Technische Universität München

Fakultät für Informatik
Lehrstuhl für Bildverarbeitung und Künstliche Intelligenz

Learning by Association

Strategies for solving computer vision tasks with less labeled data

Philip Häusser

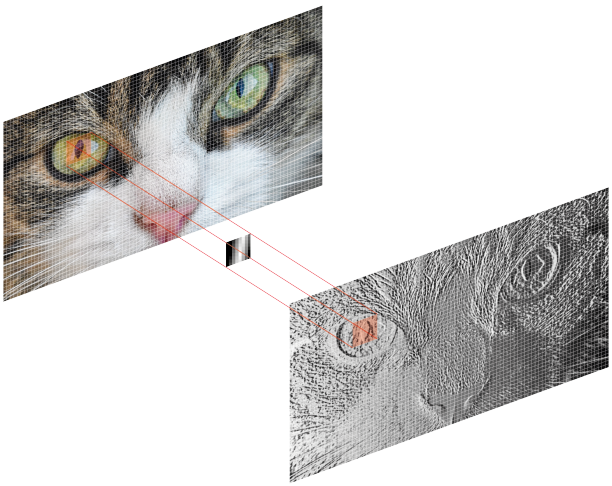
Vollständiger Abdruck der von der
Fakultät für Informatik der Technischen Universität München
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigten Dissertation.

Vorsitzender: Prof. Dr. Darius Burschka
Prüfer der Dissertation: 1. Prof. Dr. Daniel Cremers
2. Prof. Dr. Thomas Brox

Die Dissertation wurde am 26.06.2018 bei der
Technischen Universität München
eingereicht und durch die
Fakultät für Informatik am 21.10.2018 angenommen.

LEARNING BY ASSOCIATION

PHILIP HÄUSSER



Strategies for solving computer vision tasks with less labeled data

ABSTRACT

The world is undergoing a transformation. Self-learning systems and so-called *artificial intelligence* facilitate the automation of many tasks that were thought infeasible for machines. In the field of computer vision in particular, this revolution was taken to the next step by artificial neural networks, more precisely convolutional neural networks (CNNs) which hold state of the art in image and video classification, segmentation, regression on optical flow, scene flow and many other tasks.

However, unlike human learners, CNNs require vast amounts of labeled training data. This is often costly or even impossible to obtain. In this dissertation, we propose and investigate a new approach to solve this problem for typical tasks in computer vision: *learning by association*.

This training schedule is suitable for any embedding learning task such as classification, domain adaptation, clustering and multimodal learning. The key idea is to *associate* training examples for which labels are known with unlabeled examples. These associations happen in a neural network's embedding space where a novel cost function facilitates state-of-the-art results with significantly less labeled training data.

Neural networks are a key component to the development of artificial intelligence. Once we understand how they can be trained effectively, their benefit for humanity will be fully unleashed. Coming a step closer to this goal is the motivation of this work.

ZUSAMMENFASSUNG

Die Welt befindet sich im Umbruch. Selbstlernende Systeme und so genannte *künstliche Intelligenzen* ermöglichen die Automatisierung vieler Aufgaben, deren Lösung durch Maschinen lange als unmöglich galt. Insbesondere im Bereich der Bildverarbeitung und des Computersehens wurde diese Revolution durch Neuronale Netze ermöglicht, genauer gesagt durch Neuronale Faltungs-Netze (CNNs). Diese Modelle sind unübertroffen in Bild- und Videoklassifizierung, sowie Segmentierung, Regression des optischen Flusses, des Szenenfluss und vieler weiterer Aufgaben.

Anders als der lernende Mensch benötigen CNNs beträchtliche Mengen an annotierten Trainingsdaten. Diese zu erlangen ist oft teuer oder gar unmöglich. In dieser Dissertation präsentieren und analysieren wir einen Lösungsansatz für typische Aufgaben in der Bildverarbeitung: *Assoziatives Lernen*.

Diese Trainingsprozedur ist die für alle Formen des Lernens von Darstellungen, bzw. Merkmalen geeignet: Klassifizierung, Domänenanpassung, Clustering und multimodales Lernen. Die Schlüsselidee ist es, Trainingsbeispiele mit bekannter Annotierung mit unbeschrifteten Beispielen zu *assoziieren*. Diese Assoziierungen finden im Einbettungsraum eines neuronalen Netzes statt, wo eine neuartige Kostenfunktion erstklassige Ergebnisse bei deutlich reduziertem Bedarf an annotierten Trainingsdaten ermöglicht.

Neuronale Netze sind eine Schlüsselkomponente für die Entwicklung von künstlicher Intelligenz. Sobald wir verstehen, wie neuronale Netze effektiv trainiert werden können, wird sich ihr Nutzen für die Menschheit völlig entfalten. Diesem Ziel ein Schritt näher zu kommen ist die Motivation dieser Arbeit.

ACKNOWLEDGMENTS

When I began my studies in 2010, I was not remotely envisaging a doctoral degree. It is due to a few very special people that I decided to dare this endeavor and now am holding this document in my hands.

First of all, I would like to thank my advisor, Prof. Daniel Cremers. In 2014, when I was working on my Master's, I got in touch with machine learning and neural networks. Searching for opportunities to deepen my knowledge on these important and exciting topics, Daniel invited me to his chair at the TUM and generously offered me a position in his group. I am very thankful for the chance he gave me. Besides being a brilliant scholar, Daniel is amenable for everyone. His way of explaining most complicated things in simple words is a great inspiration for me.

I started my PhD co-supervisedly with Prof. Patrick van der Smagt whom I would like to thank a lot for the outstanding opportunity to learn about neural networks. Dr. Justin Bayer and Dr. Christian Osendorfer deserve my highest praise for their exemplary teaching skills.

My first project was joint work with the computer vision group at the University of Freiburg, headed by Prof. Thomas Brox. Together with a few of his PhD students and other members of my group, I worked on *FlowNet*, the first neural network to estimate optical flow. In particular, I would like to mention Dr. Alexey Dosovitskiy and Dr. Philipp Fischer who showed me so many tricks and helped me get ramped up in the field of deep learning. Our joint projects were extremely insightful and a great personal experience. I would like to thank Thomas, Alexey and Philipp for amazing hackathons, inspiring conversations and fantastic team work.

In 2016 and 2017, I did internships at Google Brain in Zurich. My first host at Google was Alexander Mordvintsev, the inventor of *deep dream*. Alex has a fascinating intuition for neural nets and a brilliant mind that allows him to turn all of his amazing ideas into code. Working with him was very inspiring and I am very proud of the work we did together. The host of my second internship was Dr. Sylvain Gelly. He and his colleagues, in particular Dr. Karol Kurach, taught me how to write code at Google scale. The many discussions on research questions stimulated many new ideas. It was a great experience to work at Google Brain and a substantial part of this dissertation is due to

my time there.

Finally, I would like to thank the people who worked with me on papers and research projects. Thomas Frerix and I spent so many fruitful hours in front of white boards. Vladimir Golkov excelled in thinking out of the box and bringing ideas back on paper in the language of mathematