the field of adversarial machine learning, with a focus on *whitebox* models, where the parameters θ are fully known.

4.1.1 Adversarial Machine Learning

Adversarial machine learning, i.e., adversarial noise and adversarial examples, were first described in the image recognition domain by Szegedy *et al.* [180], and then further investigated by Kurakin *et al.* [109, 112].

This chapter focuses on a white box model scenario. In adversarial machine learning, a white box model refers to a machine learning model whose parameters θ and internal workings are fully known to an adversary. In contrast to black box models, where the inputs and outputs of the model are known, white box models allow adversaries to directly manipulate the model's behavior and optimize attacks. Therefore, white box models are often more vulnerable to adversarial attacks than black box models.

Most methods to generate adversarial noise depend on computing the gradient of the network with respect to its input [4]. The most popular method of generating adversarial noise for a neural network with known parameters is the Fast Gradient Sign Method (FGSM) [66], of which a formal description will be given in Section 4.2. Similar to stochastic gradient descent, it is gradient-based and may be used as a one-step method or applied iteratively. This method can generate untargeted adversarial examples, causing the network to misclassify, or targeted adversarial examples that intentionally mislead the network to classify an input into a specific, incorrect class [28].

The perturbation introduced to create adversarial examples for speech recognition networks can be minor, remaining imperceptible to human listeners [191]. However, this subtle alteration can be significant enough to cause system misclassification. From an adversarial's perspective, the amplitude of the adversarial perturbation should be lower or below the hearing threshold compared to the original input x_t, ensuring it remains unnoticeable.

A method for defending against white-box attacks involves the use of adversarial training, where the model is trained on both clean and adversarial examples, forcing it to learn robust features that are less susceptible to attacks. Adversarial training also has to take label leaking [111] into account: Untargeted adversarial examples, which are generated using the ground truth, contain information about that ground truth.

Consequently, the network might inadvertently learn to predict the correct labels based on these adversarial examples during the adversarial training process.

4.1.2 Audio Adversarial Examples

While most work on adversarial machine learning focuses on image recognition, adversarial examples in the time domain for speech recognition are more difficult [44], due to the use of preprocessed acoustic features and temporal dynamics [206]. A general introduction to adversarial attacks on ASR systems and countermeasures is found in [89]. Audio adversarial examples are even transferable between architectures [2, 131, 185].

Adversarial examples are very relevant to ASR systems, as they are not just a theoretical construct but have been shown to work in real-world physical environments [210]. Carlini *et al.* [29] demonstrated working hidden voice commands in a physical room against HMM-based ASR systems, being transferable and functional across a wide range of simulated rooms and environments. For this, they simulated the room impulse response (RIR) in the construction of the adversarial noise and included the RIR in the optimization process. Their adversarial examples also were transferable on the Google Assistant on the phone when played over-the-air.

In the same publication, Carlini *et al.* [29] were also the first to publish adversarial machine learning on conventional hybrid DNN/HMM ASR. Another significant work on adversarial machine learning for hybrid DNN/HMM speech recognition is the Backstitch algorithm [192], an adversarial training approach that modifies the original SGD optimizer to reduce the sample bias of the training process. This algorithm takes advantage of specific properties of the Natural Gradient [5, 149], which is used in the kaldi toolkit but not for hybrid CTC/attention ASR.

CTC-based adversarial examples were first constructed based on DeepSpeech by Hannun *et al.* [76] using the FGSM. Carlini *et al.* [30] later successfully constructed adversarial examples for longer sentences on DeepSpeech [76]. However, their constraint for minimizing the added adversarial noise did not consider the limits and sensitivities of human auditory perception, so the introduced perturbations could still be noticed by a human.

Audio adversarial examples on sequence-based ASR systems were first demonstrated for the attention-based system Listen-Attend-Spell [31] by Sun *et al.* [176] who apply FGSM to the sequential attention decoder. In a similar fashion, adversarial training feeds the corresponding sequence-based adversarial examples to the model as a form of data augmentation.

Some works focus on making adversarial examples inconspicuous; see Sharif *et al.* [168] in the domain of face recognition. Similarly, audio adversarial examples may be made undetectable by the human hearing using psychoachustic hiding [150, 164]. This technique restricts the adversarial noise to the spectral envelope of the human hearing threshold. The human hearing threshold may be derived based on MP3 compression [$1^{\dagger}$, 1^{+}]. Schönherr *et al.* [164] were the first to develop imperceptible audio adversarial examples for conventional hybrid DNN-HMM ASR. As demonstrated in a subsequent publication [165], audio adversarial examples with psychoacoustic hiding can also be constructed to be transferable

between architectures and successful in over-the-air scenarios. Vadillo *et al.* discuss suitable metrics to evaluate human perceptability of adversarial examples [186]. Further methods have been proposed in the literature for hidden adversarial attacks that, however, are not relevant for our investigation into ASR systems, e.g., the DolphinAttacks [214] leverage nonlinearities of microphones and modulated their audio signal on an inaudible ultrasound carrier.

As most ASR systems build on MFCCs or F-bank features, their audio adversarial examples are in feature domain, not in time domain. Features over time have a similar representation as spectrogram and thus can be treated with approaches to adversarial machine learning derived from computer vision [89]. Due to the lossy compression of the feature extraction step, it is non-trivial to transform adversarial features back to the time domain. Previous works demonstrate various approaches to this issue [92], but with relative success, as the transformed audio sounds drastically different from the original. Andronic *et al.* describes a method to generate such adversarial examples in time domain [1^{+}, $1^{\dagger}$] that will be discussed in Section 4.4.

4.1.3 Countermeasures Against Adversarial Examples

Similar to adversarial machine learning, most research on countermeasures against adversarial machine learning has been done in the computer vision domain. Many of those techniques inspired works in the ASR, that Hut *et al.* [89] categorized into reactive and proactive strategies. Proactive approaches harden the ASR network during training using adversarial training [177] or distillation [139]. Reactive approaches reduce or detect adversarial noise with an already trained ASR network using audio processing methods, such as compression or down-sampling. Local smoothing, down-sampling were shown to be effective for short utterances [207]. Band-pass filtering and compression (MP3 and AAC) and more complex speech coding algorithms (Speex, Opus) mitigated adversarial noise on a keyword spotting system [153]. Similar to the work in Section 4.4, Das *et al.* [48] used MP3 compression to mitigate targeted adversarial examples in the CTC-based DeepSpeech model.

4.2 Generation of Adversarial Examples

This section introduces four methods to generate adversarial noise using the FGSM [66, 5^{+}] on the hybrid CTC/attention architecture: 1. At the attention decoder at a fixed point in the decoding sequence, 2. at the attention decoder using an averaged gradient over several points in the decoding, 3. from the CTC loss, and 4. a combination of attention-based and CTC-based loss.

4.2.1 Adversarial Examples in Feature Domain

The adversarial noise $\delta(x_t)$ is generated for a specified sample $X = x_{1:T}$ in feature domain. To obtain the adversarial example, this noise is then added onto the sample:

$$\hat{x}_t = x_t + \delta(x_t), \qquad \forall t \in [1, T]. \tag{4.1}$$

The FGSM method uses backpropagation, implying the consideration of a whitebox model with known network parameters. The generation of adversarial examples is restricted to untargeted types, meaning the adversarial noise optimizes for the increase of the objective function $\mathcal{L}_{\text{CE}}(X, y; \theta)$, given a token sequence y and the parameters of the model θ. Analogous to the learning rate of SGD, FGSM can be applied in an iterative manner weighted by the factor ϵ that controls the intensity of the adversarial noise.

For T time steps utilizing the ASR model, a decoded sequence $\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_L\}$ is deduced from $X = \{x_1, \cdots, x_T\}$. Subsequently, this reconstructed label sequence is used for generating adversarial noise instead of the ground truth to avoid label leaking.

4.2.2 Attention-based Static Window Adversarial Examples

Attention-based decoding uses a beam search, as discussed in Section 2.66. Sun *et al.* [176] were the first to propose an application of the FGSM method to sequence-to-sequence models. They derived the sequential attention cross-entropy loss $\mathcal{L}_{\text{CE}}(X, y_l; \theta)$ with respect to the feature sequence X by iterating over sequential token posteriors. Then, the adversarial noise $\delta(x_t)$ is calculated for the full decoded sequence as

$$\delta_{\text{SW}}(x_t) = \epsilon \cdot \text{sgn}(\nabla_{x_t} \sum_{l=1}^{L} \mathcal{L}_{\text{CE}}(X, \hat{y}_l; \theta)), \quad l \in [1; L]. \tag{4.2}$$

This method applies the FGSM on the gradient of all sequence steps equally, and thus equally distributes adversarial noise over the full audio utterance.

The latter method calculates a gradient of all sequential outputs and can be further reduced using the autoregressive properties of the attention mechanism; Under the assumption that errors propagate to subsequent decoding steps, only a few steps are needed to be calculated. As discussed in Section 3.4, attention-based decoding is prone to disruption under certain circumstances, for example, word loops when the attention focus is lost. It thus is already sufficient to target a part in the sequence, that not only distorts the corresponding decoded token but also may distort further decoding. Localized adversarial loss is generated from a single decoding step or an interval of steps, by applying FGSM on a *static window* of w gradient steps from the starting token at index l_γ up to the ending token at index $l_{\gamma+w}$:

$$\delta_{\text{SW}}(x_t) = \epsilon \cdot \text{sgn}(\nabla_{x_t} \sum_{l=\gamma}^{\gamma+w} \mathcal{L}_{\text{CE}}(X, \hat{y}_l; \theta)), \qquad l \in [1; L]. \tag{4.3}$$

4.2.3 Attention-based Moving Window Adversarial Noise

Some audio segments are more susceptible to adversarial noise than others, as they are given a higher weight by the attention mechanism. In many cases, the static window method of Equation 4.3 changed only a localized part of the transcription. To generate noise that affects the full transcription, the static window of adversarial noise can be repeated in intervals. By accumulating the obtained gradients, localized maxima of adversarial noise are reduced and dissipated over the utterance.

The moving window approach uses the same window as the static window approach with length w and repeats it with stride ν to obtain an accumulated gradient $\nabla_{\mathrm{MW}}(x_t)$. The iterative gradient accumulation can lead to high variances, which can be balanced using gradient normalization [53], i.e., the gradient is normalized by the $L1$ norm of itself. With that, the adversarial noise $\delta_{MW}(x_t)$ for the *moving window* is determined as:

$$\nabla_{\mathrm{MW}}(x_t) = \sum_{i=0}^{\lceil L/\nu \rceil} \left(\frac{\nabla_{x_t} \sum_{l=i\cdot\nu}^{l_w+i\cdot\nu} \mathcal{L}_{\mathrm{CE}}(X, \hat{y}_l; \theta)}{||\nabla_{x_t} \sum_{l=i\cdot\nu}^{l_w+i\cdot\nu} \mathcal{L}_{\mathrm{CE}}(X, \hat{y}_l; \theta)||_1} \right), \quad l \in [1; L] \tag{4.4}$$

$$\delta_{MW}(x_t) = \epsilon \cdot \mathrm{sgn}(\nabla_{\mathrm{MW}}(x_t)) \tag{4.5}$$

Sun *et al.* [176] apply the adversarial gradient on each sequence element. Due to sequential dependencies in the decoding process, applying the gradient on all steps may not always be necessary. Skipping certain parts of the sequence reduces the computational overhead and still generates an effective adversarial example.

4.2.4 Adversarial Noise from CTC Loss

With CTC, the loss function is applied in a single step upon the full utterance, as explained in Section 2.4.6. It then suffices to apply the FGSM on CTC loss $\mathcal{L}_{\mathrm{CTC}}$ over the reconstructed label sentence $\hat{Y}$:

$$\delta_{\mathrm{CTC}}(x_t) = \epsilon \cdot \mathrm{sgn}(\nabla_{x_t} \mathcal{L}_{\mathrm{CTC}}(X, \hat{Y}; \theta)). \tag{4.6}$$

4.2.5 Hybrid CTC/Attention Adversarial Noise

To obtain hybrid CTC/attention adversarial noise, both attention δ_{att} and CTC δ_{CTC} adversarial noise are fused together in the form of multi-objective optimization [119] with the weighting parameter $\xi \in [0; 1]$.

$$\delta_{\mathrm{Hybrid}}(x_t) = (1 - \xi) \cdot \delta_{\mathrm{att}}(x_t) + \xi \cdot \delta_{\mathrm{CTC}}(x_t), \quad \forall t \in [1, T] \tag{4.7}$$

This multi-objective weighting of adversarial noise, derived from both CTC loss and attention loss, mirrors the multi-objective structure of the hybrid CTC/attention architecture. The adversarial noise for the attention mechanism may be derived using the windowed methods of Equations 4.3 or 4.4, or also derived from the full sequence as in

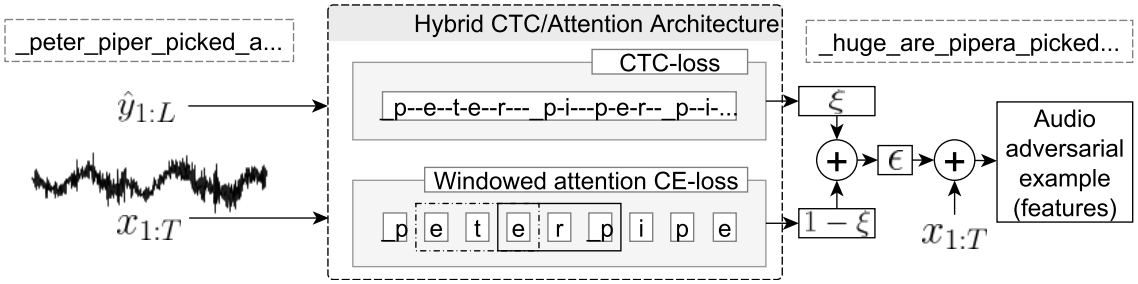


Figure 4.1: Generation of hybrid CTC/attention adversarial examples. The reconstructed label sequence and feature vector $x_{1:T}$ serve as initial input data. With that, the hybrid CTC/attention model performs an inference and determines attention and CTC losses. Adversarial noise is then generated from the windowing method on the attention loss and the FGSM on the CTC loss. Both noise vectors are added together, weighted by the weighting parameter ξ and scaled by the noise factor ϵ. This results in the adversarial noise that is added onto the feature vector $x_{1:T}$ to obtain the adversarial example.

Equation 4.2. The generation process of hybrid CTC/attention adversarial examples is shown in Figure 4.1. Several enhancements to this algorithm are plausible. It can be augmented to produce targeted adversarial examples and iteratively refine the adversarial noise, as proposed by Carlini *et al.* [30]. Additionally, the integration of psychoacoustic hiding is feasible, allowing the adversarial noise to be limited to the psychoacoustic hearing threshold in either the spectral or feature domain [164, 1^{+}].

4.2.6 Generated Adversary Examples

This section demonstrates the impact of adversarial examples on decoding results based on two noise-free sentences. Noise-free audio rules out interference of other noise sources and improves decoding performance.

The first example sentence was obtained from the Tacotron text-to-speech toolkit [169] as an example sentence:

Peter Piper picked a peck of pickled peppers.
How many pickled peppers did Peter Piper pick?

The second example sentence was generated using the Tacotron text-to-speech model of the ESPnet toolkit[i]. Their model was trained on the LJ Speech Dataset [93] based on audio of the LibriVox project [117] and is available on the ESPnet project page as *ljspeech.tacotron2.v2*. The ground truth of the sentence "Anie" is:

Anie gave Christina a present and it was beautiful.

The adversarial examples are calculated using the pre-trained model *tedlium2.rnn.v1* provided by the ESPnet toolkit, an RNN-based ASR network that was trained on the TED-LIUM 2 dataset[ii]. Inference is done without a language model. With that, the

[i]The model is available on the ESPnet project page as *ljspeech.tacotron2.v2*.

[ii]The hybrid CTC/attention model *tedlium2.rnn.v1* has a four-layer encoder with unidirectional LSTM units and projection neurons. Its decoder uses the location-aware attention mechanism of Section 2.46 and has only a single layer. Each layer has 1024 neurons. Its location-aware attention

two sentences “Anie” and “Peter” are recognized as:

peter piper picked a peck of **pickle** peppers
how many **pickle** peppers did Peter Piper **pack**
any gave christina a present and it was beautiful

Recognition errors are printed in bold. While the transcription is very similar in its pronunciation compared to the ground truth, it contains a few errors that are explained by the lack of a language model. These two decoded sentences serve as the reference to generate adversarial examples to avoid label leaking.

The subsequent paragraphs demonstrate adversarial examples and their impact on the decoded sentence. Throughout these examples, the additive weight of the noise to the example is consistently set to $\epsilon = 0.3$, a value also used by Sun *et al.* for adversarial training [177]. The CTC weight parameter is uniformly set to $\xi = 0.5$ for all hybrid CTC/attention examples.

First, adversarial examples that are obtained using the FGSM method from CTC loss. The CTC-based adversarial examples result in the following transcription:

peter piper **a** picked a **pack** of **tackled tappers**
how many **piggle peppers didn't pay their** piper pick
any **dove** christian **no presented** it was beautiful

Here, the word errors caused by adversarial noise are printed in bold. By inspection, errors caused by CTC-based adversarial noise are distributed across the sentences.

While the CTC-based adversarial noise is applied globally on the utterance, the static window method applies FGSM at only certain decoding steps at decoding. Table 4.1 shows the sentences “Anie” and “Peter” with static window adversarial examples. This noise mainly causes a localized disturbance on the tokens at these steps, however, also effects decoding of the whole utterance; for example, the static window was applied to four tokens of “Peter”, and the adversarial example achieved an error rate of 87.5%. Table 4.1 also includes the hybrid adversarial example that has a slightly lower WER of 56.3%.

The moving window method applies adversarial noise across the utterance and thus avoids localizing the adversarial noise. Table 4.2 lists the adversarial examples generated with the moving window method. In this example, applying the static window method on “Peter” achieves a WER of 93.8% w.r.t. the ground-truth; this is a higher error rate than with the static window method. However, this example also uses more than 20 tokens in the calculation, compared to only four tokens in the static windowing example. The hybrid adversarial example achieves an error rate of 56.3% WER with the same parameter configuration.

Adversarial noise changes the recognized sentence and adds additional recognition errors. Examples generated with the moving window exhibit higher error rates than compared with static window or CTC-based adversarial examples. The inspection of “Anie” and “Peter” makes it evident that the moving window method created the most effective perturbations; thus it is a suitable choice as augmentation method in adversarial

mechanism is used with five channels and 100 filters. The training weight of the CTC loss is set to $\alpha = 0.5$. The model was trained for 10 epochs.

Table 4.1: Recognized sentences with the static window method from attention-only and hybrid adversarial examples. Word errors caused by adversarial noise are printed in bold. The attention-based adversarial noise fo the static window acts as a localized distortion on certain tokens; those affected tokens are underlined in the text.

Adv. Example Type	Recognized Sentence
$\hat{y}$ with $l_w = 3$, $\gamma = 4$	any gave christina a present and it was beautiful
Attention-only	any **game christian out priasant** and it was beautiful
Hybrid, $\xi = 0.5$	any **game christian** a present and it was beautiful
$\hat{y}$ with $l_w = 4$, $\gamma = 26$	peter piper picked a peck of pickle peppers. how many pickle peppers did peter piper pack
Attention-only	**either** piper **cricker ticket tickless techners came a typical turkished plea** piper pick
Hybrid, $\xi = 0.5$	peter piper picked a **pick** of **tickle tappers** how many **tickle tapper stood plea piper** piper pick

Table 4.2: Recognized sentences with the moving window method from attention-only and hybrid adversarial examples. Word errors caused by adversarial noise are printed in bold. The attention-based adversarial noise acts as a localized distortion on certain tokens; those affected tokens are underlined in the text.

Example type	Recognized Sentence
MW, $l_w = 2$, $\nu = 3$	any gave christina a present and it was beautiful
MW	any **canada's crystall out current since** and it was –
Hybrid MW	any **gaitians crystain out** a present and it was beautiful
MW, $l_w = 3$, $\nu = 3$	peter piper picked a peck of pickle peppers. how many pickle peppers did peter piper pack
MW-AAE	**huge her** piper **okapk pickple take her techners harmony pittle tipers stayed peter paper** pick
Hybrid, $\xi = 0.5$	**feater** piper **a picked depic of tapled tapper** how many **pickles pepper state** peter piper pick

training.

4.2.7 Adversarial Examples in Time Domain

Previous sections described adversary examples only in feature domain. This section shows how to generate adversarial examples in time domain.

Adversarial noise is generated only for the feature domain, as the neural network uses features as its input. Feature inversion converts the adversarial example into time domain [20]. Feature inversion algorithms are provided by audio libraries, such as the *Librosa* toolbox [124].

Figure 4.2 visualizes the steps from the original recording to the audio-domain adversarial example. Feature extraction is applied on an audio recording as a first step to generate its adversarial example. Using a pre-trained neural network, adversarial noise is generated that is added onto the original features. Lastly, a feature inversion algorithm obtains the adversarial example in time domain. The feature inversion step involves approximately inverting the non-bijective Mel frequency weighting, and then an application of the Iterative Short-Time Fourier Transform (iSTFT) in combination with the Griffin-Lim algorithm to reconstruct the phase information of the audio signal [143].

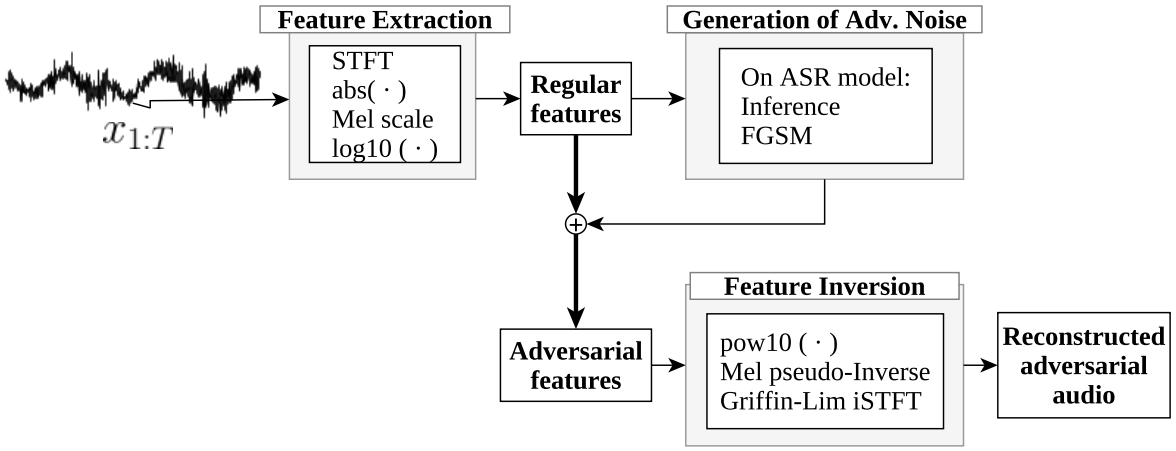


Figure 4.2: Generation of adversarial examples in time domain. Initially, feature extraction is employed on an audio recording to obtain F-bank features. A pre-trained neural network is then utilized to generate adversarial noise, which is subsequently added to the original features. A feature inversion algorithm reconstructs the adversarial example in the time domain.

This approach, however, has certain inherent limitations: The feature extraction is lossy, as it discards phase information and compresses spectral information. It is thus not possible to achieve an exact reconstruction of the original audio; reconstructed audio contains audible artefacts. Second, pitch information is not included in the feature inversion as a limitation of the inversion algorithm.

An experiment was conducted to compare raw audio to reconstructed audio, evaluating whether the relevant acoustic features are still preserved. This experiment was performed with an RNN model described in Section 2.6. The training and decoding parameters for

the model[iii] are based on the default network configuration for the VoxForge dataset in ESPnet v0.8.0 [129]. All conducted experiments utilized the VoxForge [190] dataset, a modestly-sized open-source speech database comprising brief, basic sentences spoken by various volunteers in 17 languages. Only the English portion of the dataset was used for training and analysis, containing approximately 130 hours of audio from 1234 speakers. This section was further divided, allocating 80% for training and 10% each for validation and testing. The audio files are 16-bit encoded wav files with a 16kHz sampling rate. This dataset's uniqueness lies in its availability in an uncompressed format, permitting additional compression and ensuring the elimination of any prior compression artifacts.

To obtain a comparison set that contains reconstructed audio, feature extraction and then feature inversion is applied to the test set. The performance of the model is then evaluated on both the original and reconstructed test set. In the experiment, the reconstructed audio had an absolute increase of 1.3% CER.

4.3 Adversarial Training for Robust Models

Adversarial training augments regular training samples with adversarial noise, effectively extending the training data. This section explores the potential additional benefit of this augmentation. Specifically, it examines whether the integration of adversarial examples within the same amount of training data contributes to enhanced model performance. The broader goal of adversarial training is to improve the robustness of the model against adversarial noise, and as discussed later, it also improves generalization and robustness against regular noise. This section described the experiment setup, demonstrates the effects of adversarial noise using two case studies, and then discusses the effect of adversarial training on model robustness for noisy datasets.

4.3.1 The Adversarial Training Algorithm

The adversarial training algorithm in this section is based on the *adversarial regularization* described by Sun *et al.* [176]. After a warm-up period of N training epochs, the training data is augmented with adversarial examples. The algorithm decides for each mini-batch with a predefined probability p_a whether to augment the current mini-batch with adversarial noise.

However, two properties to this algorithm are changed for this experiment: Sun *et al.* include a weighting factor α to distinct sequence components; this weighting factor is not used in this work and the adversarial noise is derived from the current mini-batch, i.e., one inference at a time. Kurakin *et al.* [111] proposed sampling the perturbation step size ϵ to avoid overfitting to adversarial examples. This experiment also includes

[iii]The VGGBLSTMP architecture used involves a convolutional VGG_2-Net, followed by a 4-layer bidirectional LSTM encoder, in which each layer has 320 units and projection neurons. The decoder is composed of a single layer with 300 units. The multitask learning parameter α is set to 0.3. A maximum of 25 epochs was used with a patience of 3. During training, a scheduled sampling with a 0.5 probability was applied. Complete parameters' configuration can be found in [1^+].

their improvement and thus, the perturbation step size ϵ is randomly sampled for each mini-batch.

At training, the expanded sequential loss function that includes the augmented adversarial noise is applied to the model and determined according to Goodfellow*et al.* [66] as

$$\widehat{\mathcal{L}_{\mathrm{CE}}}(X, y; \theta) = \sum_i (\mathcal{L}_{\mathrm{CE}}(X, y_i; \theta) + \mathcal{L}_{\mathrm{CE}}(\hat{X}, y_i; \theta)). \tag{4.8}$$

Here, $\mathcal{L}_{\mathrm{CE}}$ is the cross-entropy loss of equation 2.28. With this approach, the loss function in the adversarial training consists of two equally weighted components, i.e., the regular loss and the adversarial loss. Goodfellow*et al.* also introduced a weighting factor in their formulation, but it is omitted here. Similar to Equation4.8, their losses were equally weighted as well.

4.3.2 Evaluation of Adversarial Training

The following experiment uses the TED-LIUM 2 dataset to train the ASR model and the moving window method to augment its training data. This dataset includes many shorter utterances that are covered more effectively with a small window size and overlapping segments. The moving window parameters for window length $l_w = 4$ and utterance partitions $\nu = 2$ were chosen beforehand based on effectiveness of their adversarial examples in Subsection 4.2.6 and the dataset. The sampling probability of augmenting a mini-batch p_a is set to 0.05. The noise factor ϵ is sampled uniformly from a range of $[0.0; 0.3]$. The adversarial training in this experiment is performed without SpecAugment to obtain comparable results; nevertheless, it is possible to combine adversarial training with SpecAugment. To keep the benchmarked models comparable, models shall be exposed to the same number of training samples, i.e., all models are trained for 10 epochs. All augmented models have a warm-up period of $N = 5$, i.e., their adversarial training starts after the 5th epoch.

The pre-trained LSTMP model[iv] of the ESPnet toolkit [195] provides a baseline that helps to compare results. While the LSTMP model can be used for streaming ASR, this property is not needed for this experiment. Thus, the models trained in the experiment use bidirectional encoders that are more often used than unidirectional LSTM encoders. Other than that, the parameters of the various models are the same. Thus, the three RNN models are trained in this experiment: (1) A model with a regular training regime, (2) an attention-only model with attention-based adversarial training, and (3) a hybrid CTC/attention model with hybrid CTC/attention adversarial training. Decoding with support of a language model uses a weight in decoding of $\beta = 1.0$ and a pre-trained RNNLM language model that has 2 layers with each 650 units.

Table 4.3 shows the evaluation results of those four models on four variants of the TED-LIUM 2 test set: (1) The unchanged test set, (2) adversarial examples generated

[iv]This model, *tedlium2.rnn.v1*, was already mentioned in Section 4.2.6. It is a RNN network as described in Section 2.6 that has a four-layer unidirectional encoder with projection neurons, and only one decoder layer. Each layer has 1024 neurons.

Table 4.3: Benchmark for adversarial training on the TED-LIUM 2 dataset. The first row shows results with a pre-trained model of the ESPnet toolkit as baseline. The second row contains results using a BLSTMP model. The third and the fourth row include the CER and WER results for the attention-only and for the hybrid model, respectively. The first value in each cell corresponds to the CER and the second to the WER, both given as percentage. The CTC weight during adversarial noise generation is determined by the value of ξ; the parameter λ sets the weight of the CTC model during decoding.

CER/WER	ξ	λ	LM	Dataset			
				test	test AAE	noise 30dB	noise 5dB
pre-trained model	-	0.0	-	20.7/22.8	90.7/89.1	23.6/25.8	78.8/78.8
	-	0.5	-	15.7/18.6	86.1/89.9	18.1/21.3	66.1/68.3
	-	0.5	✓	16.3/18.3	**98.5/92.2**	19.2/20.8	73.2/72.7
regular model	-	0.0	-	18.5/20.2	69.4/67.7	21.1/22.7	76.9/76.0
	-	0.5	-	14.4/16.8	58.0/60.8	16.7/19.2	63.9/66.1
	-	0.5	✓	15.4/17.0	65.9/62.7	17.8/19.2	71.2/69.6
att.-based adversarial training	0.0	0.0	-	17.7/19.6	63.6/63.3	21.0/22.8	74.7/74.4
	0.0	0.5	-	14.3/16.9	**53.5/56.8**	16.5/18.9	62.6/65.0
	0.0	0.5	✓	15.1/16.9	60.3/58.3	17.5/18.9	69.0/68.0
hybrid adversarial training	0.5	0.0	-	17.9/19.8	65.2/65.0	20.4/22.3	74.9/75.0
	0.5	0.5	-	**14.0/16.5**	**54.8/58.6**	**16.2/18.7**	**63.5/65.8**
	0.5	0.5	✓	14.8/16.6	61.8/59.9	17.0/18.5	70.0/69.2

using the moving window method, (3) the test set augmented with low white noise at 30 dB, and (4) the test set augmented with loud 5 dB white noise. In general, the generated adversarial examples are effective on all tested models; recognition performance is reduced for all models in comparison of the regular test set to the adversarial example test set.

Also, decoding with CTC reduces the error rates. Decoding with the support of a RNNLM increased error rates. Here, the language model weight was set to a relatively high value of $\beta = 1.0$ in all decoding runs. Language model rescoring would increase the recognition performance, but in this case, a constant value was set in all decoding runs to keep the results comparable to each other.

The result table includes CER as well as WER, as this gives a hint how a certain noise affected the model. This analysis has parallels to the discussion of error patterns in Chapter 3. A CER higher than the WER may indicate word loops or dropped sentence parts, and a higher WER may indicate that the words itself are more often misspelled. Section 3.4 discusses possible causes for generated patterns that are generated when the attention decoder looses its alignment. However, in most adversarial examples, CER and WER remain at similar levels, and word loops or dropped sentence parts appear rarely.

Decoding the adversarial examples with a language model caused a slightly higher averaged rate of CER to WER than regular noise for all models. The baseline model performed worse on the adversarial example test set with an error rate of over 90%. Its transcriptions showed on average 90% substitution errors and only 20% insertion or deletion errors.

It is partly a consequence of using a bidirectional encoder that the regularly trained model performs better on the regular test set than the baseline model. However, the results show a general trend that adversarial training improved the robustness on regular noise and specially on adversarial noise. Both adversarially trained models performed better than the non-adversarially trained models, especially on the adversarial examples test set. Hybrid adversarial training improved not only the robustness against adversarial examples, but also improved performance on the regular test set by an absolute of 2.1% to 16.5% WER in comparison to the baseline model. It also achieved an improvement of 24% up to 33% absolute WER compared to the baseline on adversarial examples. Decoding in combination with CTC and LM achieves a WER of 58.6%, but with the regularly trained model only a WER of 92.2%.

This evaluation compared various types of adversarial noise. Among those types, attention-only noise achieved the highest error rates. Yet this did not translate to better performance in adversarial training: Attention-only as well as hybrid adversarial training achieved a similar performance in this experiment.

4.4 MP3 Compression as Countermeasure

This section investigates the MP3 audio compression codec as a method to diminish adversarial noise. This technique can be applied to a pretrained model without any need for modifications to the existing model architecture. The MP3 algorithm encodes audio streams using lossy audio compression that discards inaudible audio components, e.g, that are below the psychoacoustic hearing threshold [23, 24, 218]. MP3 compression itself introduces additional artifacts, notably low-pass filtering and spectral valleys. While MP3 compression in addition with noise can be used for data augmentation [132], high compression ratios degrade ASR performance [18, 19, 145, 146]. This effect is primarily observed with compression ratios greater than 32 kbps [171]. Schönherr *et al.* [164] were the first to hypothesize MP3 as a countermeasure against adversarial examples. Zhang *et al.* [215] employed MP3 compression to combat adversarial noise stemming from CTC loss. The following sections demonstrate an experimental proof of MP3 compression effectiveness against adversarial noise, specifically with the hybrid end-to-end ASR network.

This hypothesis is evaluated using four models trained on various levels of MP3 compressed training data. Their performance on regular and adversarial examples are listed in Section 4.4.2. Section 4.4.4 discusses the effect of various MP3 compression ratios on robustness against adversarial examples. The results indicate that MP3 compression partially reduces the error rates on adversarial examples. This desirable effect of MP3 compression does not apply to samples augmented with regular noise, as indicated

by the experiments in Section 4.4.5. The investigation concludes with the proof that MP3 compression reduces adversarial noise by using an estimation of the SNR value of adversarial audio examples before and after MP3 compression.

4.4.1 Experimental Evaluation

For experimental evaluation, four ASR models trained on uncompressed audio data, as well as 24, 64 and 128 kBit MP3 compressed data. The compression was executed using the Lame MP3 encoder [35]. Previous experiments use the TED-LIUM 2 corpus, which is already compressed, as the audio data is obtained from the TED website that provides only mp4 video files with compressed audio [159]. To eliminate effects of an already applied MP3 compression on the experiment, the experiments are conducted on an uncompressed corpus. Therefore, the ASR model in the following experiments on MP3 compression are trained on the freely available VoxForge dataset that contains 130.1 hours of uncompressed audio [190], of which 105 hours were used as training set. These four models share the same parameter set and are based on the previous RNN model configuration mentioned in Section 4.2.7.

To generate adversarial audio, the moving window adversarial noise with gradient normalization is applied, as described in Equation 4.4 of Section 4.2.3. This gradient is then weighted by the factor ϵ, which is set to 0.3 in the following experiments, and then added to the original features. To prevent label leaking, a previously decoded label sequence is utilized as a reference [110]. Accumulated gradients from a sliding window with a fixed length l_w and stride ν contribute to the adversarial noise $\nabla_{\text{Adv}}(x_t)$, with gradient normalization ensuring the appropriate accumulation of the gradient directions [53].

$$\nabla_{\text{Adv}}(x_t) = \nabla_{\text{MW}}(x_t) \tag{4.9}$$

$$\delta_{AAE}(x_t) = \epsilon \cdot \text{sgn}(\nabla_{\text{Adv}}(x_t)) \tag{4.10}$$

$$\hat{x}_t = x_t + \delta_{AAE}(x_t), \quad \forall t \in [1, T] \tag{4.11}$$

$$y \neq f(\hat{x}_t, \theta) \tag{4.12}$$

No language model was used for decoding, only the hybrid CTC/attention ASR model.

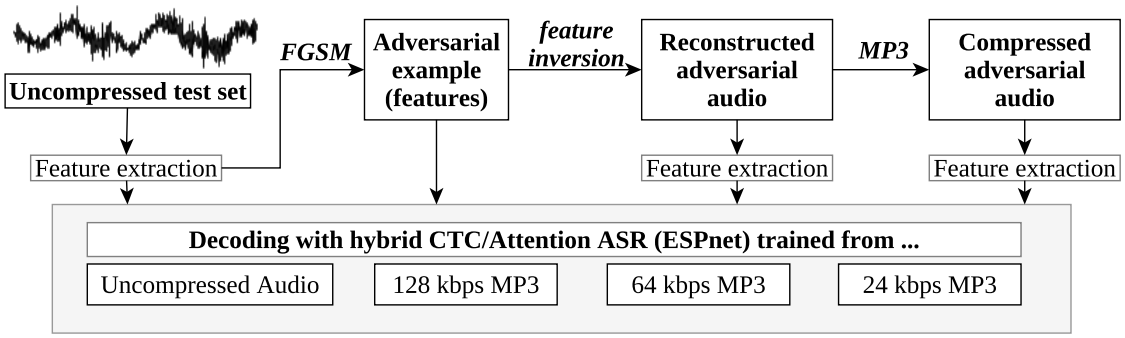


Figure 4.3: General experimental workflow for MP3-compressed audio adversarial examples.

4.4.2 Performance on Regular and Adversarial Examples

Table 4.4 lists the decoding results of the experiment workflow depicted in Figure 4.3. Decoding only uses the ASR model, without any language model; in this case, CER is more suitable as a performance metric. All models achieve on the regular test set an average CER of 18.8. The model trained with uncompressed data performed slightly better, whereas the model trained on 24kbps performed slightly worse.

Table 4.4: Results of the MP3 compression experiment on regular and adversarial test sets. The table contains the performance of the four ASR models trained on various levels of MP3 compression. The results correspond to four types of test datasets from the original test dataset to its MP3-compressed adversarial audio as shown in Figure 4.3. The last row gives the relative CER improvements from adversarial features to MP3-compressed adversarial audio.

CER	**MP3 compression on the training dataset**			
	Uncompressed	128 kbps	64 kbps	24 kbps
Type of test dataset				
Original test dataset	17.8	18.8	18.6	20.2
Adversarial features	70.5	72.3	71.8	69.0
Reconstructed adversarial audio	62.2	64.0	63.1	60.5
24 kbps MP3 adversarial audio	57.4	58.4	56.5	55.3
Relative CER change	−18.58%	−19.23%	**−21.31%**	−19.86%

The test set augmentation with adversarial noise led to additional 52% of relative CER on average for all models. All models exhibit a CER of around 71 on adversarial features. The significant increase of CER demonstrates that the adversarial examples are effective on this dataset.

Reconstruction of adversarial features to adversarial audio led to a slight decrease of decoding errors. All models exhibit an average CER of 62 on the reconstructed adversarial audio set. This slight improvement of error rates in comparison to adversarial features may be attributed to the data processing. Starting from adversarial features, the corresponding audio is reconstructed and given to feature extraction for decoding with the ASR model. This process is lossy and seems to impact the adversarial noise as well.

The final dataset consists of 24kbps MP3-compressed reconstructed adversarial audio, that reduces CER to an average of 56.9. The last row compares these results to the adversarial features test set, as the error rates are reduced by around 20%. While the MP3 compression improves the CER, error rates are still significantly higher than on the original test set. The reduced error rates indicate that MP3 compression is partially effective as a countermeasure against adversarial noise.

For all models, MP3 compression improved the performance on the test set in terms of relative CER significantly. While the model trained on 24kbps exhibited the best performance on adversarial examples, but deteriorated performance on regular test data. The model trained on 64kbps MP3 data performs relatively well on the original data

and shows the best relative CER improvement on the MP3-compressed adversarial example. Thus, this model provides a trade-off between regular ASR performance and countermeasure against adversarial noise.

4.4.3 Visualization of Adversarial Noise and MP3 compression

MP3 compression induces changes in the spectral composition diagram that are visible in the spectrogram. This section investigates the effect of adversarial noise by inspection of the spectrum of MP3 compressed adversarial examples.

Figure 4.4 shows the spectrograms of the original audio, the reconstructed audio and the uncompressed adversarial example. In the reconstructed audio, a small band of the lower frequencies is missing that is lost after reconstruction. Mel filters start at roughly 40Hz, and any spectral information in lower frequencies than that is not included as extracted features. When comparing the reconstructed audio and the adversarial example, the adversarial noise introduced slight changes in the spectral composition of the audio that are distributed across the spectrogram. The spectrogram of the adversarial example exhibits a slightly elevated level of background noise and slightly blurred spectral features, e.g., the original audio spectrum exhibits vowel harmonics that are reduced in the spectrum of the adversarial audio.

Figure 4.5 displays the spectrograms of 128, 64 and 24 kbps MP3 compressed audio adversarial examples. These are the MP3 compressed counterparts to the adversarial example in Figure 4.4. MP3 compression with 128 and 64 kpbs had a relatively low impact on the spectral composition, and are barely distinguishable by hearing and by visual inspection of the spectrum. On the third spectrogram, 24 kpbs compression nearly cropped the spectrum at 6 kHz and the spectral distribution is not nearly as fine-grained as before.

4.4.4 Quantifying the Effect of MP3 compression using SNR

MP3 compression reduces error rates on adversarial examples, but does MP3 also directly reduce the adversarial noise? This section investigates this question by inspection of the spectrum of MP3 compressed adversarial examples. Signal-to-noise ratio (SNR) helps to quantify the adversarial noise and its reduction by MP3 compression. The SNR before and after MP3 compression is estimated and evaluated regarding statistical significance.

SNR is generally calculated as

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \frac{\text{signal power}}{\text{noise power}} \tag{4.13}$$

In our case, adversarial noise is generated and added in the feature domain. Because it is not trivial to determine signal power from the feature domain, the signal power is derived from the reconstructed time domain:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \frac{\text{Reconstructed audio power}}{\text{Adversarial audio power - Reconstructed audio power}}, \tag{4.14}$$

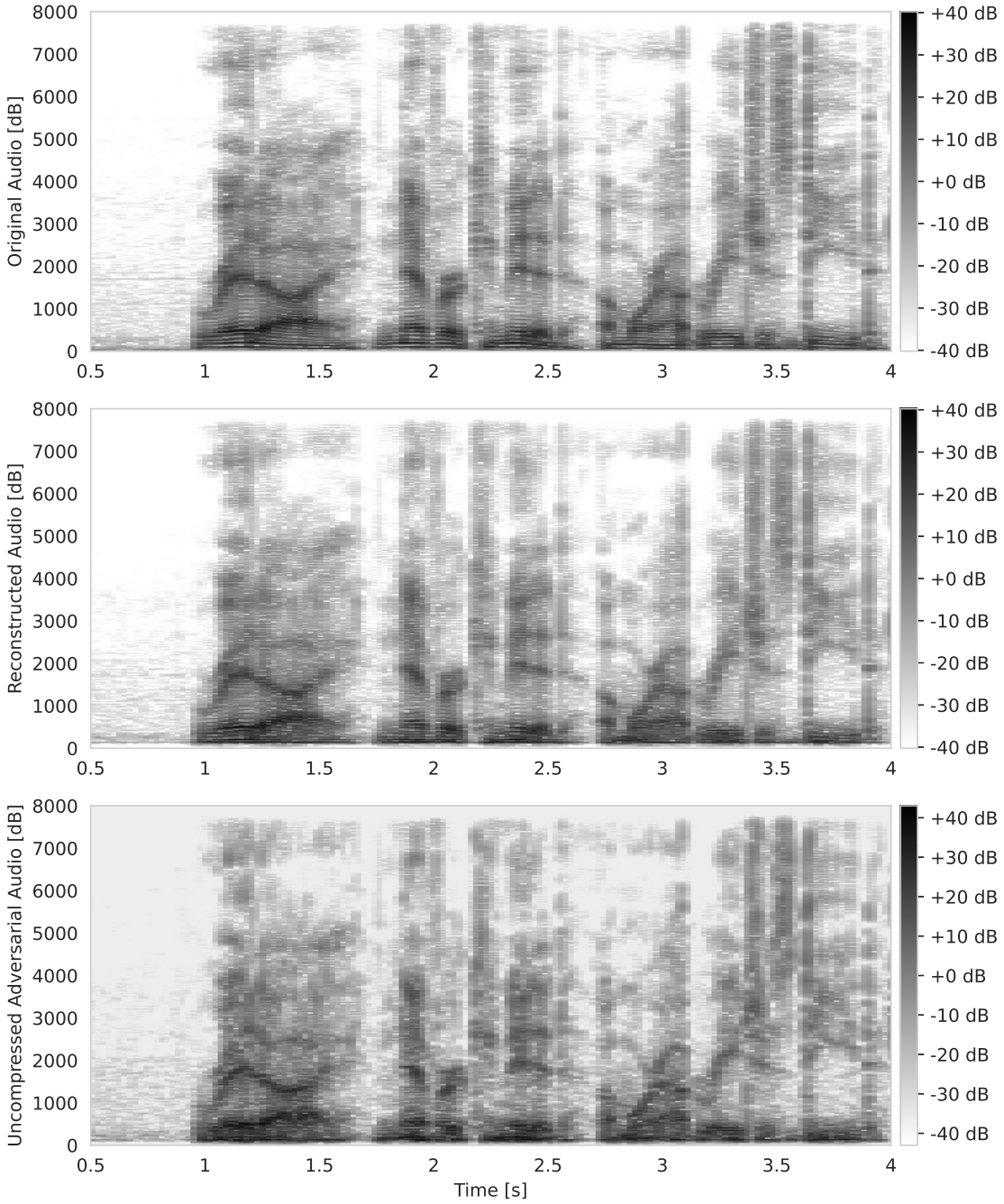


Figure 4.4: Spectrograms for original and reconstructed audio, and the reconstructed adversarial example. The original audio displays clearly visible horizontal lines in the lower frequency bands, indicating harmonic frequencies. These lines, however, become somewhat obscured in the reconstructed audio and are significantly blurred in the reconstructed adversarial example.

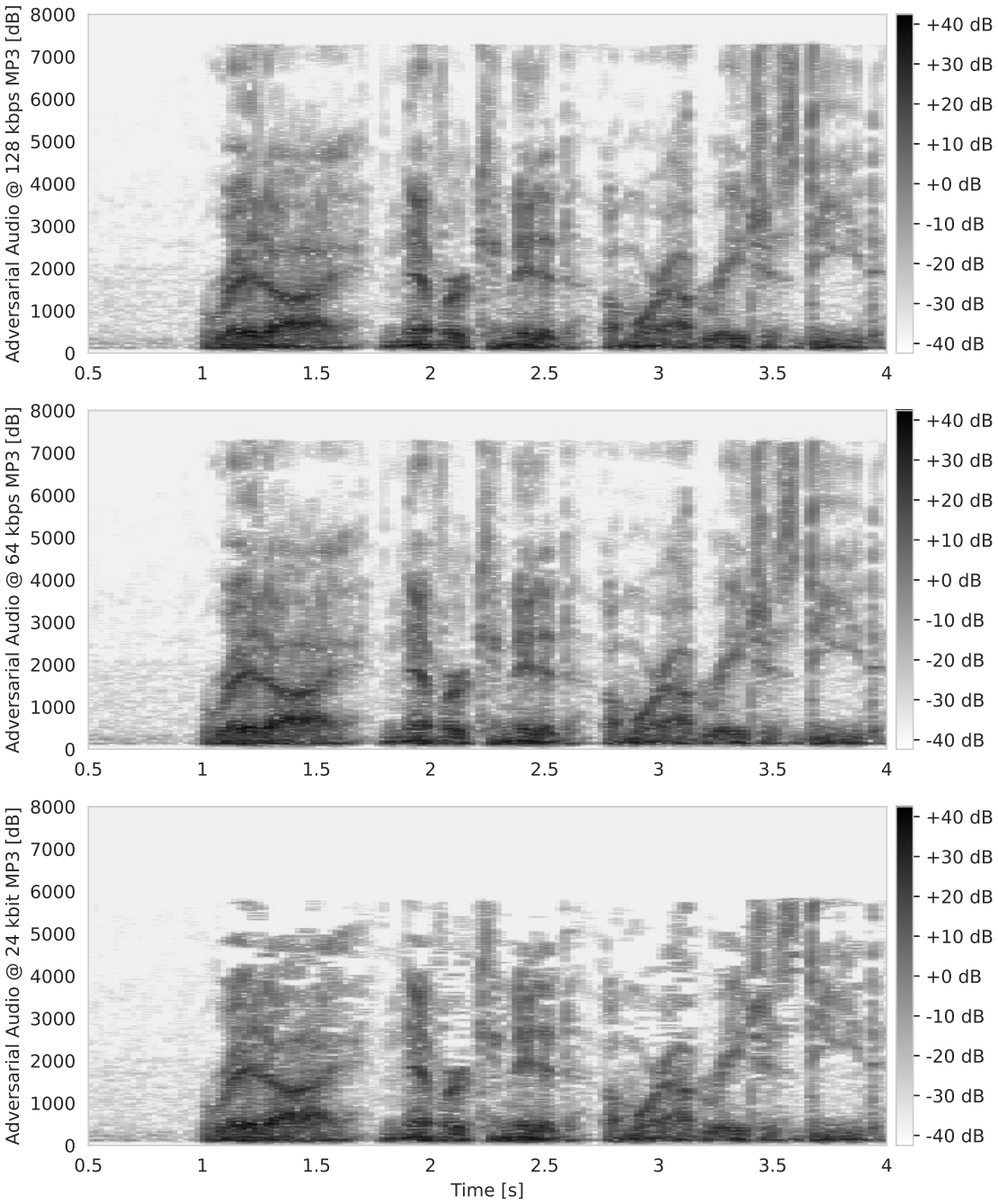


Figure 4.5: Spectrograms of 128, 64 and 24 kbps MP3 compressed audio adversarial examples. MP3 compression reduces spectral entropy in higher frequencies above 7.5kHz, and even above 6 kHz for 24 kbps compression. Compression still preserves most of the spectral features when compared to uncompressed adversarial audio in Figure 4.4.

whereas each signal power is determined from the corresponding audio signal in time domain $x_a(t)$ as

$$\mathrm{P}_a = \frac{1}{T}\sum_{t=1}^{T} x_a(t)^2. \tag{4.15}$$

The SNR is then computed for compressed as well as for uncompressed adversarial audio. Audio reconstruction and MP3 compression changes the audio file, e.g., by audio normalization or by padding or dropped values due to fixed block lengths of intermediate processing steps. To ensure an accurate SNR estimation, the SNR is calculated from the same-type audio signals that are either reconstructed or MP3-compressed.

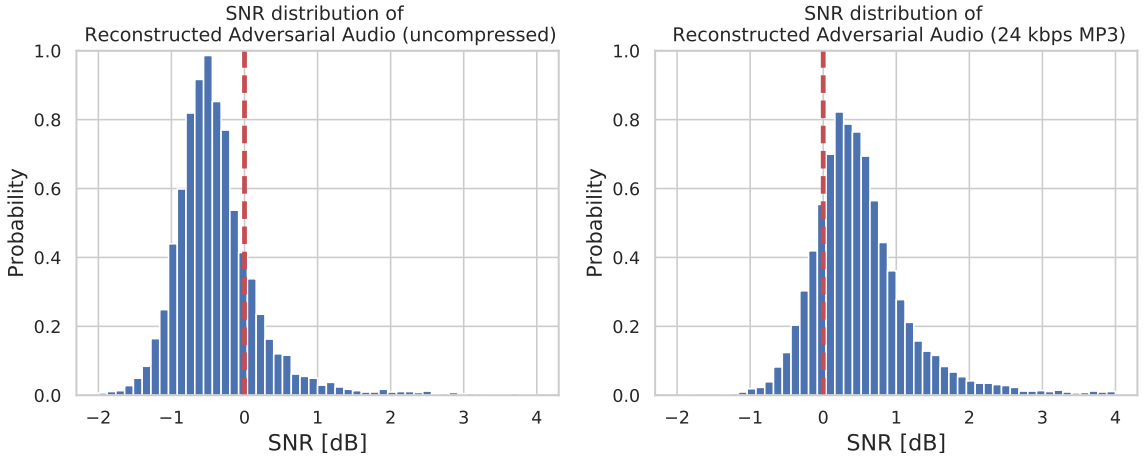


Figure 4.6: SNR estimation for uncompressed and 24 kbps MP3 compressed adversarial audio [1+]. The histograms are normalized and the number of histogram bins was set to 50 for both plots. The measured SNRs of uncompressed adversarial audio predominantly show negative values, suggesting audible adversarial noise. After MP3 compression, there is a notable shift towards positive SNR values, indicating a reduction in adversarial noise and an improvement in SNR.

Adversarial noise strongly depends on a given audio utterance, and thus, the SNR value is obtained for each utterance rather than to calculate an average SNR value for the dataset. Measured SNRs of utterances follow a certain distribution that can be visualized in a histogram. Figure 4.6 shows the SNR distributions of uncompressed and MP3-compressed adversarial audio datasets. Measured SNRs of uncompressed adversarial audio are between −5dB and +25dB, while MP3-compressed adversarial audio ranges between −5dB and +35dB. Notably, the majority of SNR values of uncompressed adversarial audio is negative, i.e., the adversarial noise outweighs the original audio signal and is audible. After MP3 compression, the centre of the SNR distribution is shifted towards positive values. The trend to positive SNR values indicates that the SNR is improved and adversarial noise is diminished after MP3 compression.

With these SNR distributions, the main thesis of this section can be validated: Does MP3 compression reduce adversarial noise? The SNR distributions are evaluated with the Kolmogorov–Smirnov test [123] to assess whether they originate from the same

underlying probability distribution. This statistical test constructs two hypotheses: (1) The null hypothesis states that MP3 compression did not improve the SNR and thus, the SNR distributions before and after the compression are identical. (2) The alternative hypothesis is that these two distributions are not identical. These two hypotheses are evaluated with the Kolmogorov–Smirnov test statistic. This is the two-sided form of the Kolmogorov–Smirnov test which yields a p-value that indicates the probability of the null hypothesis under the given data [83].

To pass this test, any p–value that is smaller than 0.05 validates the alternative hypothesis with sufficient statistical significance. The Kolmogorov–Smirnov test of the SNR distributions between uncompressed and MP3-compressed adversarial audio returned a p–value of 0.039. Therefore, the test finds that MP3 compression increases the SNR values of adversarial samples and reduces adversarial noise.

4.4.5 Comparison to Regular Noise

Results in Table 4.4 demonstrated that MP3 compression improves ASR performance on samples with adversarial noise. This section investigates whether MP3 compression also reduces the impact of other noise types, namely white, pink, brown and babble noise.

These noise variants have distinct spectral distributions: White noise is sampled from the uniform distribution and resembles a hissing sound; its spectral function is constant over the audible frequency range from 20 Hz up to 20 kHz. Pink noise is characterized by a reduced spectral power density at higher frequencies which resembles common ambient noises such as leaves in the wind or a waterfall. Brown noise has a greater intensity on lower frequencies in comparison to pink noise, and rather sounds like heavy rainfall. In mathematical terms, the spectral power density of pink noise is inversely proportional with frequency f, i.e., $S_p(f) \propto 1/f$, while the density for brown noise is inversely proportional to f so that $S_b(f) \propto 1/f^2$. And lastly, babble noise is composed of overlapping voices, representing the generalized spectrum of human speech. White, pink, and brown noises were produced with the SoX audio utility [12]. Meanwhile, babble noise was created from a benchmark recording.

Table 4.5 shows the experimental evaluation of the effects of noisy data and their MP3-compressed correspondent. This ASR model achieved a CER of 17.8 on the original uncompressed data. The VoxForge test set was augmented with noise at different intensities that was then decoded with the ASR model trained on uncompressed data from Section 4.4.2. In a second step, each augmented test set was then MP3 compressed and decoded. The table lists results for six different SNRs, from barely audible noise at 30 dB, over to noticeable distortion at 5 dB SNR, and up to the SNR of -10 dB in which the noise predominates the speech signal. Overall, increasing noise lowers the SNR and reduces ASR performance. White noise and babble noise are more evenly distributed on the spectrum and seem to be the most detrimental to the speech recognition system. Pink noise and brown noise exhibit higher spectral density on lower frequencies and have less interfering impact on the investigated ASR decoder.

Error rates turn out higher for the MP3-compressed test sets when compared with the uncompressed set, regardless of noise type or SNR. To put this into contrast, MP3

Table 4.5: CER values for test sets augmented with regular noise distributions without and with 24 kbps MP3 compression. All results were obtained using the ASR model trained from uncompressed data. In all cases, MP3 compression increased the CER.

CER	**SNR [dB]**					
	30	**10**	**5**	**0**	**-5**	**-10**
White noise	19.1	32.7	41.9	53.7	66.2	78.2
+ MP3 compression	29.1	51.2	61.7	71.2	78.7	86.0
Relative CER change	+52.4%	+56.6%	+47.3%	+32.6%	+18.9%	+10.0%
Pink noise	18.5	29.1	38.1	51.7	67.4	82.1
+ MP3 compression	26.9	42.5	53.0	66.4	79.8	89.9
Relative CER change	+45.4%	+46.0%	+39.1%	+28.4%	+18.4%	+9.5%
Brown noise	17.9	19.7	21.9	26.1	34.1	47.8
+ MP3 compression	25.3	29.0	32.5	38.0	47.3	60.6
Relative CER change	+41.3%	+47.2%	+48.4%	+45.6%	+38.7%	+26.8%
Babble noise	18.3	35.8	53.4	77.4	89.0	93.6
+ MP3 compression	25.8	48.2	66.0	83.6	93.1	95.4
Relative CER change	+41.0%	+34.6%	+23.6%	+8.0%	+4.6%	+1.9%

compression on adversarial examples reduces error rates, as listed in Table 4.4. These results provide further evidence that MP3 compression reduces adversarial noise to a certain proportion, whereas the investigated other types of noise are not diminished by compression.

5

Integrated Feature Extraction with Sinc Convolutions

Hybrid CTC/attention speech recognition relies on F-bank features, i.e., pre-processed frequency-domain features. Very similar to Mel Frequency Cepstral Coefficients (MFCCs), the F-bank features are not parametrized learnable features but rather designed to emulate the human hearing.

The work presented in this chapter is motivated by advances in speech and speaker recognition on learnable feature extraction, which can be integrated into end-to-end trainable neural architectures as a network layer. In particular, Sinc convolutions mirror the behaviour of band pass filters in time domain and are an effective approach to classify directly from raw audio. Two exemplary applications investigate the properties of integrated Sinc convolutions:

The first exemplary application is a keyword spotting task based on the Google speech commands dataset. Architectures for this task are usually small and employed in battery-powered microcontrollers. Our example architecture demonstrates how networks with Sinc convolutions can have only 64k parameters, but still outperform larger networks that use conventional convolutions.

The second exemplary application is integrated feature extraction in an RNN-based hybrid CTC/attention speech recognition network. For evaluation, a Sinc convolutional model was trained and tested on the TED-LIUM 2 dataset. The resulting Sinc convolutional ASR model was much smaller but still slightly outperformed a comparable baseline model.

The work in this chapter is driven by recent advancements in speech and speaker recognition, which utilize learnable feature extraction techniques that can be integrated into an end-to-end trainable neural network [58, 114, 140, 155–157, 199, 20†]. This chapter summarizes the work published by Mittermaier *et al.* [20†, 20^{+}] and by Kürzinger *et al.* [6†].

The first section of this chapter contains a general introduction to previous work regarding Sinc convolutions, keyword spotting and fully end-to-end speech recognition. The second section introduces the classification architecture using Sinc convolutions for

the keyword spotting task. The third section demonstrates a speech recognition system with integrated Sinc convolutions as learnable feature extraction input layer.

5.1 Related Work

This section offers an organized overview of existing literature, categorized into three essential topics: 1. The original SincNet architecture that utilizes Sinc convolutions for hybrid DNN/HMM speech recognition; 2. prior work to keyword spotting on raw audio; 3. and end-to-end speech recognition on raw audio data.

5.1.1 SincNet and Derivatives

For traditional hybrid DNN/HMM speech recognition, the relationship between the acoustic speech signal and HMM states, representing phonemes, is modeled by first extracting acoustic features from the speech signal and then training a classifier to estimate the emission probabilities of the HMM states. Several prior publications have focused on using raw audio input for acoustic modeling [64, 135–137, 184]. Tüske *et al.* directly utilized the raw audio as input to the DNN [184]. Both Golik *et al.* [64] and Palaz *et al.* [137] proposed approaches using CNNs as a filter to transform raw speech signals for input into the DNN, aiming to estimate the HMM state class conditional probabilities at the output. As noted by Palaz *et al.* [137], this method allows for the simultaneous learning of relevant features and the classifier from the raw speech signal. This eliminates the need for separate feature extraction and statistical modeling steps. Moreover, their approach demonstrated that relevant feature representations are more effectively learned by processing the input raw speech at the sub-segmental level, consequently yielding improved ASR performance.

A significant insight in the domain of feature extraction is that parameterizable convolutions with a filter structure exhibit superior convergence behavior on raw audio data compared to CNNs alone [156]. Based on this insight, Ravanelli *et al.* proposed the SincNet architecture [156], which uses Sinc convolutions as its input layer, followed by CNN layers. This architecture was first applied to speaker recognition [156, 157] and subsequently to phoneme recognition [155].

The advantages of combining Sinc convolutions with regular CNN layers have been explored in the SincNet architecture. However, alternative convolutional filters can offer benefits such as faster convergence, improved data structure compatibility, and reduced network parameters. Section 5.3 descrives an approach that involves the application of depthwise separable convolutions [41, 95] on top of Sinc convolutions. This approach was first employed by Mittermaier *et al.* [20†], where Sinc convolutions are effectively used for low-parameter keyword spotting to optimize power consumption in battery-driven devices. The benefits of depthwise separable convolutions are particularly evident in low-parameter settings, as shown in previous studies such as HelloEdge [88, 217]. Another technique to decrease the number of parameters in convolutions is lightweight convolutions [200], which are based on depthwise convolutions.

5.1.2 Keyword Spotting

The application of an architecture, which is a derivative of the SincNet architecture, to the Keyword Spotting (KWS) task, specifically wakeword recognition, is presented in Section 5.3. KWS models are trained to detect the presence of specific spoken words within unconstrained speech [59]. Such models are an integral component of voice-activated systems and virtual assistants, allowing them to "wake up" or start processing user commands upon detecting the wake word. This section examines prior architectures relevant to keyword spotting.

In recent years, DNNs [33] and CNNs have been utilized effectively for KWS [40, 115, 181, 216, 217]. Zhang *et al.* evaluated various neural network architectures, including CNNs, LSTMs, and GRUs, in terms of accuracy, computational operations, memory footprint, and deployment on embedded hardware [217]. They achieved their best results using a CNN with depthwise separable convolutions (DSConvs). Further analysis of the memory requirements of the model in [217] is conducted in [167]. Choi *et al.* also built upon this work by utilizing a ResNet-inspired architecture, but instead of using 2D convolution over a time-frequency representation of the data, they convolve along the time dimension and treat the frequency dimension as channels [40].

5.1.3 Fully End-to-End Speech Recognition

Despite being referred to as end-to-end models, i.e., models trained in an end-to-end manner, end-to-end speech recognition systems still require the use of pre-processed acoustic features, such as Mel-filter-banks, thus precluding a purely end-to-end pipeline based on the raw audio signal. Notable prior work to integrating feature extraction into end-to-end models has been made by Latif *et al.* [114] and Won *et al.* [199].

Section 5.4 presents a fully end-to-end speech recognition system that incorporates the Sinc convolution as its first input layer. The kernels of the first convolutional layer are limited to learning only the shapes of parametrized Sinc functions. However, this is not the first architecture of this structure to be proposed. The SincNet architecture has also been applied to end-to-end speech recognition. Parcollet *et al.* introduced the E2E-SincNet [140], which integrates SincNet as building block into the hybrid CTC/Attention architecture. Their aim was to move towards a fully end-to-end speech recognition system.

However, there were previous architectures that employed fully end-to-end speech recognition from the raw waveform. These approaches also started to outperform conventional hybrid DNN/HMM speech recognition [6, 76, 108, 211]. One approach to classify directly from the raw waveform, using Gammatone filterbanks [199]. Another approach to classify directly from the raw waveform is to use the scattering transform [211]. The last two approaches inspired the use of CNNs that are guided by mel filter coefficients. Another common approach is also to classify using similar methods as in image recognition. For example, Hannun *et al.* train an end-to-end recurrent network on spectrogram features [76]. But only few architectures directly classify on raw audio such as [108, 212].

5.2 Architecture Elements

This section explores the Sinc convolution, along with other architectural elements used in keyword spotting and speech recognition architectures. These elements comprise the Hamming and Hann windowing functions, the use of the log compression function as an activation layer, and depthwise separable convolutions.

5.2.1 Sinc Convolutions

A bandpass filter in spectral domain can be constructed as a learnable convolutional filter, applied to the signal in time domain [156]. The rectangular function in the spectral domain corresponds to a Sinc function in time domain.

$$\mathrm{sinc}(x) = \frac{\sin(x)}{x} \tag{5.1}$$

Therefore, a band pass filter with cutoff frequencies f_1 and f_2 corresponds to the Fourier transform of two Sinc functions.

$$G[f, f_1, f_2] = \mathrm{rect}(\frac{f}{2f_2}) - \mathrm{rect}(\frac{f}{2f_1}) \tag{5.2}$$

Application of the band pass filter is a convolution in time domain. A Sinc convolutional kernel with L coefficients is derived by sampling the continuous filter function

$$g[n, f_1, f_2] = 2f_2\,\mathrm{sinc}(2\pi f_2 n) - 2f_1\,\mathrm{sinc}(2\pi f_1 n). \tag{5.3}$$

The cutoff frequencies for the filter shall fulfill $f_1, f_2 \geq 0$ and $f_2 \geq f_1$. To ensure that the filter frequencies are bound to this condition, the Sinc convolutional layer integrates two learnable parameters w_1 and w_2 that directly relate to the cutoff frequencies:

$$f_1 = |w_1| \tag{5.4}$$

$$f_2 = |w_1| + |w_2 - w_1|. \tag{5.5}$$

During training, only the two parameters are tuned, the low and high cutoff frequency of each band-pass filter. The convolutional kernel is symmetrical about the y-axis and has an odd number of coefficients. It's essential for the kernel size to be odd to incorporate the coefficient at $y = 0$. Throughout this study, the chosen kernel length was $L = 101$.

5.2.2 The Hamming Windowing Function

The windowing function is an additional step in the feature extraction to mitigate spectral distortions that are caused by discontinuities at the edges of the windowed signal segment. For extraction of conventional F-bank features, the windowing function is applied onto the windowed audio data before frequency transformation.

Sinc convolutions fully operate on the signal in the time domain. Here, the Window function softens the convolutional kernel to approximate it towards an ideal filter. It is

advantageous to convolve the windowing function directly on the convolutional kernel, as both the windowing function and the convolutional filter can be precomputed. The Hamming window is multiplied with the L weights in the Sinc-convolution kernel:

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{L}\right) \tag{5.6}$$

$$g'[n, f_1, f_2] = g[n, f_1, f_2] \cdot w[n] \tag{5.7}$$

The architecture discussed in this chapter use the Hamming windowing function, whereas the standard feature extraction of hybrid CTC/attention models is the Hann windowing function [134], described by

$$w_{\text{Hann}}[n] = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{L-1}\right)\right]. \tag{5.8}$$

5.2.3 Log Compression as Activation Function

The Sinc convolution is combined with an activation function that serves as a non-linearity. The original publication on Sinc convolutions uses the ReLU activation function, which may drop some information. This work uses the log compression:

$$\text{LogCompression}(x) = \log(|x| + 1) \tag{5.9}$$

Log compression is an already established preprocessing step in regular feature extraction. Mittermaier *et al.* demonstrated that log compression can improve the classification performance on raw audio when compared to regular ReLU activation function [211, 212, 20$^+$, 20†]

5.2.4 Depthwise Separable Convolutions

Depthwise separable convolutions are modifications of standard convolutions, factoring the convolution operation into two components: depthwise and pointwise convolutions. This factorization decreases the quantity of parameters and computations, thus making the network more lightweight without major performance loss.

Kaiser *et al.* [95] provide an in-depth and mathematical overview of depthwise separable convolutions; this is a brief summary: Regular 1D convolutional layers perform convolution along the time axis, simultaneously merging the input channels. However, in Depthwise Separable Convolution (DSConv), each input channel is first convolved with a separate 1D filter (depthwise convolution). Subsequently, a 1×1 or pointwise convolution is used to merge the distinct channels. This decomposed approach significantly reduces the parameter and computation requirements of the convolution operation. The parameter count for regular convolution, DSConv, and Grouped Depthwise Separable Convolution (GDSConv) is expressed as follows [95, 20†]:

$$N_{\text{Conv}} = k c_{in} c_{out} \tag{5.10}$$

$$N_{\text{DSConv}} = k c_{in} + c_{in} c_{out} \tag{5.11}$$

$$N_{\text{GDSConv}} = k c_{in} + \frac{c_{in} c_{out}}{g} \tag{5.12}$$

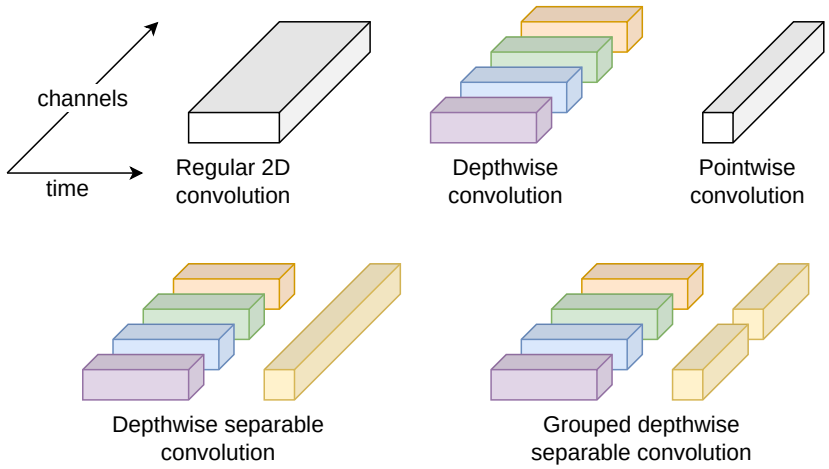


Figure 5.1: Progression of kernels from a standard convolution to a grouped depthwise separable convolution; parameters are shared across elements of the same color. In the context of 1D convolutions, standard convolutions operate along the time axis, spanning across all channels simultaneously. On the other hand, depthwise convolutions process each input channel independently.
Pointwise convolutions are a specific form of convolution where a 1D filter is applied across all channels. Combining depthwise and pointwise convolutions results in a depthwise-separable convolution, where channels are convolved separately along the time axis (depthwise) and then information across channels is integrated using a 1x1 convolution (pointwise).
Building upon this, grouped depthwise-separable convolutions are derived by segmenting channels into g distinct groups, with a DSConv being independently applied to each of these groups. Thus, grouped depthwise-separable convolutions essentially represent a grouped execution of depthwise-separable convolutions operations across partitioned channel sets.

In situations where the kernel length k is less than the number of output channels c_{out}, the majority of parameters are situated within the term for the pointwise convolution $c_{in}c_{out}$. This relates to approximately 95% of parameters in the architecture described in Section 5.4.

Two basic strategies can be used to further reduce the number of parameters within the DSConv: (1) utilizing weight sharing in the depthwise convolution, as lightweight convolution [200] or sub-separable convolution [95], and (2) grouping channels in the pointwise convolution, as originally proposed as super-separable convolution in [95]. Kaiser *et al.* [95] proposed the use of grouped convolution with DSConv, which reduces the parameters for the pointwise convolution by a factor g_i, indicating the number of channel groups in each layer. To facilitate information exchange between groups, different numbers of groups per layer, specifically 2 and 3 as in the original paper, are utilized for the KWS architecture in Section 5.3. This low-parameter model uses GDSConv to reduce the parameter count more efficiently than simply minimizing model depth.

5.3 Keyword Spotting with Sinc Convolutions

The task of Keyword Spotting (KWS) is to enable speech-based user interaction on smart devices. Applications for these devices often require real-time capability and high accuracy, while also being subject to constraints on hardware resources and power consumption.

Previous KWS architectures have extracted acoustic features and applied a neural network to classify keyword probabilities, with a focus on optimizing memory footprint and execution time. These architectures were designed to utilize MFCC features. The work presented in this section seeks to improve upon previous work by taking additional steps to reduce power and memory consumption while maintaining classification accuracy.

This is achieved by eliminating power-intensive audio preprocessing and data transfer steps through the use of an end-to-end architecture that extracts spectral features using parametrized Sinc-convolutions and reduces memory footprint through the use of grouped depthwise separable convolutions. The resulting network achieved an accuracy of 96.4% on the Google Speech Commands test set with only 62k parameters, demonstrating an improvement of 0.3% compared to a similarly sized ResNet model that relies on feature pre-processing.

5.3.1 Architecture for Keyword Spotting

The architecture of the proposed model is illustrated in Fig. 5.2, starting with the SincConv layer responsible for extracting features from the raw audio input. Standard activation functions are replaced with log-compression as per Equation 5.9, a successful technique in other CNN models handling raw audio data [211, 212].

Following feature extraction, the architecture comprises five layers of (G)DSConv to further analyze the obtained features. The first of these layers utilizes a larger kernel size to expand the number of channels to 160. The remaining four layers each maintain 160 input and output channels. Every (G)DSConv block includes the (G)DSConv layer, batch normalization [90], spatial dropout [182] for model regularization, and average pooling to reduce temporal resolution.

Post (G)DSConv blocks, global average pooling is applied, generating a 160-element vector. This vector undergoes processing through a Softmax layer, classifying into one of 12 categories, including *unknown* and *silence* classes.

The base model, as depicted in Fig. 5.2, consists of 122k parameters. The optimal hyperparameters were selected through comprehensive experiments, aiming to achieve the highest validation accuracy with a number of parameters similar to those in models presented in [40, 217]. The low-parameter version of this model, achieved by applying alternating groups of 2 and 3 to the DSConv layers, significantly reduces the total parameters to 62k. This reduction ensures a comparable footprint to the small-medium models by Zhang *et al.* [217].

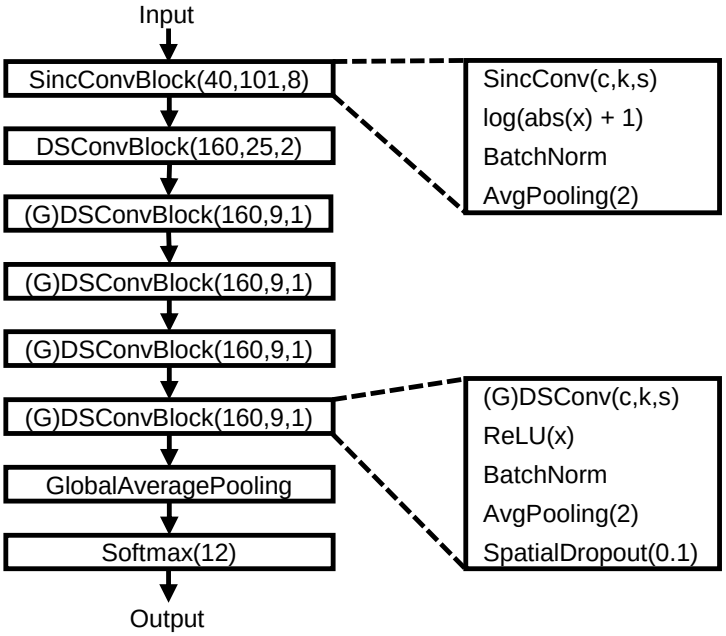


Figure 5.2: The keyword spotting architecture, elaborated upon in Section 5.3 and proposed by Mittermaier [20†], employs convolutions parameterized as c, k, s, denoting the number of output channels, kernel length, and stride, respectively. The parameter configurations of convolutions are indicated in the convolution blocks. In the low-parameter model comprised of 62k parameters, convolutional layers from the third to the sixth are grouped.

5.3.2 Experimental Setup

The investigated model is trained and evaluated utilizing the Google's Speech Commands dataset [193], a popular dataset for assessing KWS systems. The first version of this dataset comprises 65k one-second audio utterances of 30 distinct keywords spoken by 1881 unique speakers. It's important to note that this experiment focuses on the first version of the dataset, which differs from the second version that consists of 2618 unique speakers. While the second version of the dataset has 105k samples and five extra keywords [193], previous KWS research only reported results on the first version.

The standard setup classifies audio into 12 categories: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", *unknown*, or *silence*. The additional 20 keywords are categorized as *unknown*, while pre-provided background noise files are labeled as *silence*. To maintain reproducibility of the benchmarks, a distinct test set is provided, complete with a pre-established list of samples for the *unknown* and *silence* categories.

In the training process, every sample from the training set is used, resulting in a class imbalance due to a higher number of *unknown* samples. To counteract this, class weights are employed during the training phase, assigning lesser weight to *unknown* samples, thereby ensuring their influence on the model aligns with other classes. Consequently, the model can experience more *unknown* word samples during training without developing a bias.

The model is trained for 60 epochs, using the Adam optimizer [46] with an initial learning rate of 0.001, and a learning rate decay of 0.5 applied after 10 epochs. The model demonstrating the highest validation accuracy is saved for subsequent evaluation of accuracy on the test set.

5.3.3 Evaluation of Results

The base model, which is made up of DSConv layers without grouping, achieves a competitive accuracy of 96.6% on the Speech Commands test set. The low-parameter model that uses GDSConv layers nearly matches this accuracy at 96.4%, despite having approximately half the parameters. These results affirm the efficacy of GDSConv for reducing the model's size.

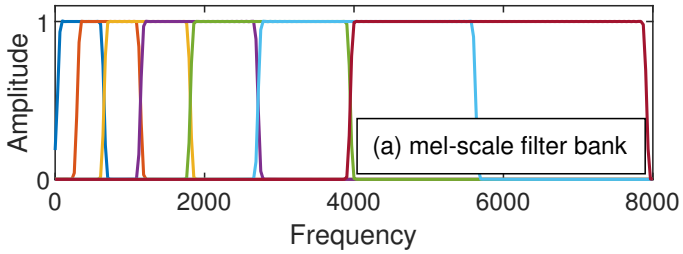
As detailed in Table 5.1, these results are compared to similar research. The network demonstrates a higher efficiency in terms of accuracy for a given parameter count when contrasted with the DSConv network in [217]. Our base model's accuracy exceeds their largest model by 1.2%, even though their model has roughly four times more parameters. Our results also surpass those of Choi *et al.* [40], improving accuracy for a given number of parameters. Their use of 1D convolution along the time dimension may suggest this method's usefulness for audio processing, particularly for KWS.

Unlike prior studies, the presented architecture obviates the requirement for preprocessing for feature extraction, attributed to the SincConv layer's capacity for direct raw audio sample feature extraction. Such a characteristic allows for the complete execution of inference as floating-point operations, eliminating the necessity for auxiliary hardware modules for the processing or transfer of preprocessed features. As a result, microcontrollers with diminished modules demonstrate enhanced energy efficiency. Moreover, no residual connections were used in this network architecture, as this adds memory overhead and increases challenges for hardware acceleration modules.

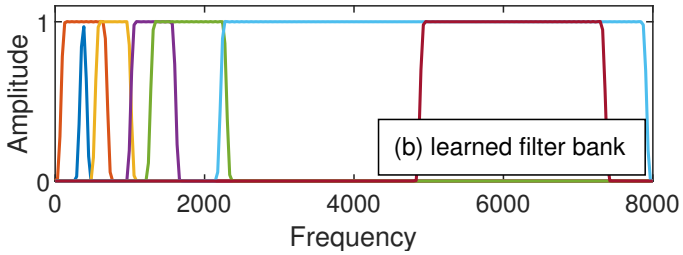
The architectures were also evaluated on the newer second version of the Speech Commands dataset to ensure future comparability. As shown in Table 5.1, this resulted in even higher accuracies of 97.4% and 97.3% for two investigated models, respectively. Models trained on this second version of the dataset tend to perform better on both the second version's test set and the first version's test set [193].

It's crucial to note that the comparative models presume preprocessed audio features and therefore do not account for the number of parameters or operations of any preprocessing steps. As our architecture incorporates feature extraction from raw audio with the SincConv layer, the total model parameters in Table 5.1 includes the SincConv layer. From a power-efficiency perspective, a single SincConv filter only requires two read operations to generate the filter for each channel. Power-optimized spectral transformations also involve the precomputation of filter maps [179].

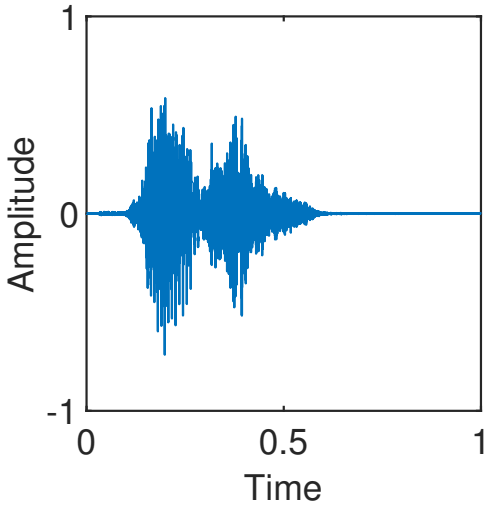
Figure 5.3 provides a visualization to understand the initial setup of the sinc filters and their progression during training. The stages from the initialization of the Sinc convolution using the Mel-scale filter bank to the fully trained filter is shown, Additionally, the latter part of this visualization represents the sinc filter's response to the keyword "Yes".



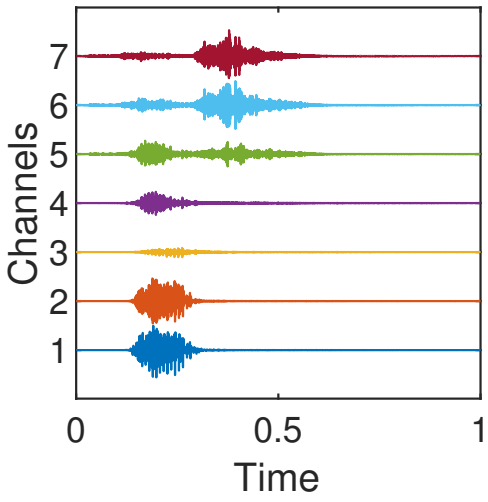
(a) Sinc filters are initialized with values of the Mel-scale filter bank.



(b) After training, the start and stop frequencies of the bandpass filters have shifted.



(c) Time signal of the keyword "Yes".



(d) Filter bank output of the trained Sinc convolution filter for the keyword "Yes".

Figure 5.3: From the Mel-scale filter bank as initialization for the Sinc convolution to the trained filter [20†]. The filter responses of a Sinc convolution layer to the keyword "Yes" with only seven channels is given as an example.

5.3.4 Discussion of Sinc Convolutions for KWS

Sinc convolutions present an efficient approach to apply neural networks directly on raw audio without necessitating preprocessing. Their outputs are more interpretable than conventional CNNs, and its light footprint – marked by a reduced parameter count – makes it well-suited for small embedded architectures. Additionally, GDSConv provides a means to further trim down the already small memory footprint of DSConv, without significant accuracy loss. Energy-efficient neural networks that maintain high accuracy are crucial for always-on, battery-operated devices performing keyword spotting. The architecture was constructed on the assumption that in a neural network, the number of parameters plays a significant role in power consumption, as memory access typically consumes more power than computation [34]. Based on this observation, an energy-efficient keyword spotting architecture combines SincConvs for feature extraction with GDSConv layers.

The discussed architecture employs parametrizable SincConvs to extract pertinent features from raw audio. The base model, composed of DSConvs – which already have fewer parameters than a regular convolution – attains a competitive accuracy on Google's Speech Commands dataset. Grouping convolutional channels to GDSConv, which further decreases the number of parameters, culminates in a low-parameter model with merely 62k parameters.

5.4 Sinc Convolutions in the Hybrid CTC/Attention Architecture

The motivation for integrating Sinc convolutions and depthwise separable convolutions into the hybrid CTC/attention architecture lies primarily in the aim to create a fully end-to-end trainable system that can work directly on raw audio data, eliminating the need for a separate, pre-processing step for audio features. Conventionally, ASR systems have depended on pre-processed frequency-domain features, handcrafted to emulate human hearing, as inputs. Such a pre-processing step adds a discrete stage to the ASR process and might not be fully optimal as these features are not learned during the training process.

Sinc convolutions, as integrated frontend for the ASR system, allow for spectral feature extraction directly from the raw audio input. This enables the model to learn more optimal feature representations from the raw data during the training process, potentially improving the model's performance. Depthwise separable convolutions, on the other hand, reduce the computational complexity and model size while maintaining similar performance characteristics as regular convolutions. This makes the model more efficient and lightweight.

Thus, the integration of these two types of convolutions aims to create a lightweight, low-parameter, machine-learnable feature extraction for ASR systems that can outperform conventional systems relying on pre-processed features, both in terms of model performance and computational efficiency. The architecture discussed in this section

was proposed by Kürzinger *et al.* [6†] and investigated in the works of Klewitz [12+] and Lindae [19+].

5.4.1 Lightweight Sinc Convolutions for End-to-End Decoding

The frontend is designed to derive more abstract latent representations from a sequence of raw audio frames $R = r_{1:T}$. This is accomplished by utilizing a layer of parameterized Sinc-convolution followed by multiple layers of depthwise convolutions. These layers transform the raw audio input into filter features, effectively substituting the traditionally precomputed log-mel filterbank features used in end-to-end ASR systems. The above architecture is referred to as Lightweight Sinc-Convolutions (LSC) and is depicted in Figure 5.4. This frontend architecture employs two essential building blocks, the Sinc convolution block and the Depthwise separable convolutional layers.

The Sinc convolution block proceeds in a sequential manner, initiating with the Sinc convolution layer, denoted as SincConv(c,k,s), which comprises c channels, a kernel size k, and stride s. This is immediately followed by log compression, 1D batch normalization, and 1D average pooling, which averages every two adjacent values, thereby halving the data length. The block for Depthwise separable convolutional layers, represented as DConvBlock(c, k, s), encompasses a depthwise separable convolutional layer characterized by c channels, kernel size k, and stride s. Subsequently, it integrates a leaky ReLU activation and a 1D batch normalization. During training, this block incorporates a dropout layer with a 15% probability. With these building blocks, the equation for the LSC frontend is given by:

$$\text{LSC}(R) = [\text{DConvBlock}]^5(\text{SincBlock}(R)) \tag{5.13}$$

Contrasting with precomputed filterbank features, Sinc convolutions can be incorporated within a neural network as a differentiable layer in relation to cutoff frequencies. For improved approximation of an ideal Sinc filter, the filter kernel weights are multiplied with the Hamming window. In standard hybrid DNN/HMM ASR, the window function is employed on the input data to mitigate artifacts during spectral transformation. Multiplying the kernel with the window function does not yield an equivalent result due to the non-distributive nature of convolution over multiplication. Nonetheless, this process attenuates edge artifacts in the kernel, mitigating spectral distortions in a similar manner. This method is more efficient, as it only requires a single application on the kernel filter rather than on each input frame. Figure 5.5 gives a comparison of a Sinc convolution kernel with and without a window function.

The Sinc convolution block applies log-compression as its activation function after the Sinc convolution, effectively compressing the output to a value range beneficial for further convolutional layers. Contrary to the ReLU activation function used in SincNet, log-compression discards less information, thus potentially enhancing the performance of classification on raw audio.

After the Sinc-convolutional block, multiple depthwise convolutions are employed to extract more abstract latent representations. These extract short-time context along the

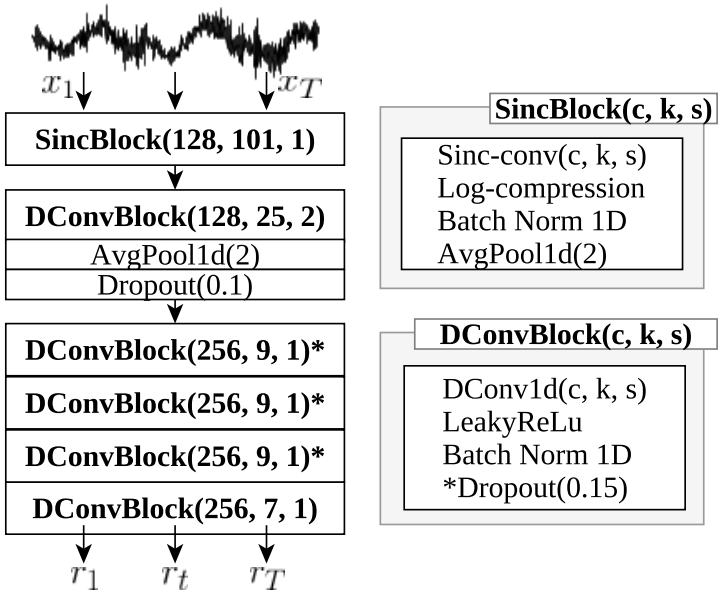
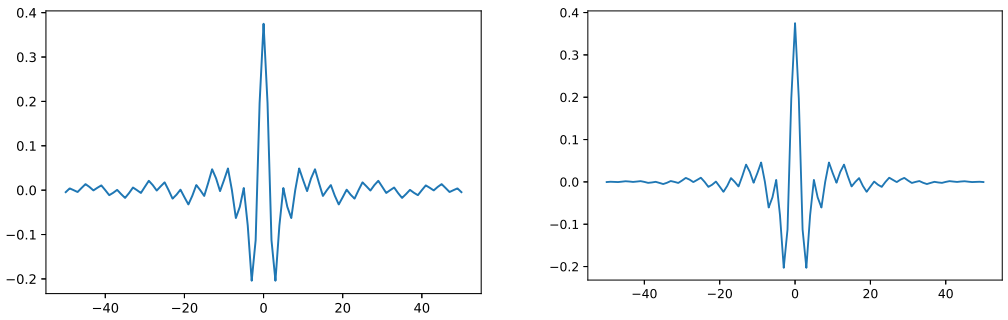


Figure 5.4: The figure illustrates Lightweight Sinc-Convolutions that are employed as a frontend for raw audio inputs. The process begins by segmenting the raw audio stream into discrete frames. Following this, Sinc-convolutions and several layers of depthwise convolutions are implemented for a low-parameter extraction of speech features.



(a) Sinc filter without window function (b) Sinc filter with window function

Figure 5.5: Comparison of Sinc filters with and without a window function.

time dimension but are restricted to data within a single frame. This segment of the architecture serves as a coupling layer between the raw audio signal and the encoder.

The feature extraction frontend has strong similarities with the lightweight keyword spotting architecture by Mittermaier *et al.* [20†], which employs depthwise separable convolutions. A key distinction is that the adaptation to ASR excludes the inter-frame pointwise convolution to further decrease the number of parameters. Depthwise convolutions contribute more significantly to high-level feature extraction than pointwise convolutions, while comprising only a small fraction of the learnable parameters.

5.4.2 Integration into the ASR Architecture

The back-end architecture, as depicted in Fig. 5.6, translates the features extracted by the LSC front-end into a sentence. This experiment uses the RNN-based model described in section 2.6. The encoder forms the first part of this process, which employs the front-end extracted features and is composed of BLSTM layers supplemented with projection layers. This maps the sequence of latent representations, denoted as $X = [x_1, \cdots, x_{T_X}]$, to a series of high-level representations expressed as $H = h_{1:T} = \text{BLSTMP}(x_{1:T_H})$.

The remaining architecture, i.e., the hybrid CTC/attention decoder, aligns with the standard RNN architecture described in Section 2.6. The transcription process from raw audio to a sentence incorporates beam search using shallow fusion, bolstered by a language model which has been trained on a substantial text corpus.

5.4.3 Experimental Evaluation

This architecture is assessed using the TED-LIUM 2 corpus [159]. The raw audio stream is segmented into frames of 25 ms each, with a 15 ms overlap between two successive frames, thereby resulting in a stride of 10 ms. These frame parameters are selected in line with the standard setting for MFCC features in the Kaldi toolkit [148]. Given that the audio is recorded at a sampling frequency of 16 kHz, each frame comprises of 400 samples with an overlapping portion of 240 samples.

The network training is executed using the Adadelta optimizer with parameters $\rho = 0.95$, $\epsilon = 10^{-8}$. The training process runs for 22 epochs before early termination halts the training; subsequently, the model exhibiting the highest validation accuracy is selected for evaluation. In this experiment, the Sinc convolution integrates 128 Sinc filters, which are initialized with mel filterbank weights.

These convolutions are selected such that the length of a frame is reduced to one, thereby leaving 256 features at the culmination of the feature extraction process. The encoder component of this network consists of four BLSTM layers with projection neurons of size 512. The decoder is composed of a single LSTM layer of size 512 coupled with an attention layer of the same size. The CTC/Attention weight during training is set to $\kappa = 0.5$. The model uses 500 unigram units as tokens created by SentencePiece, as described in Section 2.4.8.

Spectral augmentation, as described in Section 2.5.1, is employed as a data augmentation technique directly on the windowed raw audio signal. This is advantageous

compared to applying it on the extracted features; time warping introduces distortion into the audio signal, while masking functions similarly to dropout, thus enhancing the robustness of the model. The model training relied on the preset configuration as per the ESPnet toolkit; no additional adaptations or modifications were made. For the decoding process, a pre-established language model supplied by ESPnet was employed. Language model rescoring was conducted with a language model weight set at $\beta = 0.5$ and a CTC weight $\lambda = 0.4$.

5.4.4 Results

The performance of the proposed architecture is compared with a standard RNN baseline model that employs log-mel F-bank features, as constructed in Section 2.6, and the top-performing result on the TED-LIUM 2 corpus [195]. This result utilizes a VGGnet coupled with a BLSTM encoder and projection neurons, denoted as *VGG-BLSTMP*. Table 5.2 provides a parameter comparison between the two investigated models, i.e., using F-bank features and Sinc convolutions.

The proposed model, built for raw audio input, exhibits fewer learnable parameters than the VGG-BLSTMP model, primarily due to the integration of Sinc and depthwise convolutions. This results in LSC employing only 16k front-end parameters, compared to VGG's 259k parameters. The incorporation of pointwise convolution in the depthwise convolutional blocks would add an additional 240k parameters.

The back-end layers, smaller at 512, as opposed to VGG-BLSTMP's 1024, reduces the parameter count to 22.3m from 106m. The CTC layer, necessitating a short-term context, is facilitated by the BLSTM in the encoder, allowing for the application of frame-wise 1D filter convolutions without loss of inter-frame information.

Despite its reduced size, the model outperforms in terms of WER on the TED-LIUM 2 test set. The model's loss convergence during training exhibits a smooth, asymptote-like curve. On the other hand, the VGG-BLSTMP architecture's convergence curve is more step-like, with an initial warm-up phase, sudden improvements, and then a plateau.

Table 5.3 compares the performance of the LSC and the VGG-BLSTMP model on the TED-LIUM 2 dataset. The LSC model records a 0.9% decrease in WER, resulting in 11.7% on the test set, when decoded with a similar-sized language model (8.9m versus 7.4m parameters). The WER is further lowered by 1% to 10.7% when applying the larger language model used for ESPnet's transformer network on TED-LIUM 2, though it increases the parameter count to 139m. A more lightweight language model can be chosen for evaluation depending on the application context. For a clearer comparison, the parameter-reduced VGG-BLSTMP model, with only 19m parameters, is similar in size to the LSC model under evaluation. However, this model only achieved a WER of 18.6% when evaluated with the large RNNLM [102].

The employment of spectral augmentation was also found to enhance the network's performance. This is the case even without specific fine-tuning for raw audio data input. Spectral augmentation resulted in a WER improvement of 2.3% and 0.9%, when decoding was conducted without and with the large language model respectively.

The effectiveness of log-compression as an alternative activation function was also

evaluated. For this, a model instance was tested where the ReLU function was used as the activation function in the SincBlock (as depicted in Figure 5.4). With ReLU, slightly worse WERs of 11.0%/10.9% for the development and test set were achieved, confirming the findings of Mittermaier *et al.* [$20^{\dagger}$, 20^{+}]. This indicates that log-compression is advantageous for the extraction of filter features from raw audio. The results reinforce the premise that leveraging learnable feature extraction methods can bring about significant performance improvements and reductions in the complexity of ASR systems.

Figure 5.7 displays four of the Sinc-convolution filters that were learned through the LSC process, while Figure 5.8 offers a visual representation of both the default mel filters used for log-mel F-Bank features and the filters learned by the model. This was achieved by sorting these filters and then plotting their upper and lower limits.

These filters exhibited a noticeable shift towards higher amplitude and a wider band-pass in the spectral domain. Remarkably, one Sinc filter converged to a configuration that allowed for the passage of the entire raw audio signal, featuring a lower start and higher stop frequency of approximately 8kHz to 21kHz. This suggests the network's tendency to process the raw data directly. The kernel of this filter is presented in Figure 5.9. Apart from that, the learned filters still largely mirror the initial distribution mapped along the values of the log-Mel scale.

5.5 Integration into the ESPnet Toolkit

The LSC module for speech recognition from raw audio data, as detailed in Section 5.4.1, was submitted and integrated into the ESPnet toolkit. The provided module [$11^{\dagger}$], seamlessly integrates LSC into both RNN and Transformer models and offers recipes for datasets such as TED-LIUM 2, TIMIT, and Librispeech. The framework allows for the straightforward incorporation of Sinc convolutions, positioning it as a pre-encoder. The workflow with LSC follows a sequence: 1. The frontend module receives the raw audio and generates audio frames using a sliding window, 2. spectral augmentation serves as data augmentation (however, in time domain instead of spectral domain), 3. normalization, 4. LSC pre-encoder with Sinc convolutions and depthwise separable convolutions, 5. encoder, and 6. decoder. While there are resemblances between LSC and SincNet, significant distinctions exist, notably the inclusion of depthwise convolutions and log compression. Additionally, the module allows initialization of Sinc convolution parameters using the Bark scale instead of the Mel scale [219]. The bark scale's broader filters at lower frequencies can be advantageous for learnable filter parameters. During the module's development, it was noted that reducing grouping in the depthwise convolutional layers can increase complexity, impacting both memory consumption and training duration.

Table 5.1: Keyword spotting performance of discussed models on the Speech Commands dataset version 1 and version 2 [193].

Model	Accuracy (v1)	Accuracy (v2)	Parameters
DS-CNN-S [217]	94.1%	-	39k
DS-CNN-M [217]	94.9%	-	189k
DS-CNN-L [217]	95.4%	-	498k
ResNet15 [181]	95.8%	-	240k
TC-ResNet8 [40]	96.1%	-	66k
TC-ResNet14 [40]	96.2%	-	137k
TC-ResNet14-1.5 [40]	**96.6**%	-	305k
SincConv+DSConv	**96.6**%	97.4%	122k
SincConv+GDSConv	96.4%	97.3%	**62k**

Table 5.2: Architecture parameters for a model with F-bank features and for raw audio input.

	VGG+BLSTMP (ESPnet)	LSC+BLSTMP (Ours)
Feature Type	F-bank + Pitch	Sinc Convolutions
Input sample rate	16 kHz	16 kHz
Window function	Hann (Eq.)	Hamming
Windowing frame size	25 ms	25 ms
Windowing frame shift	10 ms	10 ms
Features frequency bins	$80 + 3$	128
Coupling	VGGnet	Depthwise Convolutions
Output dimension	2688	256
Size of the front-end	259k	**16k**
Size of the back-end	106m	**22.3m**
Size of the RNNLM	7.4m	8.9m / 139m

Table 5.3: Comparison of ASR results on the TED-LIUM 2 Dataset between the LSC and the VGG-BLSTMP model, on the development and test sets with and without employing a language model. The LSC model demonstrates a significant reduction in WER, achieving the lowest at 10.7% on the test set with the employment of a language model.

Model	Feature type	LM	Dev	Test
VGG-BLSTMP	F-bank	✓	12.8	12.6
VGG-BLSTMP (small)	F-bank	✓	19.8	18.6
LSC	raw	-	15.1%	15.7%
LSC+SpecAug	raw	-	13.5%	13.4%
LSC	raw	✓	11.4%	11.6%
LSC+SpecAug (ReLU)	raw	✓	11.0%	10.9%
LSC+SpecAug (logCpr)	raw	✓	**10.7**%	**10.7**%

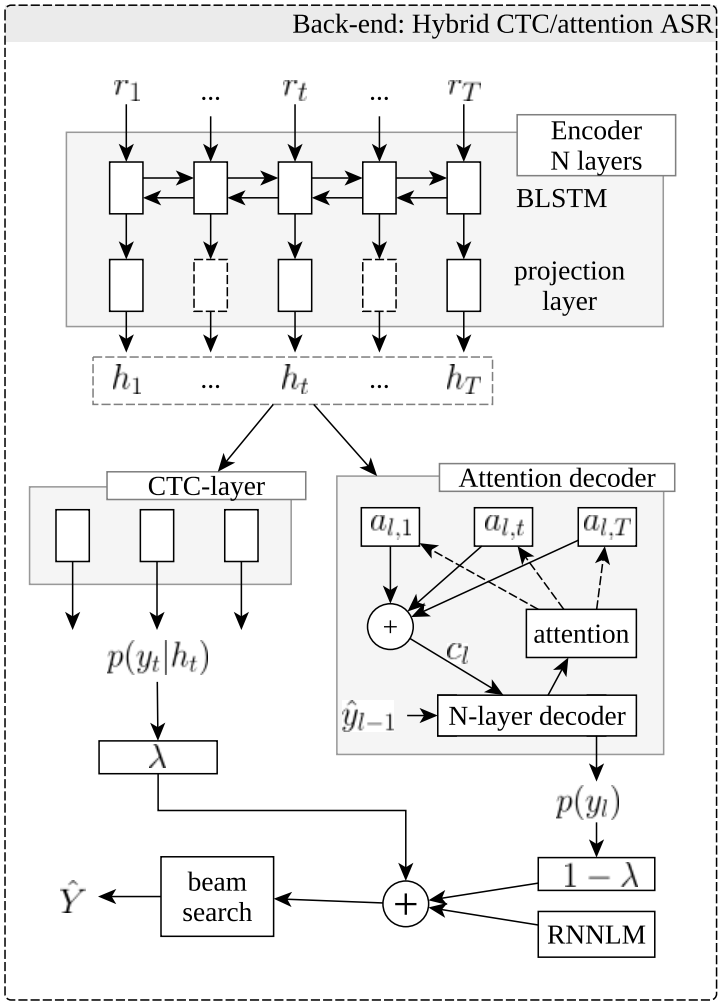


Figure 5.6: Back-end for the RNN-based architecture that classifies from raw audio: The front-end's extracted features are conveyed through a BLSTM encoder with projection neurons. The encoder states are subsequently employed by both a CTC and an attention decoder. Apart from the raw audio input module, this setup corresponds to the standard RNN architecture described in Section 2.6.

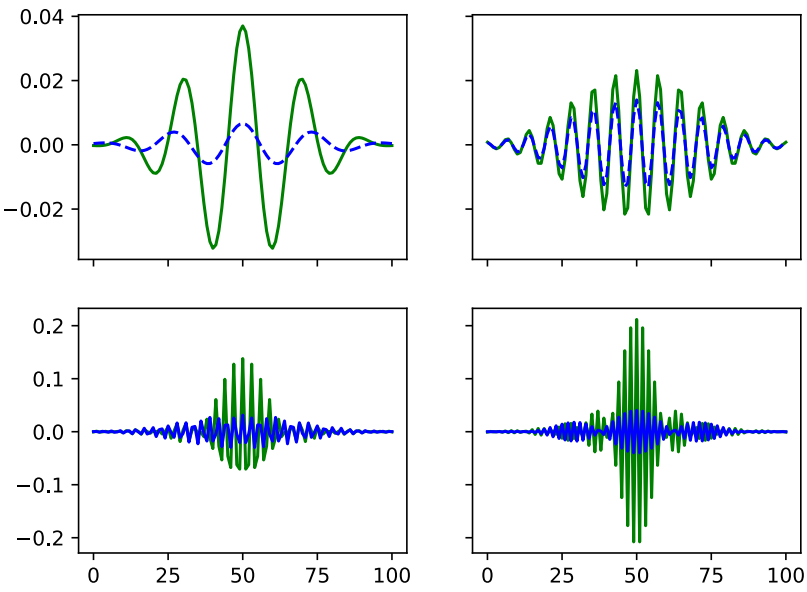


Figure 5.7: Four exemplary Sinc-convolution kernels. The corresponding mel-scale filters are marked with blue dashed lines. The kernels' center frequencies are arranged in a sequential order from top left to bottom right, and they are 790Hz, 2.2kHz, 5.4kHz, and 7.9kHz respectively.

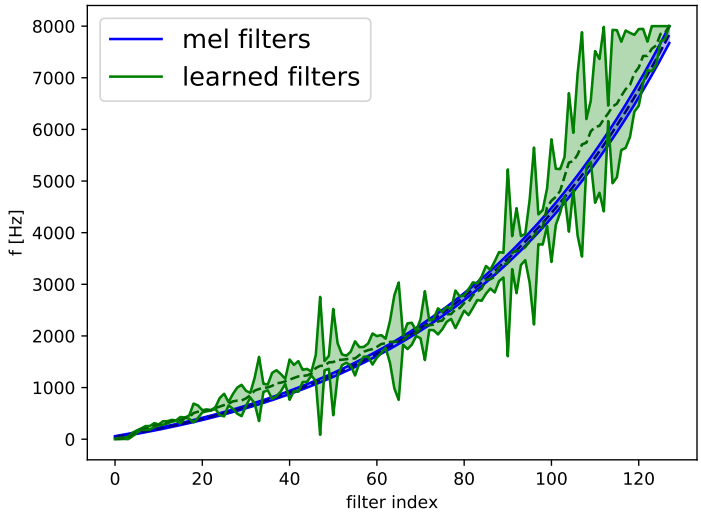


Figure 5.8: Lower and upper edges of initialized and learned Sinc-convolution filters are plotted for visualization.

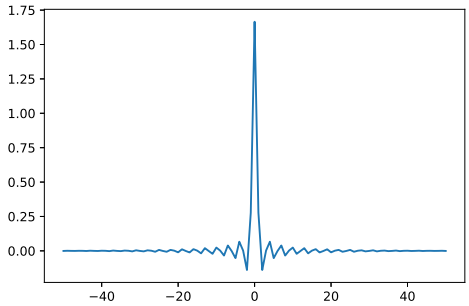


Figure 5.9: The filter kernel that ultimately converged to a passthrough filter. The structure of this filter is characterized by a large kernel value at the center, with near-zero values elsewhere. By convolving this filter with the audio signal, the signal essentially passes through untouched. This indicates that the network is also attempting to learn directly from the raw data.

6

CTC Segmentation

This chapter discusses CTC segmentation, an algorithm to determine utterance alignments within audio recordings. Previous approaches of aligning text to audio rely on pre-trained HMMs or TTS neural nets. CTC segmentation fills this gap and provides a tool for alignment using already pre-trained end-to-end ASR models. This algorithm was first explored by Winkelbauer [34^{+}] and published by Kürzinger *et al.* [10†].

CTC segmentation also generates a confidence score of how well each utterance fits to its audio segment. This score helps to filter out bad utterances, and thus is useful for automated corpus construction. The second half of this chapter illustrates this process through the development of both German and Japanese speech corpora.

After Section 6.1 reviews previous work on established alignment methods, it introduces the CTC segmentation algorithm. The algorithm is available as a Python package that is documented in Section 6.2. Section 6.3 shows the construction of a German dataset and provides an an example of how this additional training data improved ASR performance. Section 6.4 demonstrates advanced techniques to align large data exemplary on JTubeSpeech, a large Japanese dataset published by Takamichi *et al.* [21†].

6.1 CTC Segmentation

This section begins by examining traditional alignment methodologies, predominantly rooted in HMMs and the Viterbi algorithm. It then introduces the CTC segmentation algorithm that eliminates the need for HMM models, relying solely on a CTC-based speech recognition network for alignment generation.

6.1.1 Related Alignment and Segmentation Tools

Training data for ASR requires labeled speech data. Modern ASR systems take audio segments with text as training data. Utterances in a speech dataset comprise an audio segment of usually several seconds and the text as ground truth. In many cases, these utterances are cut from a longer audio recording which can be done manually or automated using alignment tools. Forced alignment tools determine the most probable alignment of a token sequence to an audio recording.

Many popular tools use the Viterbi or Baum–Welch algorithm to align phonemes based on HMMs. The program first encodes the ground truth text as a graph of phoneme sequences. These sequences are aligned with the phoneme state probability time sequence obtained from a pre-trained acoustic model. Utterance segments are derived from phoneme timings. HMM-based forced alignment is a popular approach included in many speech recognition toolkits; such as the Munich Automatic Segmentation (MAUS) system [162] of the Hidden Markov Model Toolkit (HTK [209]), or Gentle that is included in Kaldi [148].

The DeepSpeech aligner tool DSalign [97] adopts a more flexible approach compared to other forced alignment tools that require correct data. Text preprocessing and normalization is performed on the ground truth transcriptions to facilitate alignment with the automatic speech recognition transcripts. The speech audio is divided into fragments of approximately 15 seconds and transcription of all audio fragments is performed. The preprocessed text and transcribed audio are roughly aligned using the Smith–Waterman algorithm and non-fitting audio segments are removed. This tool requires an additional alignment step that removes any artifacts from splitting the audio, e.g., gaps in the transcription.

In an alternate approach, the ground truth text is synchronized to the audio using audio that was generated from the transcription. The Aeneas [144] alignment tool uses synthetic audio from a text-to-speech (TTS) module. From synthetic and original audio, Aeneas uses the Dynamic Time Warping (DTW) algorithm to generate a synchronization map, and subsequently determine utterance alignments.

6.1.2 The CTC Segmentation Algorithm

CTC segmentation is an algorithm to align text to audio using a pre-trained end-to-end ASR model with a CTC output layer. Using the output probabilities of a CTC-based ASR network, the algorithm determines transition probabilities and token timings. As a distinct advantage, acoustic models to classify HMM states are no longer necessary, but only the CTC output probabilities. The HMMs of acoustic models used in Gentle or MAUS infer phoneme probabilities for each time step. In contrast to labeling each time step with a classified phoneme, the CTC layer classifies the occurring time step of a token.

A CTC network infers frame-based character posteriors $p(c_j|t, X)$ from a given audio recording. The audio recording contains multiple utterances, i.e., sentences, that are consecutively encoded to a sequence of tokens. Given a sequence of tokens c_j with $j \in [1; M]$, the algorithm determines transition probabilities over the CTC output frames $t \in [1; T]$. The algorithm returns these probabilities in the form of a table with N rows and M columns, in which the i-th row corresponds to the i-th audio frame and the j-th column corresponds to the j-th token. The cells of this table contain the maximum joint probabilities $k_{t,j}$ of segment alignments that are obtained using dynamic programming

by the following rules:

$$p_{\text{stay}} = k_{t-1,j} \cdot p(\text{blank}|t) \tag{6.1}$$

$$p_{\text{transition}} = k_{t-1,j-1} \cdot p(c_j|t) \tag{6.2}$$

$$k_{t,j} = \begin{cases} \max(p_{\text{stay}}, p_{\text{transition}}) & \text{if } t > 0 \wedge j > 0 \\ 0 & \text{if } t = 0 \wedge j > 0 \\ 1 & \text{if } j = 0 \end{cases} \tag{6.3}$$

Each maximum joint probability $k_{t,j}$ is determined by what transition is more probable: $k_{t-1,j} \cdot p(\text{blank}|t)$ represents the probability that transition path continues at the current token index j and $k_{t-1,j-1} \cdot p(c_j|t)$ gives the probability of a transition from the previous token.

To distinguish start and ends of utterances, separators are inserted into the token sequence[i]. Unrelated audio parts at the beginning of the recording are skipped by setting the transition cost for the first token to zero. This approach effectively allows the algorithm the flexibility to begin when the token probability signals the presence of the first token in the utterance. This capability to start aligning at the most pertinent section also sets CTC segmentation apart from other forced alignment methods.

Token alignments are then obtained by backtracking from the frame with the highest joint probability of the last token k_M at $t = \arg\max_{t'} k_{t',M}$. The highest transition probability determines the alignment a_t, i.e., the token index the audio frame t is aligned to, as

$$a_t = \begin{cases} M-1 & \text{if } t \geqslant \arg\max_{t'}(k_{t',M-1}) \\ a_{t+1} & \text{if } k_{t,a_{t+1}} \cdot p(blank|t+1) > k_{t,a_{t+1}-1} \cdot p(c_j|t+1) \\ a_{t+1}-1 & \text{else.} \end{cases} \tag{6.4}$$

The alignment probability ρ_t to every audio frame is determined from the alignment a_t as $\rho_t = p(c_{a_t}|t)$. A confidence score s_{seg} for each utterance helps to evaluate how well each utterance was aligned, or, how well the recording segment fits to the ground truth utterance text. Audio frames within the given utterance are split into parts of length L. L is chosen as the equivalent to one second which depends on the ASR model, for example, $L = 30$ in the work described in Section 6.3. Then, the alignment probabilities within each part are averaged to a mean value m_j. Finally, the confidence score s_{seg} is defined as the lowest of averages within the utterance:

$$s_{\text{seg}} = \min_j m_j \quad \text{with} \quad m_j = \frac{1}{L} \sum_{t=jL}^{(j+1)L} \rho_t \tag{6.5}$$

Taking only the lowest minimum probability helps to minimize and filter out utterances with mismatches between audio and text. In this way, long utterances exhibit low confidence scores if a single word is a mismatch or missing in the transcription.

[i]The implementation uses the label of the blank token as separator, as it is never used in the ground truth text.

Figure 6.1 provides an example of the first step, where all state probabilities are written in a large table. The second step, shown in Figure 6.2, involves the backward pass that determines the most probable path through the trellis. Finally, Figure 6.3 visualizes the determination of the confidence score.

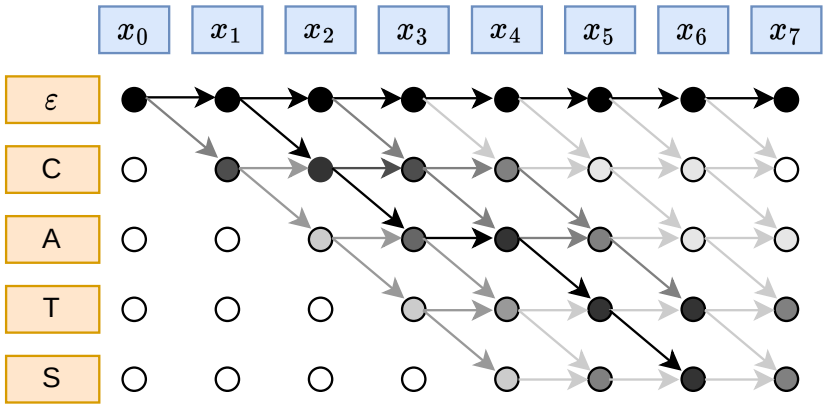


Figure 6.1: Step 1: Calculate all stay and transition probabilities.

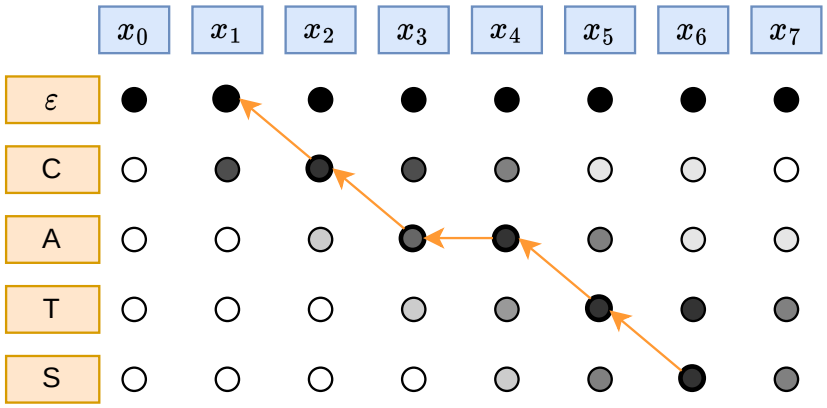


Figure 6.2: Step 2: Determine the most probable path in the backward pass.

	C	A	T	S	-	A	N	D	-	D	O	G	S
ρ_t	0.9	0.8	0.85	0.8	0.5	0.2	0.4	0.1	0.7	0.9	0.8	0.7	0.9
m_j			0.84				0.30				0.80		
s_{seg}							$\min\{0.84, 0.30, 0.80\} = 0.30$						

Figure 6.3: Step 3: Calculate the alignment score by averaging alignment probabilities within each part of an utterance (e.g., 1-second segments) and selecting the lowest average.

The runtime complexity of CTC segmentation can be reduced based on two assumptions. Under the assumption that the ratio of audio frames per token is nearly constant,

the complexity is reduced to $O(M)$ instead of $O(M \cdot N)$. A further optimization assumes that the audio position is proportional to the token sequence position. With that, not all probabilities $k_{t,j}$ are required, but only the audio frames in the interval $[t - W/2, t + W/2]$ with $t = jN/M$ and window size W.

6.1.3 Alignment of an Example Sentence

Figure 6.4 gives an example of CTC segmentation. This utterance is included in the ESPnet toolkit as examples sentence[ii] that was derived from the WSJ dataset [45]. It is transcribed as:

THE SALE OF THE HOTELS IS PART OF HOLIDAY'S STRATEGY
TO SELL OFF ASSETS AND CONCENTRATE ON PROPERTY MANAGEMENT

The upper plot shows the raw audio waveform with estimated alignments. The second heatmap contains the CTC layer activations over the audio time. Here, the lowest row shows the blank at index 0 that has the most activations, the other tokens in the order of their appearance in the dictionary are

|, E, T, A, O, N, I, H, S, R, D, L, U, M, W, C, F, G, Y, P, B, V, K, ', X, J, Q, Z.

The third plot contains the path probability matrix; from this matrix, the most probable alignments are estimated.

6.1.4 Evaluation of Alignments

The TED-LIUM 2 dev and test sets comprise recordings from 19 unique speakers and serve as datasets for evaluation. In this dataset, long audio recordings of TED talk presentations are labeled with begin and end timestamps of separate sentences. These alignments have been done manually and serve as reference for the evaluation.

Alignments are validated by three metrics, (1) the mean deviation of segment timings from the ground truth, (2) their respective standard deviation and (3) the ratio of alignments within 0.5 seconds from ground truth timings.

As ASR model type also has a certain impact on CTC segmentation. This evaluation is done with pretrained RNN-based and Transformer-based ASR models[iii]provided by the ESPnet toolkit [102, 195].

Results are compared with three other established forced alignment tools: MAUS, Gentle and Aeneas. As these tools yield phone-wise alignments, this experiment obtains the utterance timings from the onset of the first phoneme, and the offset of the last phoneme of the corresponding utterance.

[ii]The example audio file in the ESPnet git repository at `test_utils/ctc_align_test.wav`.

[iii]The experiment employs pretrained models from ESPnet v1. The Transformer model utilized features a self-attention encoder with 12 layers, each comprising 2048 units. Conversely, the RNN model comprises an encoder with a VGGnet-preencoder, bidirectional LSTM units, and projection neurons. Its encoder encompasses four layers, each containing 1024 units with sub-sampling occurs in the second and third layers, as described in Section 2.6. On the TED-LIUM 2 test set, the Transformer yields a WER of 10.4, while the RNN model achieves a WER of 12.6.

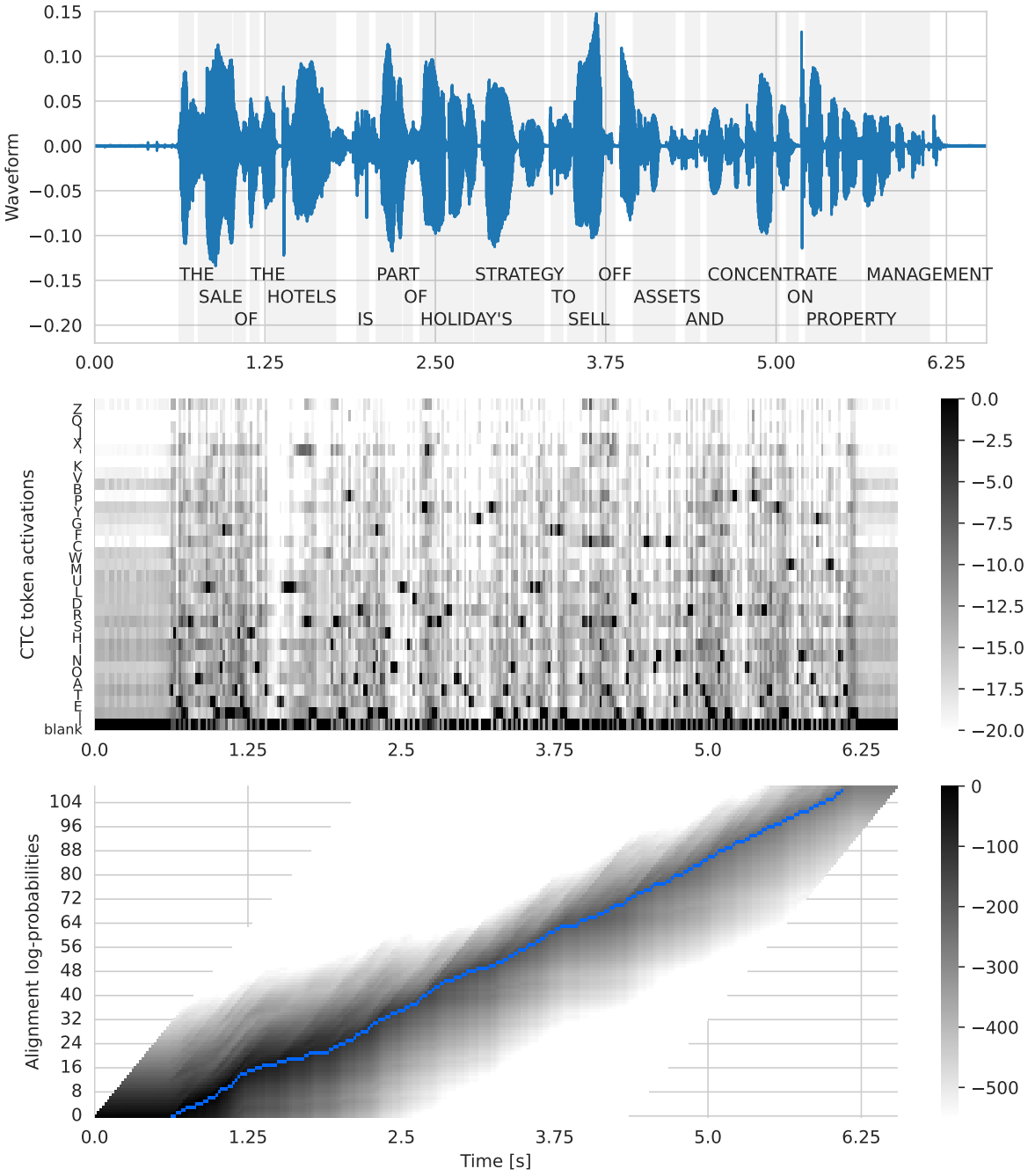


Figure 6.4: Aligned example sentence. The plot above shows the raw audio waveform with estimated alignments. The second heatmap contains the CTC layer output; the blank token (at index 0) is in the lowest row. The bottom image displays the path probability matrix; the central line marks the most probable path through the trellis. This illustration is inspired by Hira's forced alignment visualization [82, 205].

Table 6.1: Accuracy of alignments generated by CTC segmentation, MAUS, Gentle and Aeneas on the TED-LIUM 2 dev/test sets. Alignments are validated by three metrics, (1) the mean deviation of segment timings from the ground truth, (2) their respective standard deviation and (3) the ratio of alignments within 0.5 seconds from ground truth timings.

	Mean	**Std**	< 0.5**s**
Other alignment tools			
MAUS (HMM-based using HTK)	1.38s	11.62	74.1%
Aeneas (DTW-based)	9.01s	38.47	64.7%
Gentle (HMM-based using kaldi)	0.41s	1.97	82.0%
CTC Segmentation			
RNN trained on TED-LIUM 2	0.34s	1.16	90.1%
Transformer trained on TED-LIUM 2	0.31s	0.85	88.8%
Transformer trained on Librispeech	0.35s	0.68	85.1%

Table 6.1 shows the alignment results. CTC segmentation produced alignments that are significantly closer to the manually labeled ground truth when compared to segment timings generated by other tested alignment algorithms.

While alignment metrics in Table 6.1 demonstrate that there is a certain difference in timing accuracy, it does not distinguish between utterance start and end timings. Figure 6.5 visualizes how these timings are distributed in relation to the ground truth. The histogram contains alignment distributions generated by CTC segmentation and the Gentle alignment tool. In both cases, the majority of timing deviations are smaller than one second. Alignments from CTC segmentation are mostly within 0.5 seconds with a mean deviation of 0.35s, closely followed by the accuracy of HMM-based Gentle alignments that exhibit a mean deviation of 0.41s. These experimental results on the TED-LIUM 2 dataset indicate that alignments from CTC segmentation are more accurate when compared to Viterbi- or DTW-based algorithms.

CTC segmentation distinguishes transitions between utterances from transitions between tokens. The stay probability p_{stay} in the trellis is not reduced at start and end of an utterance, as detailed in Equation 6.8 and 6.3. With that technique, unrelated audio segments such as preambles can be automatically skipped in a robust manner. Also, the algorithm may detect deviating transcriptions using the confidence score. To evaluate how well CTC segmentation handles unrelated audio segments in comparison to other alignment tools, a preamble and postamble are added to each of the utterances in the TED-LIUM 2 dev and test set. The preamble is taken from the last t_{pre} seconds of every audio utterance and the postamble from the first t_{post} seconds, where t_{pre} and t_{post} are randomly chosen from the uniform distribution in the interval $[10, 30]s$.

Table 6.2 compares the alignment algorithms in terms of accuracy to the ground truth labels. In general, MAUS and Aeneas seem to not skip unrelated segments but rather force alignments within the unrelated audio parts. Alignments from Gentle and CTC segmentation exhibit the highest accuracy in these cases and skip any unrelated

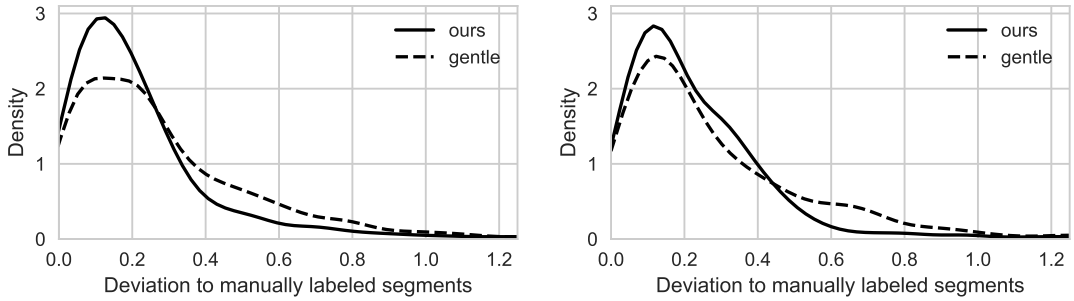


Figure 6.5: Histogram of alignment timings generated by Gentle and CTC segmentation, compared to manually labeled segments [10†]. The plot shows the relative deviation in seconds of generated segment start and end timings. CTC segmentation with the RNN-based ASR model generated more accurate timings at the start of the segment (left) than Gentle; and generated slightly more accurate segment ending timings (right). The y axis denotes density in the histogram with 60 bins.

preambles.

Table 6.2: Accuracy of alignments generated by CTC segmentation, MAUS, Gentle and Aeneas that are evaluated on the modified TED-LIUM 2 dev and test sets with preambles and postambles. Alignments are validated by three metrics, (1) the mean deviation of segment timings from the ground truth, (2) their respective standard deviation and (3) the ratio of alignments within 0.5 seconds from ground truth timings.

	Mean	**Std**	< 0.5**s**
Other alignment tools			
MAUS (HMM-based using HTK)	3.18s	18.97	66.9%
Aeneas (DTW-based)	10.91s	40.50	62.2%
Gentle (HMM-based using kaldi)	0.46s	2.40	81.7%
CTC Segmentation			
RNN trained on TED-LIUM 2	0.40s	1.63	89.3%
Transformer trained on TED-LIUM 2	0.35s	1.38	89.2%
Transformer trained on Librispeech	0.40s	1.21	84.2%

The timing data from the two experiments suggests that alignments from CTC segmentation are closer to the manually labeled ground truth than alignments that were generated with DTW- and HMM-based tools. The models trained on the TED-LIUM 2 corpus performed better on the test/dev set of the same corpus. Nevertheless, the Transformer model trained on Librispeech still generated more accurate alignments when compared to Gentle. The Transformer model also generated more accurate alignments than the RNN model, which is likely a consequence of the more powerful encoder and improved ASR performance.

6.1.5 Accuracy Constraints

CTC loss fine-tunes the neural network using weak temporal labels, i.e., training towards a specific sequential ordering of labels without predetermining a label for each time step. As such, CTC loss is indifferent to corresponding time shifts and tends to maintain the sequential token order, even if the acoustic information of output tokens is not arranged in a temporally sequential manner. Watanabe *et al.* [196] noted that this CTC's monotonic alignment property mirrors the same alignment characteristic found in conventional hybrid DNN/HMM ASR systems. Graves highlighted the significance of the CTC outputs for alignments:

> This 'collapsing together' of different paths onto the same labelling is what allows CTC to use unsegmented data, because it removes the requirement of knowing where in the input sequence the labels occur. In theory, it also makes CTC unsuitable for tasks where the location of the labels must be determined. However in practice CTC networks tend to output labels close to where they occur in the input sequence. (Graves [68])

Building on this, Graves also demonstrated an experiment where CTC effectively predicted both the labels and their approximate placements [59, 68].

The alignment accuracy of CTC segmentation strongly depends on the accuracy of the CTC activations. Several factors can impact the accuracy of CTC activations, including the performance of the ASR model, audio quality and label quality. For example, background noise may lead the algorithm to detect an early start of the sentence.

While the accuracy of CTC activations is never a direct optimization goal of CTC loss during training; nevertheless, the CTC-based alignments are mostly accurate within 500ms of hand-labeled alignments, as observed in Section 6.1.4. Still, CTC posteriors are not always accurate, temporal displacements of 500ms or more can occur in rare cases. Such misalignments arise under specific conditions but can be addressed with additional methods.

Such inaccuracies often manifest at the beginning and end of an aligned audio file. Bakhturina *et al.* noticed that sometimes the conclusion of the last utterance is truncated prematurely [11]. Their solution was to adjust the alignment of the final utterance based on a threshold set on the mean absolute signal. Side information from acoustic activity then helped to synchronize the alignment with the audio.

A distinct form of inaccuracy, primarily for Japanese speech data, was noted by Yin *et al.* [208]. Their alignment inaccuracies were considerably reduced by prepending a few seconds of audio to each audio file.

Similar to the accurate estimation of alignments, the accuracy and temporal ordering of token probabilities also affect the confidence score. This score estimates the log-space probability of the match between the transcription and audio data, and is significantly impacted by factors such as the quality of the ASR models, input data, and pre-processing. Consequently, the choice of ASR model affects which utterances are filtered out based on their confidence scores. Additionally, confidence scores can serve as a diagnostic

indicator, as atypical scores on utterances may signal a misalignment between the audio, transcription, and speech recognition model.

6.2 Python Package and Toolkit Integration

The CTC segmentation algorithm was published as a Python package [9†], suitable for aligning text to audio with any CTC-based ASR network. This Python package is adapted for and integrated into several toolkits, in Espnet [195], Speechbrain [158] and NeMo [11]. These toolkits provide user-friendly access to pre-trained models and a direct Python command line interface to CTC segmentation.

Kürzinger *et al.*'s initial version [10†] was set up as an ESPnet 1 recipe. Similar to the kaldi-toolkit, the ESPnet 1 toolkit uses recipes that are corpus-specific. However, the updated package introduces a feature for direct alignment from tokenized label sequences, enhancing speed and memory efficiency.

A second subsection introduces a modification to the algorithm itself; the CTC segmentation algorithm differentiates itself by efficiently skipping unrelated audio segments at the start of a recording. An additional modification has been made to the algorithm to filter out unrelated audio throughout the recording by adjusting the transition probabilities, which can potentially lead to skipping sections if better token matches are identified later.

The third subsection discusses the computational resources required for the alignment process, emphasizing that while the dynamic programming algorithm used is efficient, the bulk of computational demand arises when obtaining CTC activations, especially with Transformer models on long audio sequences.

6.2.1 Text Preprocessing

The accurate preparation of the ground truth token list for alignment determines the quality of alignments. A tokenizer in many speech recognition toolkits typically produces this list. However, when dealing with texts that haven't been previously tokenized, there arises a need for tokenization.

A challenge with many Byte-Pair Encoding (BPE) or unigram dictionaries is that they often possess overlapping word parts of varied lengths. Directly aligning from the transcribed token sequence is ideal for character-based ASR models but might reduce accuracy with BPE models that have many tokens. For instance, the token "cat" might be represented not only by itself but also by its individual segments: "c", "ca", and possibly their duplicates with a word-boundary delimiter.

While ASR models would correctly transcribe sentences with these tokens, the inherent ambiguity in these overlapping tokens can lead to misalignments. The reason being, the CTC activations might respond to both the token as a whole and its parts. This can particularly be problematic for models with extensive dictionaries, which encapsulate a plethora of word combinations and their constituent parts.

To mitigate such ambiguities, a preprocessing step is designed to manage the partial sequences. By identifying and handling these partial character sequences, it ensures a more accurate alignment.

Consider a scenario where we are given a text with a value "cat" and a dictionary[iv] containing both the word cat and its constituent parts: "•", "UNK", "a", "c", "t", and "cat". When employing an already tokenized list, the resulting ground truth token list simply contains the individual index of the "cat" token, i.e., (5).

This label sequence will then be aligned as the following token sequence:

$$C = \begin{bmatrix} -1 \\ 0 \\ 5 \\ 0 \end{bmatrix} \tag{6.6}$$

The first entry also contains (-1) which denotes the starting token as a helper for the algorithmic processing. Generally, all entries with (-1) are ignored and act as placeholders. All 0 entries denote separators at the start, at the ending of the label sequence and between utterances.

On the other hand, splitting the text sequence into partial sequences results in:

$$C = \begin{bmatrix} -1 & -1 & -1 \\ 0 & -1 & -1 \\ 3 & -1 & -1 \\ 2 & -1 & -1 \\ 4 & -1 & 5 \\ 0 & -1 & -1 \end{bmatrix} \tag{6.7}$$

In this instance, the algorithm identifies both the partial characters $(3, 2, 4)$ and the complete "cat" token represented by (5); furthermore, it computes transition probabilities for both combinations, and chooses the more probable sequence. The intention behind this process is to achieve enhanced time resolution for the alignment. Yet, this introduces an added layer of complexity to the algorithm. Since alignment probabilities must be computed for potential token combinations, there's a modest rise in computational overhead, while the memory overhead increases in proportion to the maximum token length in the model's dictionary.

To give an example, the RNN model employed for alignment incorporates a set of 500 tokens with a token length that can reach up to 10. This set comprises entire words, such as "different" and "actually". As an illustration, the word "something" can be tokenized starting with the parts "s", "so", "som", "some", "somet", and the complete word "something", highlighting the range of token sizes in the list.

Models based on characters, which exclusively use character tokens, avoid these ambiguities during tokenization and do not require this type of partial token text preprocessing.

[iv] Note that indices in this dictionary start from zero. The first token in the dictionary is the blank token, that is represented by a placeholder "•". On the other hand, "UNK" serves as the token representing the unknown class.

6.2.2 Adaptions to the Algorithm

A key differentiator of CTC segmentation compared to other forced alignment tools is the ability to skip unrelated audio parts at the start of a recording, such as introductory music, ambient noise, or unrelated chatter, by setting the transition cost for the first token to zero. However, many recordings not only start with unrelated segments, but also contain these unrelated parts interspersed within the audio recording itself.

Thus, the aim of an additional modification to the algorithm is to extend the existing algorithm to effectively filter out these unrelated segments, both at the beginning and throughout the recording, ensuring a more cohesive and relevant audio output. To skip these intermediate unrelated parts of the recording, the transition of the separator token can be set to zero. This is achieved by distinguishing an additional case in Equation 6.3 during the calculation of the transition probabilities $k_{t,j}$,

$$k_{t,j} = \begin{cases} \max(k_{t-1,j},\, p_{\text{transition}}) & \text{if } (t > 0 \wedge j > 0) \vee c_j = \text{separator} \\ \max(p_{\text{stay}},\, p_{\text{transition}}) \cdot p(c_j|t)) & \text{if } (t > 0 \wedge j > 0) \vee c_j \neq \text{separator} \\ 0 & \text{if } t = 0 \wedge j > 0 \\ 1 & \text{if } j = 0 \end{cases} . \tag{6.8}$$

The equation employs the stay and transition probabilities consistent with those in Equation 6.1 and Equation 6.2, which are reiterated below for clarity.

$$p_{\text{stay}} = k_{t-1,j} \cdot p(\text{blank}|t) \tag{6.9}$$

$$p_{\text{transition}} = k_{t-1,j-1} \cdot p(c_j|t) \tag{6.10}$$

The expression $\max(k_{t-1,j},\, p_{\text{transition}})$ generates a stable region or plateau for the probability. This allows the transition path to remain within this region if a suitable path isn't identified within the token sequence of the subsequent utterance. Consequently, this can lead to parts of the audio being skipped if sequences with more fitting token probabilities emerge later on.

6.2.3 Computational Resources

For the study in Section 6.3, numerous audio files span over three hours. When the text aligns appropriately with the audio, the alignment process is expected to complete within approximately 500 milliseconds[v], even for audio files spanning several hours. If the provided text lacks relevance or accuracy, the algorithm may require several seconds before indicating unsuccessful alignment, though such an outcome is not anticipated in this context.

The dynamic programming algorithm operates in a sequential manner, utilizing only a single CPU thread. However, using a multicore CPU, the process can be accelerated by conducting alignments on multiple files simultaneously, with the primary constraint

[v] Experiments were conducted on a PC with 64GB RAM, Nvidia RTX 2080ti, and an AMD Ryzen 7 2700X.

being memory consumption. Given the aligned token sequence length L_{token} and the length of the CTC output L_{ctc}, the memory consumption M is roughly proportional to their product,

$$M \propto L_{\text{token}} \times L_{\text{ctc}}. \tag{6.11}$$

Additionally, this proportionality is influenced by factors such as the floating point precision, typically 32 bits, and the sum of aligned tokens and utterances, and token length, as elaborated in Section 6.2.1.

Compared to the alignment using CTC segmentation, ASR inference to acquire the CTC activations typically requires significantly more computational resources. Therefore, optimizing alignment runtime should focus on the inference step for RNN or Transformer models. Although RNNs are rarely used in modern systems, Transformer-based networks have become the norm. Nevertheless, conventional Transformers demand substantial resources, especially for extended sequences. On extended audio files, Transformers tend to exhaust memory, whereas RNNs can handle the inference. However, it is not necessary to train an RNN for alignment of long audio files. Section 6.4.6 proposes a possible approach for this issue by splitting up the audio file in several parts.

6.3 Alignment of a German Corpus

This section outlines the construction of a German corpus as presented by Kürzinger *et al.* [10†] and in the work of Winkelbauer [34⁺]. It introduces related corpora and provides an example of text preprocessing, which is essential for preparing the text to obtain label sequences that assist the ASR model in aligning the text with the audio. The section also offers a comparison of German speech datasets and their selections for training. Results suggest that increased training data enhances speech recognition performance. Additionally, performance is further improved when realigned data from CTC segmentation is used.

6.3.1 Related Corpora

Previous work on publicly available German speech datasets was done by Milde *et al.* [127] who combined freely available German language datasets. Their work also provided a DNN/HMM model of the Kaldi toolkit trained with this collected dataset. In summary, their collection contains the Tuda-DE dataset [152], the Spoken Wikipedia Corpus (SWC, [13]) and the M-AILABS Speech Dataset [173]. See Table 6.3 for a more detailed description about these datasets and example sentences. The collection of datasets used in this study further includes the CommonVoice dataset [8] and the Librivox [117] dataset. LibriVox only provides audio files, and thus the corresponding ground truth text was obtained from the Gutenberg Project [75]. A similar collection that also includes non-free speech datasets was used by Denisov *et al.* [51]; their work uses the RNN-based hybrid CTC/attention ASR model of the ESPnet toolkit.

Table 6.3: Overview of German speech datasets utilized in the study, along with a brief description of each.

Dataset	Description
Tuda-DE [152]	Various topics narrated by 180 individuals, recorded using five different microphones.
Spoken Wikipedia Corpus / SWC [13]	Audio versions of Wikipedia articles, narrated by community volunteers.
M-AILABS [173]	Audio content sourced from political discourses and LibriVox book narrations.
CommonVoice [8]	Crowd-sourced single-sentence recordings, both contributed and reviewed by the public.
Librivox [117]	Public domain literary works narrated by volunteers and hosted on the LibriVox platform.

6.3.2 Text Preprocessing

Chapters from LibriVox audiobooks and Wikipedia articles provide the ground truth text to the corresponding audio files, but require certain preprocessing steps before its intended use as speech dataset.

Preprocessing steps include a conversion to the form of the Tuda-DE dataset that includes Latin letters from a-z and German umlauts[vi], but no numbers or punctuation tokens. Therefore, text preprocessing replaces any numbers and abbreviations contained in the original transcriptions by their spoken equivalent. Pronunciation of numbers differs depending on their context that can be determined by simple heuristics using an NLP tagger. For example, *"**18**00 Soldaten"* is spoken as *"**eintausendacht**hundert Soldaten"*, whereas *"Es war **18**00"* is pronounced as *"Es war **achtzehn**hundert"*. Number conversion is performed using the NLP tagger of the spaCy toolkit [86].

Older texts require an additional processing step; LibriVox consists of books with expired copyright that are at least 70 years old and therefore are written in old German orthography. Here, the particular texts are converted to the reformed German orthography with an automated lookup-table of word replacements.

After text conversion, the preprocessing splits long text at punctuation marks to derive single-sentence utterances from a continuous text.

6.3.3 German Speech Datasets and Selections for Training

The following experiment builds upon a previous corpus collection, and appends additional speech datasets obtained via CTC segmentation. Table 6.4 gives an overview of the specific datasets.

The construction of the German speech training dataset comprised three distinct selections:

[vi] The supported German umlauts are ä, ü, ö and ß.

Table 6.4: The German speech training data resources are used for corpus selection in model training.

Datasets		Length	Speakers	Utterances
Tuda-DE train [152]	TD	127h	147	55497
Tuda-DE dev	dev	9h	16	3678
Tuda-DE test	test	10h	17	4100
SWC, aligned by Milde *et al.* [127]	SW	285h	363	171380
M-ailabs	MA	237h	29	118521
Common Voice	CV	319h	4852	279516
SWC, aligned by CTC segmentation	SW*	210h	363	78214
Librivox, aligned by CTC segmentation	LV*	804h	251	368532

1. Milde Selection: Milde *et al.* [127] curated a dataset by integrating Tuda-DE, SWC, and M-AILABS. This culminated in 649 hours of speech data. Notably, recordings from Tuda-DE made with the Realtek microphone were intentionally omitted, a decision that is mirrored in this study as well. Speech data of the SWC in this selection was aligned with the Viterbi algorithm of the Sphinx toolkit [113].
2. CommonVoice Inclusion: This selection extends the dataset to 968 hours by further including the CommonVoice dataset. During the preprocessing phase, any numbers found in the CommonVoice annotations were converted into their spoken form.
3. Re-aligned Selection: The most extensive selection, this dataset amounts to 1460 hours. It includes re-aligned SWC and LibriVox datasets acquired through CTC segmentation, alongside Tuda-DE and CommonVoice. Post CTC segmentation, the duration of SWC data reduced from 285 hours to 210 hours. This reduction was a result of filtering out poorly aligned or incorrect sentences. Bad or misaligned sentences were filtered out using the confidence score s_{seg} with a threshold of -1.5 in log space, selecting utterances with a minimum average production probability of at least 0.22 per second. Notably, since M-AILABS partially originates from LibriVox, this selection replaces M-AILABS with LibriVox. For the alignment process, the original LibriVox text was fetched on a chapter-by-chapter basis from Project Gutenberg-DE [75] and subsequently processed as outlined in Section 6.3.2. In the same manner as with the SWC corpus, LibriVox utterances are also filtered with a confidence score threshold of -1.5.

6.3.4 Training Results and Evaluation

Benchmarking across the different training set selections is done on end-to-end Transformer ASR networks of the ESPnet toolkit. All models were trained for 23 epochs and without data augmentation such as SpecAugment. The Transformer model has a 12-layer encoder and a 6-layer decoder, each with 2048 units per layer. Attention blocks feature 4 heads, with each having 256 units. Two multi-layer LSTM language models assist the

Table 6.5: WER on training sets composed of existing and re-aligned datasets. In a comparison of the TDNN-HMM models with the end-to-end Transformer models, the Transformer models achieve better performance but with more training data.

Datasets		ASR model	LM	Tuda-DE	
Training Datasets	h			dev	test
TD + SW [127]	412	TDNN-HMM	4-gram KN	15.3	16.5
TD + SW [127]	412	TDNN-HMM	LSTM (2×1024)	13.1	14.4
TD + SW + MA [127]	649	TDNN-HMM	4-gram KN	14.8	15.9
TD + SW + MA	649	Transformer	RNNLM (2×650)	16.4	17.2
TD + SW + MA + CV	986	Transformer	RNNLM (2×650)	16.0	17.1
TD + SW + MA + CV	986	Transformer	RNNLM (4×1024)	14.1	15.2
TD + SW* + LV* + CV	1460	Transformer	None	19.3	19.7
TD + SW* + LV* + CV	1460	Transformer	RNNLM (2×650)	14.3	14.9
TD + SW* + LV* + CV	1460	Transformer	RNNLM (4×1024)	**12.3**	**12.8**

ASR model at decoding: A two-layer LSTM with 650 units per layer and a perplexity of 8.53 and a four-layer LSTM with 1024 units per layer and a perplexity of 6.46.

The obtained results are compared to the results of Milde *et al.* [127], using a TDNN-HMM acoustic model in combination with a 4-gram language model with Kneser-Ney (KN [105]) smoothing or an LSTM language model.

Table 6.5 shows the ASR performance on the Tuda-DE benchmark of model trained on different training set selections. On the first dataset selection, the TDNN-HMM model trained decoded the Tuda-DE test set with a WER of 15.9, while the end-to-end Transformer model scored a higher WER of 17.2. The Transformer model surpassed the TDNN-HMM with a WER of 15.2 on the second selection that additionally included CommonVoice and used the larger language model for decoding. The largest selection with the re-aligned SWC and LibriVox further improved the WER to 12.8.

Using a language model with higher accuracy also significantly improves the WER. The was reduced from 19.7 to 14.9 with the small language model, and ultimately reduced to 12.8 with the large language model. Here, the language model improvement originates from better recognition of German words and grammar forms. Especially compounding poses a challenge for ASR systems [127]; recognizing two words instead of the compound word results in at least two word errors. For example, the falsely recognized compound *"Tunneleinfahrt"* is decoded as *"Tunnel_ein_fahrt"* that results in one substitution and two insertion errors. A comparable observation regarding end-to-end models was reported by Boyer *et al.* in their study on the French language [22].

To what extent does additional training data improve the performance of the end-to-end ASR system? As shown in Table 6.5, adding the CommonVoice dataset with 319h of audio to the first selection barely improved the performance, i.e., the WER decreased from 17.2 to 17.1 when decoding with the small language model. The third selection replaced the SWC and the M-ailabs corpora with the re-aligned SWC and LibriVox

datasets to a total of 1460h of audio. This step considerably improved the WER from 17.1 to 14.9, or, from 15.2 to 12.8 with the large language model, which can be attributed to the increased amount of training data and more accurate utterance alignments using CTC segmentation.

6.4 Construction of JTubeSpeech - a Large Japanese Corpus

This section describes the construction of a corpus from YouTube videos and subtitles for speech recognition using CTC segmentation. The method utilized for this construction is CTC segmentation, which not only aligns the transcriptions but also serves as a tool for evaluating the fit of the transcription to the audio.

The construction of JTubeSpeech [21†], the largest publicly available Japanese dataset, follows this approach. It is a non-English speech dataset that has been generated through crawling YouTube for audio-text pairs. The subtitles are aligned with the audio using a CTC-based ASR model and CTC segmentation, which calculates a confidence score to filter the audio-text pairs. This method does only require very little language-dependent pre-processing, thanks to the end-to-end ASR framework. JTubeSpeech consistently employs techniques that only require a pre-trained CTC-based ASR model, resulting in a large-scale Japanese ASR benchmark with over 1300 hours of data. While JTubeSpeech also provides a data collection for speaker verification, this section focuses on the aspects of speech recognition and data cleansing using CTC segmentation.

6.4.1 Text-Audio Mismatches of the Scraped Data

The subtitles to the videos provided on the Youtube platform are often not synchronous with the audio track or mismatch with the subtitle text.

These mismatches can take various forms, such as subtitles written in a language different from the audio, inaccuracies in transcriptions resulting from volunteer contributions or automatic speech recognition systems, and subtitles that do not correspond to the speech at all, such as speaking notes, commentary, or annotations. Some automatically generated subtitles are highly repetitive, such as annotations of the clock time. Additionally, there are many cases where the subtitles are correct, but the timestamps are misaligned.

As a result of these inconsistencies, the reliability of the subtitles and transcriptions obtained from the platform is limited, and thus, processing steps are necessary that realign the text to speech and filter out misaligned subtitles.

6.4.2 Related Work on Corpus Construction

Constructing a speech corpus is labor-intensive, encompassing data acquisition, audio segmentation, and defining training sets. Early work on this problem was done by Macherey *et al.* [121] who automated this process for news broadcasts using HMMs and a

discriminatively trained segmenter. This segmenter differentiates between noise, speech, and music, using Markov networks and Gaussian mixtures for probability modeling. By segmenting audio and ensuring sentence-wise cutting, they facilitate training corpus creation and transcription verification with minimal manual effort, even with only preliminary textual transcriptions.

Gigaspeech [32] is an English speech recognition dataset composed of 10,000 hours of audio-transcription pairs crawled from the internet, including YouTube videos with manually-generated subtitles. The team normalized the text by normalizing cases, removing special symbols, and converting numbers and dates/times to words. Audio and text alignment was executed using DNN/HMM ASR via the Kaldi toolkit and by the Smith–Waterman algorithm.

In a similar work, Elfeky *et al.* created an extensive corpus comprising 162,000 hours of speech sourced from YouTube videos and other sources [57]. They employed an HMM-based method combined with an LSTM acoustic model for data alignment.

In "The People's Speech" by Galvez *et al.* [62], the researchers used a two-step data filtering process with both hybrid DNN/HMM and end-to-end ASR systems. Out of the initial 52,500 hours of audio, 31,400 hours remained post-cleaning. They employed forced alignment for transcript correction and timestamping, and partitioned the hypothesis transcripts into 15-second chunks for DSAlign. This method faced challenges with text-audio mismatches and lengthy audio files (up to 13.4 hours). DSAlign outperformed traditional aligners but struggled with descriptive and translated text. Due to DSAlign's extensive runtime for unmatched segments, a 200-second timeout was set. If the character error rate (CER) between the ground truth and the aligned transcript exceeded 50%, the segment was excluded.

6.4.3 Comparison to Related Corpora

Table 6.6 shows a comparison with the existing corpora. From the approximately 10,000 hours of crawled subtitled audio data, roughly 1,300 hours of clean speech audio was selected using the confidence score threshold $\theta = -0.3$. The JTubeSpeech corpus has a similar size compared to the LaboroTVspeech (Japanese) and CommonVoice (English). The duration of the JTubeSpeech corpus does not include all collected data, but the subset that was selected by the data cleansing described in Section 6.4.8.

JTubeSpeech also includes a subset of utterances for speaker verification, e.g., selected utterances with only single speakers. However, this is not subject of this thesis; the dataset composition and evaluation for this task is described in the JTubeSpeech paper [21†].

6.4.4 Obtaining Speech Data

JTubeSpeech uses Youtube as source for speech data. The online video platform Youtube offers a vast and diverse collection of videos, including those with accompanying subtitles. The acquisition of audio and subtitles is facilitated through the utilization of web scraping techniques.

Table 6.6: A comparison of speech corpora of Japanese, English and Chinese based on their source and duration. While approximately 10,000 hours of audio was crawled from Youtube for the JTubeSpeech corpus, the selected clean utterances include 1,300 hours of speech data.

Language	**Corpus**	**Source**	**Duration[h]**
Japanese	JNAS [94]	In-house recording	90
Japanese	CSJ [122]	Conference talk	600
Japanese	LaboroTVspeech [7]	Broadcast TV	2,000
Japanese	Common Voice (ja) [8]	Web-based recording	2
Japanese	**JTubeSpeech**	YouTube	1,300
English	Librispeech [138]	Web-based recording	982
English	Common Voice (en) [8]	Web-based recording	1,100
English	SPGISpeech [133]	Earnings calls	5,000
English	GigaSpeech [32]	Web crawl	10,000
Chinese	Common Voice (cn) [8]	Web-based recording	12
Chinese	HKUST [118]	Telephone	200
Chinese	AISHELL-2 [54]	In-house recording	1,000

Data collection for JTubeSpeech took place between February and April of 2021. Out of the 110000 analyzed YouTube videos, 0.92% had manual subtitles and 41.7% had automatic subtitles. A total of approximately 10000 hours of speech data was obtained. Most videos were less than 5 minutes in length, with an average duration of 3.8 minutes; only a few videos were several hours long. In general, the utterances in the videos were shorter than 25 seconds due to the synchronization of the subtitles with the video; a few utterances spanned several hours due to their timestamps.

As described in Subsection 6.4.1, many subtitles are not in sync with the audio or do not match at all. To address this, they require re-alignment and correction of mismatches.

6.4.5 Data Cleansing

The audio data obtained came with pre-existing annotations in the form of subtitles and timings. To improve the accuracy of the utterance timings, CTC segmentation was applied to determine a score for the fit between the audio and subtitles. Utterances with low scores were filtered out. In addition, due to a significant number of inaccurate subtitle timings, a full re-alignment of the subtitles and audio was conducted.

The following steps were applied:

1. **Text pre-processing:** Minimal text pre-processing was performed to ensure that the ground truth text obtained from subtitles consisted of characters or tokens in the model dictionary. Numbers were replaced with their spoken equivalent using the "num2words" Python library [55]. Repetitive automated subtitles were detected and filtered out based on the average relative Levenshtein distance between

subtitles.

2. **Alignment:** CTC segmentation was applied, with the option to skip unrelated audio parts, as described in Subsection 6.2.2. The applied scoring length L = 30 corresponds to 0.96s of audio.

3. **Cleaning:** Samples with low quality were eliminated based on the CTC segmentation confidence score. For the final selection of utterances, the score threshold is set to $\theta = -3.0$; this can be interpreted as a production probability of the audio sequence given the text of at least 75% each second.

6.4.6 Alignment of Long Audio Files

The alignment process necessitates the inference of audio utilizing the encoder and the CTC layer of a pre-trained ASR model. However, the inference of the CTC model encountered a practical limitation. The memory complexity of a Transformer-based model displays a quadratic relationship with audio length, limiting a device with 64 GB of memory to perform inference on no more than 500 seconds of audio. Many audio files, however, exceed three hours in length. RNN-based models, with their relatively linear memory complexity for longer audio data, offer a potential solution for decoding longer files with reduced memory requirements. Yet, these models face limitations as well, with a maximum inference capability of 2.7 hours of audio before reaching a software-dependent memory limit imposed by the Pytorch toolkit.

To address the aforementioned limitation, long audio files are divided into smaller segments. The CTC activations are obtained from the inference of the ASR model on these segments. These CTC posteriors from the segments are then concatenated into a sequence that represents the CTC posteriors of the full audio file. This sequence is then subjected to CTC segmentation.

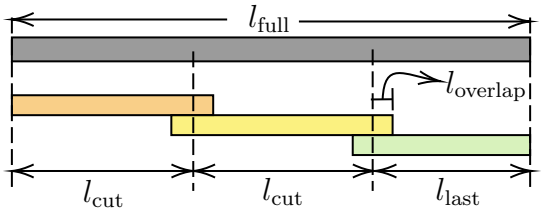


Figure 6.6: Partitioning of longer audio files to smaller parts.

As depicted in Figure 6.6, a long audio file of l_{full} samples is partitioned into smaller segments, each with length l_{cut}. The maximum length of the final segment l_{last} may exceed the default length l_{cut} by up to 25% to avoid excessively short segments. In the setup presented in this chapter, the largest audio segment comprised 320 seconds of audio.

Furthermore, to minimize distortions during inference due to abrupt cuts in the audio, an overlap with a duration of l_{overlap} is added to each side of the segment. Examination of the CTC posteriors revealed that an overlap between audio segments is necessary to

reduce distortions and maintain an acceptable level of impact on scoring. The minimum length of this overlap is dependent on the model, with the Transformer model used in this study requiring a minimum of 600 milliseconds. Based on this approximate value, an overlap of 1 second was chosen for the overlap duration. Before concatenating the inferred posteriors of the segments, the CTC posteriors corresponding to the overlaps are removed. To preserve timing information and maintain the correct form of the concatenated CTC posterior tensor, all lengths were chosen as multiples of the samples–to–posteriors ratio.

6.4.7 Distribution of Confidence Scores

Two pre-trained hybrid CTC/Attention models are employed for alignment, a Conformer model [72, 74] and an RNN model. Both models were trained on the LaboroTVspeech corpus [7, 188], a large Japanese corpus with 2000 hours of speech data. In general, the Transformer-based model has a better ASR performance than the RNN-based model on the LaboroTVspeech corpus.

CTC segmentation was applied to all videos, resulting in the calculation of an average score for each video. Videos with subtitles that did not match the audio had a very low average score. Based on this notion, a subset of 15000 videos with the highest average scores was selected and referred to as "top15k". This selection filtered out videos with poorly matching subtitles. In total, top15k comprises around 300GB in 16kHz-sampled monaural WAV format.

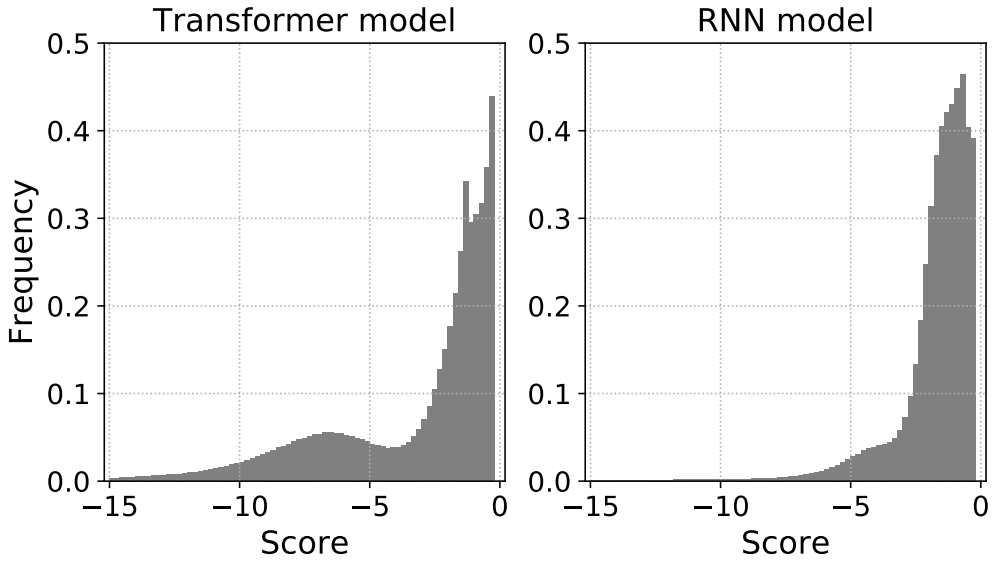


Figure 6.7: Histograms that visualize the distributions of CTC confidence scores for Transformer- and RNN-based ASR models over all utterances of the top15k subset.

Table 6.7 shows the distribution of CTC confidence scores for the Transformer- and for the RNN-based model. The Transformer model exhibited more spiky CTC activations which led to higher confidence scores for well-fitting utterances in general.

Similarly, mismatching utterances also had a lower score than with the RNN-based model. By inspection, better ASR performance seems to correlate with more confident CTC activations, resulting in higher scores for correct transcription-audio matches, and lower scores for mismatches.

6.4.8 Dataset Compositions

Several ASR models were trained using selected utterances, which were determined by filtering out utterances below a certain confidence score $s_{\text{seg}} \geq \theta$. These selections were evaluated on the hybrid CTC/attention Conformer model of the ESPnet toolkit [74]. The Conformer encoder has 12 Conformer blocks with a kernel size of 31, 512 attention dimensions, eight attention heads, and 2048 feed-forward dimensions. The model uses a Transformer decoder that has six blocks of eight attention heads, and 2048 feed-forward dimensions. SpecAugment is applied as data augmentation during training. The detailed configuration can be found in the ESPnet JTubeSpeech recipe [194].

Figure 6.8 gives an overview over the dataset composition. Several subsets were selected, starting with videos featuring only one speaker. The d-vector [187], or encoded speaker representation, was calculated for each recording. Recordings with multiple speakers were omitted, and only videos with single speakers were selected. In this step, videos with synthesized speech were also excluded due to their low d-vector variance. This was done to create a corpus with two purposes: speech recognition and speaker verification. The focus of this section is speech recognition; further details on the speaker verification selection can be found in the original publication [21†].

The single speaker video set was divided into three parts in a $90\% - 8\% - 2\%$ ratio, forming the training set, evaluation set, and development set[vii], respectively. Both the eval and the dev sets were filtered by their CTC segmentation confidence scores s_{seg}, each set by $\theta = -0.3$ and $\theta = -1.0$, resulting in an "easy" and "normal" variant for each of them; both were manually evaluated and verified to have accurate transcriptions. The single–speaker videos were filtered using different confidence score thresholds $\theta \in \{-0.3, -0.5, -1.0, -2.0, -3.0\}$, resulting in five selections of varying sizes, ranging from 12.7 hours up to 362.0 hours of speech. Table 6.7 lists the dataset compositions, their θ thresholds, number of videos, utterances and duration.

6.4.9 Training and Evaluation of Models with Cleaned and Re-aligned Data

Re-aligning existing utterance alignments can improve training data and the resulting ASR model, an approach proposed by Chen *et al.* [32]. The benefits of realigning using CTC segmentation were analyzed through a comparison of two models trained on

[vii] The evaluation set is synonymous to the "test" set, that is a portion of the data set that is set aside and used to evaluate the performance of the model after it has been trained. The "dev set" or "development set" refers to a portion of the data set that is used to evaluate its performance during model training, and in some cases tune the learning rate and other hyperparameters.

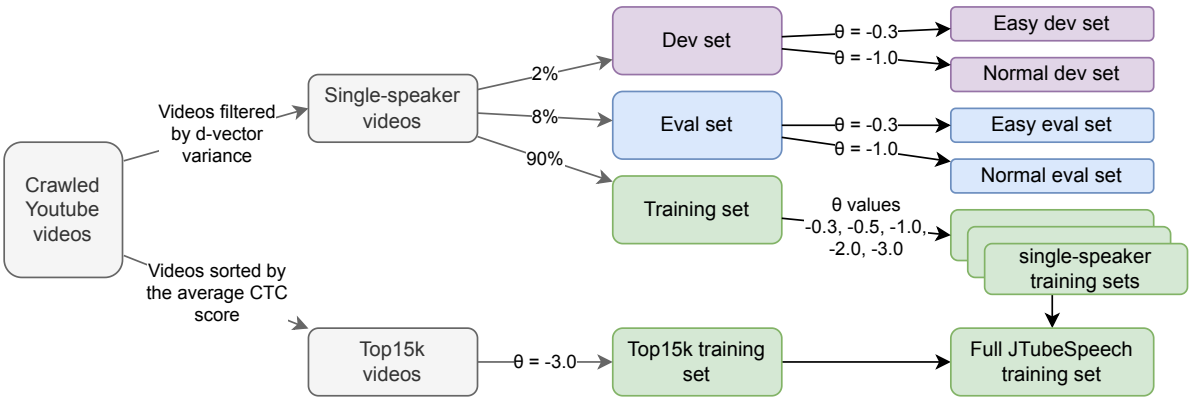


Figure 6.8: JTubeSpeech dataset overview.

Table 6.7: Various dataset selections of the JTubeSpeech corpus.

Data subset	θ	**Videos**	**Utterances**	**Duration[h]**
Easy dev set	-0.3	110	785	0.7
Easy eval set	-0.3	106	829	0.7
Normal dev set	-1.0	128	1,036	1.1
Normal eval set	-1.0	129	834	0.8
Single–speaker train set	-0.3	1,297	14,797	12.7
Single–speaker train set	-0.5	1,792	26,209	24.2
Single–speaker train set	-1.0	2,906	66,563	71.9
Single–speaker train set	-2.0	4,068	186,733	227.4
Single–speaker train set	-3.0	4,342	285,846	362.0
Top15k train set	-3.0	14,418	1,048,699	1087.1
Top15k + single–speaker train set	-3.0	17,761	1,270,124	1376.9

different text timings of the same dataset selection. In the first run, a Conformer model was trained on the single–speaker training set filtered with a score threshold of $\theta = -1.0$, with 71.9 hours of data. The performance of the ASR model trained only on cleaned data using original timings is described in Table 6.8.

In the second run, another Conformer model was trained using the same utterances as training data, but using the original YouTube timings; the data was filtered with a score threshold of $\theta = -1.0$, but only by calculation of a CTC score without full subtitle-audio alignment. This CTC score for each audio-text pair was calculated by cutting out the audio segment of each subtitle and deriving only its confidence score. Table 6.8 demonstrates a significant improvement in performance achieved through the use of CTC segmentation, effectively highlighting the benefits of realigning the data.

Table 6.8: A comparison of two Conformer models: One was trained with re-aligned and filtered data using CTC segmentation, the other model was trained with data that had the original utterance timings and was only filtered based on the utterance CTC scores. The model trained on the utterance timings obtained with CTC segmentation exhibits better performance.

WER	dev set	eval set		
	easy	normal	easy	normal
Original timings	11.5	16.5	9.2	15.7
CTC segmentation	**9.2**	**14.3**	**6.9**	**13.6**

7

Conclusion

Revisiting the introductory thesis, this dissertation recognized the historical importance of the hybrid DNN/HMM systems in the ASR domain. Yet, with the evolving inclination towards end-to-end neural networks, specific functionalities integral to DNN/HMM were missing in the emerging hybrid CTC/attention framework. This work identifies and bridges the gaps related to parameter exploration, susceptibility to adversarial noise in complex end-to-end models, continued dependence on pre-processing for feature extraction, and the utility of CTC outputs in speech alignment and automated dataset generation. The subsequent sections revisit these contributions to the hybrid CTC/attention framework.

7.1 Exploration of Hybrid CTC/Attention Speech Recognition

Chapter 3 examined the hybrid CTC/attention RNN architecture's parameter settings, especially multi-objective configurations. Gaussian process hyperparameter optimization was employed on the hybrid CTC/attention RNN network trained on the TED-LIUM 2 dataset. The two-stage experiment involved optimizing the end-to-end model's hyperparameters, focusing on the hybrid CTC/attention architecture, and subsequently refining the beam search parameters, including language model and CTC weights. Gaussian processes enable parameter optimization by capturing function uncertainty across all input points, fostering a balance between exploitation and exploration. This method outperforms random brute-force searches by considering this uncertainty and has been proven to excel in optimizing hyperparameters.

In total, 70 networks were trained and 590 beam search runs conducted. Analysis revealed distinct parameter groups, with larger networks tending to have better performance. The first group, characterized by high WER but relatively low CER, primarily consisted of results from CTC-only decoding without a language model, often exhibiting spelling errors or shifted word boundaries. The second group, with low WER but relatively high CER, was associated with attention-only decoding paired with an RNNLM language model, frequently displaying word loops and dropped utterance parts

but mostly accurate spelling. The best performance in the experiment was achieved using a hybrid CTC/attention decoding approach.

7.2 Adversarial Machine Learning

As models grow larger, they become more complex and also more prone to adversarial noise. Previous research explored adversarial machine learning in CTC-based and attention-based networks. Chapter 4 explores adversarial machine learning for the hybrid CTC/attention architecture.

Two methods for generating adversarial noise using attention were introduced: a static window and a sliding window. Examples generated with the sliding window exhibit higher error rates than compared with static window or CTC-based adversarial examples. The multi-objective construction of hybrid CTC/attention adversarial examples was demonstrated. Subsequent adversarial training of a hybrid CTC/attention ASR network enhanced its resilience against adversarial examples and showing improvement against white noise. Notably, the adversarial training of an RNN model also improved its regular speech recognition performance.

Adversarial features were converted to audio using feature inversion. Furthermore, the reconstructed adversarial audio was MP3 compressed. It was hypothesized that MP3 compression might mitigate adversarial noise effects by eliminating inaudible sections. Evidence supported this: MP3 compression enhanced performance on adversarial noise samples but worsened it with regular noise. Furthermore, MP3 compression reduced the SNR of adversarial noise on the test data.

7.3 Integrated Feature Extraction with Sinc Convolutions

Most end-to-end ASR systems rely on pre-processed features. Chapter 5 proposed an extension to the RNN-based hybrid CTC/attention ASR architecture, incorporating a front-end for direct raw audio classification, referred to as Lightweight Sinc Convolutions (LSC). This network architecture combines Sinc Convolutions for feature extraction with grouped depthwise separable convolutional layers and is geared towards maintaining a minimalistic parameter structure. Additional enhancements such as the incorporation of log-compression as an activation function and spectral augmentation for time-based data augmentation were also discussed. When prototyping this architecture as a keyword spotting system, it achieved competitive performance on the Google's Speech Commands dataset while significantly reducing parameter count to only 62k parameters.

With the collaboration of a sizable RNN language model, the LSC model delivers a word error rate of 10.7%, marking an absolute improvement of 1.9% over the best performing model utilizing the corresponding f-Bank architecture, while containing merely 21% of the model size of the latter. Given that the model processing raw audio is

smaller yet exhibits superior speech recognition performance, the improvement on the WER can be attributed to the Sinc filters and depthwise convolutions.

7.4 Text-to-audio Alignment using CTC Segmentation

End-to-end ASR systems often demand more training data compared to traditional DNN/HMM configurations, especially as they evolve in depth, complexity, and parameter count. Chapter 6 demonstrated the ability of a pre-trained hybrid CTC/attention model to leverage its CTC output for aligning text with corresponding audio.

This method, termed CTC segmentation, can be employed to automatically generate labeled speech datasets by extracting utterances along with their precise time-based alignments. Therefore, alignment does not require a separate HMM-based acoustic model, and any CTC-trained model is compatible with CTC segmentation. The alignment accuracy was assessed using the manually labeled TED-LIUM 2 dataset and compared with the alignments from conventional DNN/HMM systems. In a comparison, alignments from CTC segmentation are closer to the manually labeled ground truth than alignments generated with DTW- and HMM-based tools.

Furthermore, CTC segmentation provides a confidence metric indicating the congruence of each utterance to its associated audio segment. This metric is instrumental in filtering out inaccuracies, making it a valuable tool for automated dataset creation. Expanding on this, the latter half of the chapter describes the creation of two distinct speech datasets: one in German, derived from openly accessible audiobooks, and another in Japanese, sourced from Youtube videos. The latter was constructed using an almost language-agnostic approach requiring only minimal text preprocessing.

References

[1] M. Abdel-rahman. *Deep Neural Network acoustic models for ASR*. PhD thesis, University of Toronto, November 2014. → p. 11

[2] S. Abdoli, L. G. Hafemann, J. Rony, I. B. Ayed, P. Cardinal, and A. L. Koerich. Universal adversarial audio perturbations. *ArXiv*, `arXiv:1908.03173`, 2019. → p. 51

[3] L. Acerbi and W. Ji. Practical Bayesian Optimization for Model Fitting with Bayesian Adaptive Direct Search. In *Advances in Neural Information Processing Systems*, pp. 1836–1846, 2017. → p. 39

[4] M. Alzantot, B. Balaji, and M. Srivastava. Did you hear that? adversarial examples against automatic speech recognition. *arXiv preprint* `arXiv:1801.00554`, 2018. → p. 50

[5] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276. MIT Press, 1998. → p. 51

[6] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. N. Hannun, T. Han, L. V. Johannes, B. Jiang, C. Ju, B. Jun, P. Legresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu. Deep speech 2: End-to-end speech recognition in English and Mandarin. *33rd International Conference on Machine Learning, ICML 2016*, 1:312–321, 2016. → p. 73

[7] S. Ando and H. Fujihara. Construction of a Large-Scale Japanese ASR Corpus on TV Recordings. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6948–6952, 2021. → pp. 109, 111

[8] R. Ardila, M. Branson, K. Davis, M. Kohler, J. Meyer, M. Henretty, R. Morais, L. Saunders, F. Tyers, and G. Weber. Common Voice: A Massively-Multilingual Speech Corpus. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 4218–4222, Marseille, France, May 2020. European Language Resources Association. Commonvoice. → pp. 103, 104, 109

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint* *arXiv:1607.06450*, 2016. → p. 26

[10] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint* *arXiv:1409.0473*, 2014. → pp. 22, 31

[11] E. Bakhturina, V. Lavrukhin, B. Ginsburg, and Y. Zhang. Hi-fi multi-speaker english tts dataset. *arXiv preprint* *arXiv:2104.01497*, Apr. 2021. arXiv:2104.01497. → pp. 99, 100

[12] B. Barras. SoX: Sound eXchange. http://sox.sourceforge.net/, 2012. Flash informatique, No. 9, pp. 3–6. → p. 69

[13] T. Baumann, A. Köhn, and F. Hennig. The Spoken Wikipedia Corpus Collection, 2016. → pp. 103, 104

[14] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015. → p. 45

[15] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. → pp. 39, 40

[16] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011. → p. 39

[17] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. → p. 12

[18] M. Borsky, P. Mizera, P. Pollak, and J. Nouza. Dithering techniques in automatic recognition of speech corrupted by MP3 compression: Analysis, solutions and experiments. *Speech Communication*, 86(November):75–84, 2017. → p. 62

[19] M. Borsky, P. Pollak, and P. Mizera. Advanced acoustic modelling techniques in MP3 speech recognition. *Eurasip Journal on Audio, Speech, and Music Processing*, 2015(1). EURASIP Journal on Audio, Speech, and Music Processing, 2015. → p. 62

[20] L. E. Boucheron and P. L. De Leon. On the inversion of mel-frequency cepstral coefficients for speech enhancement applications. *ICSES'08 - ICSES 2008 International Conference on Signals and Electronic Systems, Proceedings*, pp. 485–488, 2008. → p. 58

[21] H. A. Bourlard and N. Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer Science & Business Media, 1994. → p. 18

[22] F. Boyer and J.-L. Rouas. End-to-End Speech Recognition: A review for the French Language. *preprint arXiv*, 2019. arXiv:1910.08502. → p. 106

[23] K. Brandenburg. MP3 and AAC Explained. *Audio Engineering Society, 17th International Conference 2004 October 26–29*, pp. 99–110, 1999. → p. 62

[24] K. Brandenburg, C. Faller, J. Herre, J. D. Johnston, and W. B. Kleijn. Perceptual coding of high-quality digital audio. *Proceedings of the IEEE*, 101(9):1905–1919, 2013. → p. 62

[25] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pp. 227–236. Springer, 1990. → pp. 13, 16

[26] A. E. Bryson Jr, W. F. Denham, and S. E. Dreyfus. Optimal programming problems with inequality constraints. *AIAA journal*, 1(11):2544–2550, 1963. → p. 17

[27] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. → p. 40

[28] N. Carlini. Tutorial on Adversarial Machine Learning with CleverHans, 2017. https://nicholas.carlini.com/slides/2017_odsc_advex.pdf. → p. 50

[29] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden Voice Commands. In *25th USENIX Security Symposium (USENIX Security 16)*, pp. 513–530. USENIX Association, 8 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/carlini. → pp. 49, 51

[30] N. Carlini and D. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 1–7. IEEE, 2018. → pp. 49, 51, 55

[31] W. Chan, N. Jaitly, Q. Le, and O. Vinyals. Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4960–4964. IEEE, IEEE, 2015. → pp. 20, 21, 31, 33, 45, 51

[32] G. Chen, S. Chai, G. Wang, J. Du, W.-Q. Zhang, C. Weng, D. Su, D. Povey, J. Trmal, J. Zhang, et al. GigaSpeech: An Evolving, Multi-domain ASR Corpus with 10,000 Hours of Transcribed Audio. *arXiv preprint arXiv:2106.06909*, 2021. → pp. 108, 109, 112

[33] G. Chen, C. Parada, and G. Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4087–4091, May 2014. → p. 73

[34] Y. Chen, J. Emer, and V. Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, June 2016. → p. 81

[35] M. Cheng, M. Taylor, R. Brito, R. Hegemann, A. Leidinger, et al. LAME (LAME Ain't an Mp3 Encoder). `https://lame.sourceforge.io/about.php`, Sept. 2020. Originally developed by Mike Cheng (1998-1999) and maintained by Mark Taylor until 2003. Subsequent development was managed by a core team with contributions from many individuals. → p. 63

[36] T.-R. Chiang and Y.-N. Chen. Relating Neural Text Degeneration to Exposure Bias. In *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 228–239, 2021. → p. 46

[37] C. C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani. State-of-the-Art Speech Recognition with Sequence-to-Sequence Models. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:4774–4778, 2018. → p. 26

[38] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint* *arXiv:1409.1259*, 2014. → p. 22

[39] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint* *arXiv:1406.1078*, 2014. → p. 22

[40] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha. Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices. In *Proc. Interspeech 2019*, pp. 3372–3376, 2019. → pp. 73, 77, 79, 87

[41] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, July 2017. → p. 72

[42] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint* *arXiv:1412.1602*, 2014. → p. 22

[43] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, volume 2015-Jan, pp. 577–585, 2015. → p. 23

[44] M. Cisse, Y. Adi, N. Neverova, and J. Keshet. Houdini: Fooling deep structured prediction models. *ArXiv*, `arXiv:1707.05373`, 2017. → p. 51

[45] L. D. Consortium et al. CSR-II (WSJ1) complete. *Linguistic Data Consortium, Philadelphia, vol. LDC94S13A*, 1994. → p. 95

[46] J. B. D. P. Kingma. Adam: A Method for Stochastic Optimization. In *Conference on Learning Representations (ICLR)*, 2015. → pp. 31, 79

[47] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint*, 2019. `arXiv:1901.02860`. → pp. 35, 37

[48] N. Das, M. Shanbhogue, S.-T. Chen, L. Chen, M. E. Kounavis, and D. H. Chau. Adagio: Interactive experimentation with adversarial attack and defense for audio. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 11053 LNAI, pp. 677–681. Springer, 2018. → p. 52

[49] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366. IEEE, 1980. → p. 11

[50] F. De Saussure. *Cours de linguistique générale*, volume 1. Otto Harrassowitz Verlag, 1916. → p. 1

[51] P. Denisov and N. T. Vu. IMS-speech: A speech to text tool. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2019*, pp. 170–177. TUDpress, Dresden, 2019. → p. 103

[52] L. Dong, S. Xu, and B. Xu. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5884–5888, 2018. → p. 25

[53] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9185–9193, 2018. → pp. 54, 63

[54] J. Du, X. Na, X. Liu, and H. Bu. AISHELL-2: Transforming Mandarin ASR Research Into Industrial Scale. *ArXiv*, Aug. 2018. → p. 109

[55] V. Dupras, M. Grigaitis, and T. Ogawa. Num2Words: Modules to convert numbers to words., June 2021. `https://github.com/savoirfairelinux/num2words`. → p. 109

[56] M. Elbayad, L. Besacier, and J. Verbeek. Pervasive attention: 2D Convolutional neural networks for sequence-to-sequence prediction. *CoNLL 2018 - 22nd Conference on Computational Natural Language Learning, Proceedings*, pp. 97–107, 2018. → p. 22

[57] M. Elfeky, P. Haghani, T. Strohman, and M. Bacchiani. TOWARD DOMAIN-INVARIANT SPEECH RECOGNITION VIA LARGE SCALE TRAINING Arun Narayanan , Ananya Misra , Khe Chai Sim , Golan Pundak , Anshuman Tripathi ,. *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 441–447. IEEE, 2018. → p. 108

[58] J. Fainberg, O. Klejch, E. Loweimi, P. Bell, and S. Renals. Acoustic Model Adaptation from Raw Waveforms with SincNet. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019. → p. 71

[59] S. Fernández, A. Graves, and J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *International Conference on Artificial Neural Networks (ICANN)*, pp. 220–229. Springer, 2007. → pp. 73, 99

[60] P. Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38. McPherson, KS: R & D Publications, c1987-1994., 1994. → p. 29

[61] C. Gaida, P. Lange, R. Petrick, P. Proba, A. Malatawy, and D. Suendermann-Oeft. Comparing open-source speech recognition toolkits. In *11th International Workshop on Natural Language Processing and Cognitive Science*, 2014. → p. 18

[62] D. Galvez, G. Diamos, J. M. C. Torres, J. F. Cerón, K. Achorn, A. Gopi, D. Kanter, M. Lam, M. Mazumder, and V. J. Reddi. The People's Speech: A Large-Scale Diverse English Speech Recognition Dataset for Commercial Usage. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. → p. 108

[63] P. Ghahremani, B. Babaali, D. Povey, K. Riedhammer, J. Trmal, and S. Khudanpur. A pitch extraction algorithm tuned for automatic speech recognition. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 2494–2498. IEEE, 2014. → p. 32

[64] P. Golik, Z. Tüske, R. Schlüter, and H. Ney. Convolutional neural networks for acoustic modeling of raw time signal in LVCSR. In *Sixteenth annual conference of the international speech communication association*, 2015. → p. 72

[65] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016. → p. 12

[66] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, `arXiv:1412.6572`:1–11, 2014. → pp. 50, 52, 60

[67] A. Graves. Sequence Transduction with Recurrent Neural Networks. *arXiv preprint* ***arXiv:1211.3711***, 2012. → pp. 21, 31

[68] A. Graves. *Supervised sequence labelling*. Springer, 2012. → pp. 27, 31, 99

[69] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint* ***arXiv:1308.0850***, 2013. → pp. 14, 22

[70] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 369–376. ACM, ACM, 2006. → pp. 21, 27

[71] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pp. 1764–1772. PMLR, 2014. → pp. 20, 21

[72] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Proc. Interspeech 2020*, pp. 5036–5040, 2020. → pp. 37, 111

[73] C. Gulcehre, O. Firat, K. Xu, K. Cho, L. Barrault, H.-C. Lin, F. Bougares, H. Schwenk, and Y. Bengio. On Using Monolingual Corpora in Neural Machine Translation. *arXiv preprint* `arXiv:1503.03535`, 2015. → pp. 30, 32

[74] P. Guo, F. Boyer, X. Chang, T. Hayashi, Y. Higuchi, H. Inaguma, N. Kamo, C. Li, D. Garcia-Romero, J. Shi, et al. Recent developments on espnet toolkit boosted by conformer. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5874–5878. IEEE, 2021. → pp. 37, 111, 112

[75] n. Gutenberg. Projekt Gutenberg-DE, 2019. `https://gutenberg.spiegel.de`. → pp. 103, 105

[76] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep Speech: Scaling up end-to-end speech recognition. *arXiv preprint* `arXiv:1412.5567`, pp. 1–12, 2014. → pp. 11, 51, 73

[77] A. Hannun, V. Pratap, J. Kahn, and W.-N. Hsu. Differentiable Weighted Finite-State Transducers. *arXiv preprint* `arXiv:2010.01003`, 2020. → p. 28

[78] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint* `arXiv:1408.2873`, 2014. → p. 31

[79] P. Hennig and C. J. Schuler. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012. → p. 39

[80] H. Hermansky. Perceptual linear predictive (PLP) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752. Acoustical Society of America, 1990. → p. 11

[81] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint* `arXiv:1207.0580`, 2012. → p. 17

[82] M. Hira. Forced Alignment with Wav2Vec2 - Torchaudio 2.1.0 documentation. `https://pytorch.org/audio/stable/tutorials/forced_alignment_tutorial.html`, 2023. Accessed: 2023-11-12. → p. 96

[83] J. L. Hodges. The significance probability of the Smirnov two-sample test. *Arkiv för Matematik*, 3(5):469–486. Kluwer Academic Publishers, 1958. → p. 69

[84] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67. Taylor & Francis, 1970. → p. 17

[85] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2019. arXiv preprint `arXiv:1904.09751`. → p. 46

[86] M. Honnibal and I. Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017. → p. 104

[87] T. Hori, J. Cho, and S. Watanabe. End-to-end speech recognition with word-based RNN language models. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 389–396. IEEE, 2018. → p. 31

[88] 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:1704.04861*, 2017. → p. 72

[89] S. Hu, X. Shang, Z. Qin, M. Li, Q. Wang, and C. Wang. Adversarial Examples for Automatic Speech Recognition: Attacks and Countermeasures. *IEEE Communications Magazine*, 57(10):120–126. IEEE, 2019. → pp. 51, 52

[90] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015. → p. 77

[91] K. Irie, A. Zeyer, R. Schlüter, and H. Ney. Language Modeling with Deep Transformers. *Proc. Interspeech 2019*, pp. 3905–3909, 2019. → p. 20

[92] D. Iter, J. Huang, and M. Jermann. Generating adversarial examples for speech recognition. *Stanford Technical Report*, 2017. → p. 52

[93] K. Ito and L. Johnson. The LJ Speech Dataset. `https://keithito.com/LJ-Speech-Dataset/`, 2017. → p. 55

[94] K. Itou, M. Yamamoto, K.Takeda, T. Takezawa, T. Matsuoka, T. Kobayashi, K. Shikano, and S. Itahashi. JNAS: Japanese speech corpus for large vocabulary continuous speech recognition research. *Journal of the Acoustical Society of Japan (E)*, 20(3):199–206, 4 1999. → p. 109

[95] L. Kaiser, A. N. Gomez, and F. Chollet. Depthwise Separable Convolutions for Neural Machine Translation. In *International Conference on Learning Representations (ICLR)*, 2018. → pp. 72, 75, 76

[96] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1700–1709, 2013. → p. 22

[97] T. Kamp, T. Björnsson, and R. Hileman. DSalign: DeepSpeech based forced alignment tool. `https://github.com/mozilla/DSAlign`, 2020. → p. 92

[98] N. Kanda, X. Lu, and H. Kawai. Maximum a posteriori Based Decoding for CTC Acoustic Models. In *Interspeech*, pp. 1868–1872, 2016. → p. 28

[99] S. Karita. Comment on Espnet Issue Nr.1246 - Improving LM training (custom optimizer, custom scheduler, Transformer LM, etc). Personal correspondence, Oct. 2019. → p. 46

[100] S. Karita. Personal correspondence - On Repetitive Loops in ASR Transformers and RNN Networks, 11 2021. Personal correspondence. → p. 46

[101] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang. A Comparative Study on Transformer vs RNN in Speech Applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, volume 9, pp. 449–456. IEEE, 2019. → p. 30

[102] S. Karita, N. E. Y. Soplin, S. Watanabe, M. Delcroix, A. Ogawa, and T. Nakatani. Improving Transformer-Based End-to-End Speech Recognition with Connectionist Temporal Classification and Language Model Integration. In *Proc. Interspeech 2019*, pp. 1408–1412, 2019. → pp. 25, 35, 85, 95

[103] D. P. P. Z. S. Khudanpur. Speech recognition with next-generation Kaldi (k2, Lhotse, Icefall). *Interspeech: tutorials*, 2021. → p. 28

[104] S. Kim, T. Hori, and S. Watanabe. Joint CTC-Attention based End-to-End Speech Recognition using Multi-task Learning. *CoRR*, 2016. → p. 30

[105] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *1995 international conference on acoustics, speech, and signal processing*, volume 1, pp. 181–184. IEEE, 1995. → p. 106

[106] T. Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *ArXiv*, `arXiv:1804.10959`, 2018. DOI:10.18653/v1/P18-1007. → p. 29

[107] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint `arXiv:1808.06226`*, 2018. → p. 29

[108] K. Kumatani, S. Panchapagesan, M. Wu, M. Kim, N. Strom, G. Tiwari, and A. Mandai. Direct modeling of raw audio with DNNS for wake word detection. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 252–257, Dec. 2017. → p. 73

[109] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *CoRR*, `arXiv:1607.02533`, 2016. → p. 50

[110] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *CoRR*, `arXiv:1611.01236`, 2016. → p. 63

[111] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, J. Wang, Z. Zhang, Z. Ren, A. Yuille, S. Huang, Y. Zhao, Y. Zhao, Z. Han, J. Long, Y. Berdibekov, T. Akiba, S. Tokui, and M. Abe. Adversarial Attacks and Defences Competition. In *The NIPS'17 Competition: Building Intelligent Systems*, pp. 195–231. Springer, 2018. → pp. 50, 59

[112] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, 1(c):1–14, 2019. → p. 50

[113] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf. The CMU SPHINX-4 speech recognition system. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*, volume 1, pp. 2–5, 2003. → p. 105

[114] S. Latif, R. Rana, S. Khalifa, R. Jurdak, J. Qadir, and B. W. Schuller. Deep Representation Learning in Speech Processing: Challenges, Recent Advances, and Future Trends. In *arXiv preprint* `arXiv:2001.00378`, 2020. → pp. 71, 73

[115] J. Lee, T. Kim, J. Park, and J. Nam. Raw Waveform-based Audio Classification Using Sample-level CNN Architectures. In *NIPS 2017 Machine Learning for Audio Signal Processing Workshop*, 2017. → p. 73

[116] V. I. Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710, 1966. → p. 10

[117] n. Librivox. LibriVox: Free Public Domain Audiobooks, 2020. → pp. 55, 103, 104

[118] Y. Liu, P. Fung, Y. Yang, C. Cieri, S. Huang, and D. Graff. HKUST/MTS: A very large scale Mandarin telephone speech corpus. In *International Symposium on Chinese Spoken Language Processing*, pp. 724–735. Springer, 2006. → p. 109

[119] L. Lu, L. Kong, C. Dyer, and N. A. Smith. Multitask Learning with CTC and Segmental CRF for Speech Recognition. In *Interspeech*, 2017. → pp. 30, 54

[120] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang, and T. Liu. Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View. *CoRR*, 2019. `arXiv:1906.02762`. → p. 37

[121] W. Macherey and H. Ney. Towards automatic corpus preparation for a German broadcast news transcription system. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pp. I–733. IEEE, 2002. → p. 107

[122] K. Maekawa, H. Koiso, S. Furui, and H. Isahara. Spontaneous Speech Corpus of Japanese. In *Proc. LREC*, pp. 947–952, 2000. → p. 109

[123] F. J. Massey. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78. American Statistical Association, Taylor & Francis, Ltd., 1951. → p. 68

[124] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015. → p. 58

[125] Y. Miao, M. Gowayyed, X. Na, T. Ko, F. Metze, and A. Waibel. An empirical exploration of CTC acoustic models. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2623–2627. IEEE, 2016. → p. 32

[126] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, pp. 1045–1048. Makuhari, 2010. → p. 20

[127] B. Milde and A. Köhn. Open Source Automatic Speech Recognition for German. In *Speech Communication; 13th ITG-Symposium*, pp. 1–5. VDE, 2018. → pp. 103, 105, 106

[128] A.-r. Mohamed, G. E. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE transactions on audio, speech, and language processing*, 20(1):14–22. IEEE, 2011. → p. 11

[129] K. Naoyuki. VoxForge ASR results, 2020. kamo-naoyuki/espnet 186bddf. → p. 59

[130] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88. ACM New York, NY, USA, 2001. → p. 10

[131] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, and F. Koushanfar. Universal adversarial perturbations for speech recognition systems. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2019-Sep:481–485, 2019. DOI: 10.21437/interspeech.2019-1353. → p. 51

[132] J. Nouza, P. Cerva, and J. Silovsky. Adding controlled amount of noise to improve recognition of compressed and spectrally distorted speech. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 8046–8050. IEEE, 2013. → p. 62

[133] P. K. O'Neill, V. Lavrukhin, S. Majumdar, V. Noroozi, Y. Zhang, O. Kuchaiev, J. Balam, Y. Dovzhenko, K. Freyberg, M. D. Shulman, B. Ginsburg, S. Watanabe, and G. Kucsko. SPGISpeech: 5,000 hours of transcribed financial audio for fully formatted end-to-end speech recognition, 2021. `arXiv:2104.02014`. → p. 109

[134] A. V. Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999. → p. 75

[135] D. Palaz, R. Collobert, and M. M. Doss. Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. *arXiv preprint* `arXiv:1304.1018`, 2013. → p. 72

[136] D. Palaz, R. Collobert, et al. Analysis of CNN-based speech recognition system using raw speech as input. Technical report, Idiap, 2015.

[137] D. Palaz, M. Magimai-Doss, and R. Collobert. End-to-end acoustic modeling using convolutional neural networks for HMM-based automatic speech recognition. *Speech Communication*, 108:15–32. Elsevier, 2019. → p. 72

[138] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210. IEEE, 2015. → pp. 46, 109

[139] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597. IEEE, 2016. → p. 52

[140] T. Parcollet, M. Morchid, and G. Linarès. E2E-SINCNET: Toward Fully End-To-End Speech Recognition. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7714–7718. IEEE, 2020. → pp. 71, 73

[141] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. *Interspeech 2019*. ISCA, Sept. 2019. → p. 31

[142] 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. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017. → p. 28

[143] N. Perraudin, P. Balazs, and P. L. Søndergaard. A fast Griffin-Lim algorithm. In *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 1–4. IEEE, 2013. → p. 58

[144] A. Pettarin. aeneas, 2017. `https://www.readbeyond.it/aeneas/`. → p. 92

[145] P. Pollak and M. Behunek. ACCURACY OF MP3 SPEECH RECOGNITION UNDER REAL-WORD CONDITIONS - Experimental Study. In *Proceedings of the International Conference on Signal Processing and Multimedia Applications*, pp. 5–10. SciTePress - Science and and Technology Publications, 2011. → p. 62

[146] P. Pollak and M. Borsky. Small and large vocabulary speech recognition of MP3 data under real-word conditions: Experimental study. *Communications in Computer and Information Science*, 314:409–419, 2012. → p. 62

[147] M. Popel and O. Bojar. Training tips for the transformer model. *arXiv preprint arXiv:1804.00247*, 2018. → p. 31

[148] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, Hilton Waikoloa Village, Big Island, Hawaii US, Dec. 2011. IEEE Signal Processing Society, IEEE Signal Processing Society. IEEE Catalog No.: CFP11SRW-USB. → pp. 18, 84, 92

[149] D. Povey, X. Zhang, and S. Khudanpur. Parallel training of deep neural networks with natural gradient and parameter averaging. *arXiv preprint* `arXiv:1410.7455`, p. 124. Citeseer, 2014. → p. 51

[150] Y. Qin, N. Carlini, I. Goodfellow, G. Cottrell, and C. Raffel. Imperceptible, Robust, and targeted adversarial examples for automatic speech recognition. *36th International Conference on Machine Learning, ICML 2019*, 2019-Jun:9141–9150, 2019. → p. 51

[151] L. R. Rabiner. *Digital processing of speech signals*. Pearson Education India, 1978. → p. 11

[152] S. Radeck-Arneth, B. Milde, A. Lange, E. Gouvêa, S. Radomski, M. Mühlhäuser, and C. Biemann. Open Source German Distant Speech Recognition: Corpus and Acoustic Model. In P. Král and V. Matoušek, editors, *International Conference on Text, Speech, and Dialogue*, volume 9302 of *Lecture Notes in Computer Science*, pp. 480–488. Springer, Springer International Publishing, 2015. → pp. 103, 104, 105

[153] K. Rajaratnam, B. Alshemali, and J. Kalita. Speech Coding and Audio Preprocessing for Mitigating and Detecting Audio Adversarial Examples on Automatic Speech Recognition. In *UCCS Conference Paper*, 07 2018. → p. 52

[154] C. E. Rasmussen. Gaussian Processes in Machine Learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003. → p. 40

[155] M. Ravanelli and Y. Bengio. Interpretable Convolutional Filters with SincNet. In *NIPS 2018 Interpretability and Robustness for Audio, Speech and Language Workshop*, 2018. → pp. 71, 72

[156] M. Ravanelli and Y. Bengio. Speaker Recognition from Raw Waveform with SincNet. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 1021–1028, Dec. 2018. → pp. 72, 74

[157] M. Ravanelli and Y. Bengio. Speech and Speaker Recognition from Raw Waveform with SincNet. In *arXiv preprint*, 2018. → pp. 71, 72

[158] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. D. Mori, and Y. Bengio. SpeechBrain: A General-Purpose Speech Toolkit. `arXiv:2106.04624`, June 2021. → p. 100

[159] A. Rousseau, P. Deléglise, and Y. Esteve. Enhancing the TED-LIUM Corpus with Selected Data for Language Modeling and More TED Talks. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014)*, pp. 3935–3939, Mar. 2014. → pp. 63, 84

[160] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536. Nature Publishing Group UK London, 1986. → p. 17

[161] J. Schenk and G. Rigoll. *Mensch-Maschine-Kommunikation: Grundlagen von sprach-und bildbasierten Benutzerschnittstellen*. Springer-Verlag, 2010. → p. 9

[162] F. Schiel. Automatic Phonetic Transcription of Non-Prompted Speech. In *Proc. of the ICPhS*, pp. 607–610, San Francisco, 8 1999. → p. 92

[163] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117. Elsevier, 2015. → p. 17

[164] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. *arXiv preprint*, `arXiv:1808.05665`, 2018. DOI:10.14722/ndss.2019.23288. → pp. 51, 55, 62

[165] L. Schönherr, S. Zeiler, T. Holz, and D. Kolossa. Imperio: Robust Over-the-Air Adversarial Examples Against Automatic Speech Recognition Systems. *arXiv preprint* ***arXiv:1908.01551***, 2019. → p. 51

[166] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681. Ieee, 1997. → p. 15

[167] M. Shahnawaz, E. Plebani, I. Guaneri, D. Pau, and M. Marcon. Studying the Effects of Feature Extraction Settings on the Accuracy and Memory Requirements of Neural Networks for Keyword Spotting. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pp. 1–6, Sept. 2018. → pp. 26, 73

[168] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pp. 1528–1540, 2016. → p. 51

[169] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4779–4783. IEEE, 2018. → p. 55

[170] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint* ***arXiv:1409.1556***, 2014. → p. 32

[171] P. Sirum and I. Sanches. The Influence of Audio Compression on Speech Recognition Systems. In *SPECOM' 2004: 9th Conference Speech and Computer St. Petersburg, Russia*, 2004. → p. 62

[172] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012. → pp. 39, 40

[173] I. Solak. The M-AILABS speech dataset, 2019. → pp. 103, 104

[174] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958. JMLR. org, 2014. → p. 17

[175] P. Sun, Y. Jiang, E. Xie, W. Shao, Z. Yuan, C. Wang, and P. Luo. What makes for end-to-end object detection? In *International Conference on Machine Learning*, pp. 9934–9944. PMLR, 2021. → p. 21

[176] S. Sun, P. Guo, L. Xie, and M. Y. Hwang. Adversarial regularization for attention based end-to-end robust speech recognition. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 27(11):1826–1838. IEEE, 2019. → pp. 51, 53, 54, 59

[177] S. Sun, C.-F. Yeh, M. Ostendorf, M.-Y. Hwang, and L. Xie. Training augmentation with adversarial examples for robust speech recognition. *arXiv preprint* *arXiv:1806.02782*, 2018. → pp. 52, 56

[178] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4(January):3104–3112, 2014. → pp. 21, 22

[179] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec. 2017. → p. 79

[180] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, `arXiv:1312.6199`, 2013. → pp. 49, 50

[181] R. Tang and J. Lin. Deep Residual Learning for Small-Footprint Keyword Spotting. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5484–5488, Apr. 2018. → pp. 73, 87

[182] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using Convolutional Networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 648–656, June 2015. → p. 77

[183] S. Toshniwal, A. Kannan, C.-C. Chiu, Y. Wu, T. N. Sainath, and K. Livescu. A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pp. 369–375. IEEE, 2018. → p. 45

[184] Z. Tüske, P. Golik, R. Schlüter, and H. Ney. Acoustic modeling with deep neural networks using raw time signal for LVCSR. In *Fifteenth annual conference of the international speech communication association*. Citeseer, 2014. → p. 72

[185] J. Vadillo and R. Santana. Universal adversarial examples in speech command classification. *ArXiv*, `arXiv:1911.10182`, 2019. → p. 51

[186] J. Vadillo and R. Santana. On the human evaluation of audio adversarial examples. *arXiv preprint* *arXiv:2001.08444*, Jan. 2020. → p. 52

[187] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4052–4056. IEEE, 2014. → p. 112

[188] Various authors. Espnet /Egs2/Laborotv/Asr1, Aug 2021. `https://github.com/espnet/espnet/tree/master/egs2/laborotv/asr1`. → p. 111

[189] 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*, pp. 5999–6009, 2017. → pp. 25, 26, 35

[190] VoxForge.org. VoxForge Speech Corpus. `https://www.voxforge.org/home`, Dec. 2016. → pp. 59, 63

[191] J. Wang, R. Jia, G. Friedland, B. Li, and C. Spanos. One Bit Matters: Understanding Adversarial Examples as the Abuse of Redundancy. *arXiv preprint* *arXiv:1810.09650*, 2018. → p. 50

[192] Y. Wang, V. Peddinti, H. Xu, X. Zhang, D. Povey, and S. Khudanpur. Backstitch: Counteracting Finite-Sample Bias via Negative Steps. In *Interspeech*, pp. 1631–1635, 2017. → p. 51

[193] P. Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv preprint* *arXiv:1804.03209*, 2018. → pp. 78, 79, 87

[194] S. Watanabe. Recipe of Jtubespeech By sw005320 · pull Request Nr. 3311 · espnet/espnet, June 2021. `https://github.com/espnet/espnet/pull/3311`. → p. 112

[195] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, and N. Chen. ESPnet: End-to-End Speech Processing Toolkit. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 2207–2211, 2018. DOI: 10.21437/Interspeech.2018-1456. → pp. 30, 33, 42, 44, 60, 85, 95, 100

[196] S. Watanabe, T. Hori, S. Kim, J. R. Hershey, and T. Hayashi. Hybrid CTC/Attention Architecture for End-to-End Speech Recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1240–1253. IEEE, 12 2017. → pp. 2, 20, 21, 28, 30, 31, 32, 46, 99

[197] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural computation*, 1(2):270–280, 1989. → p. 45

[198] S. Wiseman and A. M. Rush. Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1296–1306, 2016. → p. 30

[199] M. Won, S. Chun, O. Nieto, and X. Serrc. Data-Driven Harmonic Filters for Audio Representation Learning. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 536–540, 2020. → pp. 71, 73

[200] F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, and M. Auli. Pay Less Attention with Lightweight and Dynamic Convolutions. *arXiv preprint* `arXiv:1901.10430`, 2019. → pp. 72, 76

[201] Y. Wu. *Harnessing Adversarial Samples Against Voice Assistants*. PhD thesis, Rutgers The State University of New Jersey, School of Graduate Studies, 2019. → p. 49

[202] H. Xiao. *Adversarial and secure machine learning*. PhD thesis, Technische Universität München, 2017. → p. 49

[203] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020. → p. 27

[204] J. Xu, X. Liu, J. Yan, D. Cai, H. Li, and J. Li. Learning to Break the Loop: Analyzing and Mitigating Repetitions for Neural Text Generation. In *Advances in Neural Information Processing Systems*, 2021. → p. 46

[205] Y.-Y. Yang, M. Hira, Z. Ni, A. Astafurov, C. Chen, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang, J. Lian, J. Hwang, J. Chen, P. Goldsborough, S. Narenthiran, S. Watanabe, S. Chintala, and V. Quenneville-Bélair. Torchaudio: Building Blocks for Audio and Speech Processing. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6982–6986, 2022. → p. 96

[206] Z. Yang, P. Y. Chen, B. Li, and D. Song. Characterizing audio adversarial examples using temporal dependency. *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–15, 2019. → p. 51

[207] Z. Yang, B. Li, P.-Y. Chen, and D. Song. Towards Mitigating Audio Adversarial Perturbations. In *Workshop track - ICLR 2018*, pp. 1–7, 2018. → p. 52

[208] S. F. Y. Yin, D. Mori, et al. ReazonSpeech: A Free and Massive Corpus for Japanese ASR. In *Annual meetings of the Association for Natural Language Processing*, 2023. → p. 99

[209] S. J. Young and S. Young. *The HTK hidden Markov model toolkit: Design and philosophy*. University of Cambridge, Department of Engineering Cambridge, England, 1993. → p. 92

[210] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. F. Wang, and C. A. Gunter. CommanderSong: A systematic approach for practical adversarial voice recognition. *Proceedings of the 27th USENIX Security Symposium*, pp. 49–64, 2018. → p. 51

[211] N. Zeghidour, N. Usunier, I. Kokkinos, T. Schaiz, G. Synnaeve, and E. Dupoux. Learning Filterbanks from Raw Speech for Phone Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5509–5513, Apr. 2018. → pp. 73, 75, 77

[212] N. Zeghidour, N. Usunier, G. Synnaeve, R. Collobert, and E. Dupoux. End-to-End Speech Recognition from the Raw Waveform. In *Proc. Interspeech 2018*, pp. 781–785, 2018. → pp. 73, 75, 77

[213] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint* `arXiv:1212.5701`, 2012. → p. 31

[214] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. DolphinAttack: Inaudible voice commands. *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 103–117, 2017. → p. 52

[215] J. Zhang, B. Zhang, and B. Zhang. Defending adversarial attacks on cloud-aided automatic speech recognition systems. *SCC 2019 - Proceedings of the 7th International Workshop on Security in Cloud Computing, co-located with AsiaCCS 2019*, pp. 23–31, 2019. → p. 62

[216] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6848–6856, June 2018. → p. 73

[217] Y. Zhang, N. Suda, L. Lai, and V. Chandra. Hello Edge: Keyword Spotting on Microcontrollers. `arXiv:1711.07128`, 2017. → pp. 72, 73, 77, 79, 87

[218] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and models*, volume 22. Springer Science & Business Media, 2013. → pp. 9, 62

[219] E. Zwicker and E. Terhardt. Analytical expressions for critical-band rate and critical bandwidth as a function of frequency. *The Journal of the Acoustical Society of America*, 68(5):1523–1525. Acoustical Society of America, 1980. → p. 86

Publications

[1†] I. Andronic, L. Kürzinger, E. R. C. Rosas, G. Rigoll, and B. U. Seeber. MP3 Compression to Diminish Adversarial Noise in End-to-End Speech Recognition. In *Speech and Computer*, pp. 22–34. Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-60276-5_3. arXiv:2007.12892. → pp. 49, 51, 52

[2†] K. Garb, M. Xhemrishi, L. Kürzinger, and C. Frisch. The Wiretap Channel for Capacitive PUF-Based Security Enclosures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022. arXiv:2202.01508.

[3†] M. Hiller, L. Kürzinger, and G. Sigl. Review of error correction for PUFs and evaluation on state-of-the-art FPGAs. *Journal of Cryptographic Engineering*, May 2020.

[4†] M. Hiller, L. Kürzinger, G. Sigl, S. Müelich, S. Puchinger, and M. Bossert. Low-area Reed decoding in a generalized concatenated code construction for PUFs. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pp. 143–148. IEEE, 2015.

[5†] V. Immler, M. Hennig, L. Kürzinger, and G. Sigl. Practical aspects of quantization and tamper-sensitivity for physically obfuscated keys. In *Workshop on cryptography and security in computing systems (CS2)*, pp. 13–18, 2016.

[6†] L. Kürzinger, N. Lindae, P. Klewitz, and G. Rigoll. Lightweight End-to-End Speech Recognition from Raw Audio Data Using Sinc-Convolutions. *Proc. Interspeech 2020*, 2(10):1659–1663, Oct. 2020. https://doi.org/10.21437/Interspeech.2020-1392. arXiv:2010.07597. → pp. 71, 82

[7†] L. Kürzinger, E. R. C. Rosas, L. Li, T. Watzel, and G. Rigoll. Audio Adversarial Examples for Robust Hybrid CTC/Attention Speech Recognition. In *International Conference on Speech and Computer*, pp. 255–266. Springer International Publishing, Oct. 2020. https://doi.org/10.1007/978-3-030-60276-5_26. arXiv:2007.10723. → p. 49

[8†] L. Kürzinger, T. Watzel, L. Li, R. Baumgartner, and G. Rigoll. Exploring Hybrid CTC/Attention End-to-End Speech Recognition with Gaussian Processes. In *International Conference on Speech and Computer*, volume 11658 LNAI, pp. 258–269. Springer International Publishing, Oct. 2019. https://doi.org/10.1007/978-3-030-26061-3_27. → p. 39

[9†] L. Kürzinger and D. Winkelbauer. CTC-Segmentation: Segment an audio file and obtain utterance alignments. (Python package), June 2021. https://github.com/lumaku/ctc-segmentation. → p. 100

[10†] L. Kürzinger, D. Winkelbauer, L. Li, T. Watzel, and G. Rigoll. CTC-Segmentation of Large Corpora for German End-to-End Speech Recognition. In A. Karpov and R. Potapova, editors, *Speech and Computer*, pp. 267–278, Cham, Oct. 2020. Springer International Publishing. https://doi.org/10.1007/978-3-030-60276-5_27. arXiv:2007.09127. → pp. 91, 98, 100, 103

[11†] L. Kürzinger. Lightweight Sinc Convolutions for Espnet2 By lumaku · pull Request Nr. 2768 · espnet/espnet, Dec. 2020. https://github.com/espnet/espnet/pull/2768. → p. 86

[12†] L. Li, Y. Kang, Y. Shi, L. Kürzinger, T. Watzel, and G. Rigoll. Adversarial Joint Training with Self-Attention Mechanism for Robust End-to-End Speech Recognition. *arXiv preprint arXiv:2104.01471*, 2021. https://dx.doi.org/10.1186/s13636-021-00215-6.

[13†] L. Li, L. Kurzinger, T. Watzel, and G. Rigoll. A Global Discriminant Joint Training Framework for Robust Speech Recognition. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2021. https://doi.org/10.1109/ictai52525.2021.00088.

[14†] L. Li, Z. Lu, T. Watzel, L. Kürzinger, and G. Rigoll. Light-Weight Self-Attention Augmented Generative Adversarial Networks for Speech Enhancement. *Electronics*, 10(13):1586. MDPI AG, June 2021. https://dx.doi.org/10.3390/electronics10131586.

[15†] L. Li, T. Watzel, L. Kuerzinger, and G. Rigoll. Towards Constructing HMM Structure for Speech Recognition with Deep Neural Fenonic Baseform Growing. *IEEE Access*, 9:39098–39110, 2021. https://dx.doi.org/10.1109/ACCESS.2021.3064197.

[16†] L. Li, Wudamu, L. Kürzinger, T. Watzel, and G. Rigoll. Lightweight End-to-End Speech Enhancement Generative Adversarial Network Using Sinc Convolutions. *Applied Sciences*, 11(16):7564. MDPI AG, Aug. 2021.

[17†] L. Li, X. Zhou, Z. Song, T. Watzel, L. Kürzinger, and G. Rigoll. Deep neural fenonic baseform growing: A novel approach to construct HMM topologies for speech recognition. In *2020 International Conference on High Performance Computing Simulation (HPCS)*. Springer, 2021.

[18†] H. Mandry, A. Herkle, L. Kürzinger, S. Müelich, J. Becker, R. F. H. Fischer, and M. Ortmanns. Modular PUF Coding Chain with High-Speed Reed-Muller Decoder. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2019. https://doi.org/10.1109/ISCAS.2019.8702484.

[19†] G. Maringer, M. Xhemrishi, S. Puchinger, K. Garb, H. Liu, T. Jerkovits, L. Kürzinger, M. Hiller, and A. Wachter-Zeh. Analysis of Communication Channels related to Physically Unclonable Functions. arXiv:2112.02198, 2021.

[20†] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll. Small-Footprint Keyword Spotting on Raw Audio Data with Sinc-Convolutions. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7454–7458. IEEE, 2020. arXiv:1911.02086. → pp. 71, 72, 75, 78, 80, 84, 86

[21†] S. Takamichi, L. Kürzinger, T. Saeki, S. Shiota, and S. Watanabe. JTubeSpeech: corpus of Japanese speech collected from YouTube for speech recognition and speaker verification. *arXiv:2112.09323*, 2021. → pp. 91, 107, 108, 112

[22†] C. Vogl, M. Sackmann, L. Kürzinger, and U. Hofmann. Frenet Coordinate Based Driving Maneuver Prediction at Roundabouts Using LSTM Networks. In *Computer Science in Cars Symposium*. Association for Computing Machinery, New York, NY, USA, Dec. 2020. https://doi.org/10.1145/3385958.3430475.

[23†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Synchronized Forward-Backward Transformer for End-to-End Speech Recognition. In *Speech and Computer*, pp. 646–656. Springer International Publishing, 2020.

[24†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Induced Local Attention for Transformer Models in Speech Recognition. In *Speech and Computer*, pp. 795–806. Springer International Publishing, 2021.

[25†] T. Watzel, L. Kürzinger, L. Li, and G. Rigoll. Regularized Forward-Backward Decoder for Attention Models. In *Speech and Computer*, pp. 786–794. Springer International Publishing, 2021.

[26†] T. Watzel, L. Li, L. Kürzinger, and G. Rigoll. Deep Neural Network Quantizers Outperforming Continuous Speech Recognition Systems. In *Speech and Computer*, pp. 530–539. Springer International Publishing, 2019.

Supervised Student Theses, Seminars, and Internships

[1+] I. Andronic. MP3 Compression as a Means to Improve Robustness against Adversarial Noise Targeting Attention-based End-to-End Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2020. → pp. 49, 51, 52, 55, 59, 68

[2+] R. Baumgartner. Gaussian Process Hyperparameter Optimization. *Research Internship*. Technical University of Munich, 2018.

[3+] F. Bootz. Implementation and Evaluation of the Post-Quantum Secure GPT Encryption Scheme for Embedded Systems. Master's thesis, Technical University of Munich, Germany, 2017.

[4+] U. Budak. Secure Design, Implementation, and Evaluation of Selected DICE-Based Applications with a PUF as a Root of Trust. Master's thesis, Technical University of Munich, Germany, 2023.

[5+] E. R. Chavez Rosas. Improving Robustness of Sequence-to-sequence Automatic Speech Recognition by Means of Adversarial Training. Master's thesis, Technical University of Munich, Germany, 2020. → pp. 49, 52

[6+] N. Günther. Implementation and Evaluation of Keyword Spotting Neural Networks on Microcontrollers. Master's thesis, Technical University of Munich, Germany, 2023.

[7+] S. Heine. Speech Recognition using Machine Learning on a GPU Server. *Research Internship*. Technical University of Munich, 2018.

[8+] M. Hu. Visualization of Attention Activations in the ESPnet Speech Recognition Toolkit. *Research Internship*. Technical University of Munich, 2018.

[9+] M. Hu. Variational Attention for End-to-End Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2020.

[10+] Z. Huang. Input Filter Layers for End-to-end Speech Recognition from Raw Audio. Master's thesis, Technical University of Munich, Germany, 2021.

[11+] S. Ji. Evaluation of Recurrent Neural Networks with Connectionist Temporal Classification for End-to-End Approaches to Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2018.

[12+] P. Klewitz. Shallow Fusion for Attention-based Speech-Recognition. *Interdisciplinary Project*. Technical University of Munich, 2020. → p. 82

[13+] A. Kreutz. A Kaldi Speech Recognition Input Method. *Interdisciplinary Project*. Technical University of Munich, 2018.

[14+] B. Li. Explainable Knowledge Reasoning based on Graph Neural Networks for Open-ended Commonsense Question Answering. Master's thesis, Technical University of Munich, Germany, 2022.

[15+] M. Li. Keyword Detection for Personal Speech-to-Text Assistants. Master's thesis, Technical University of Munich, Germany, 2018.

[16+] Z. Li. Survey and Hardware Implementation of McEliece-Type Post-quantum Cryptography. Master's thesis, Technical University of Munich, Germany, 2017.

[17+] Z. Li. Visualization of Speech Data in the Kaldi Speech Recognition Toolkit. *Research Internship*. Technical University of Munich, 2018.

[18+] Z. Li. Performance and Robustness of Distilled Neural Networks for hybrid Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2019.

[19+] N. Lindae. Sinc Convolutions for End-to-End Speech Recognition. *Interdisciplinary Project*. Technical University of Munich, 2020. → p. 82

[20+] S. Mittermaier. A Lightweight Deep Learning Model for Speech Command Recognition on Raw Audio Data. Master's thesis, Technical University of Munich, 2020. → pp. 71, 75, 86

[21+] E. Murat. Speech Recognition using GANs. *Bachelor's Thesis*. Technical University of Munich, 2019.

[22+] T. Neumaier. Bag-of-Words Classification of Spoken Languages using Vector Quantizers. Master's thesis, Technical University of Munich, Germany, 2018.

[23+] L. Perakis. Conversion of human language to machine readable format using Natural Language Processing. Master's thesis, Technical University of Munich, Germany, 2019.

[24+] C. Preisinger. Investigation of Learnable Filters for Discrete Wavelet Decomposition. *Research Internship*. Technical University of Munich, 2020.

[25+] L. Sandmeir. Optimierung eines auf Verkettung basierenden Decodieralgorithmus für RM-Codes. *Bachelor's Thesis*. Technical University of Munich, 2017.

[26+] J. Schuchardt. Speech Recognition with Vector Quantized Attention-based Encoders. *Interdisciplinary Project*. Technical University of Munich, 2019.

[27+] Z. Song. Generative Adversarial Networks for Hybrid Speech Recognition in Pytorch-kaldi. *Research Internship*. Technical University of Munich, 2019.

[28+] C. Vogl. Driving Maneuver Prediction at Roundabouts Using LSTM Networks. Master's thesis, Technical University of Munich, Germany, 2020.

[29+] Y. Wang. Adversarial Training for Improving Robustness in Hybrid Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2019.

[30+] Y. Wang. Adversarial Training for Robust Hybrid Speech Recognition. *Research Internship*. Technical University of Munich, 2019.

[31+] Y. Wang. Generative Adversarial Networks for hybrid Speech Recognition. *Research Internship*. Technical University of Munich, 2019.

[32+] K. Weiss. Post-Quantum-secure Asymmetric Encryption with QC-MDPC Codes for mbedTLS. *Interdisciplinary Project*. Technical University of Munich, 2017.

[33+] F. Will. Exploration of Generative Neural Networks for Hybrid Speech Recognition. Master's thesis, Technical University of Munich, Germany, 2018.

[34+] D. Winkelbauer. End-to-end Speech Recognition with Attention-based Models for German. *Interdisciplinary Project*. Technical University of Munich, 2019. → pp. 91, 103

[35+] Y. Wu. Variational Autoencoders and Vector-Quantizing Autoencoders for Speech Data. *Research Internship*. Technical University of Munich, 2018.

[36+] C. Xiang. Adversarial Training for Robust Speech Recognition in Pytorch-kaldi. *Research Internship*. Technical University of Munich, 2019.

[37+] H. Zhao. Post-Quantum-secure Authentication based on the Learning Parity Problem. Master's thesis, Technical University of Munich, Germany, 2017.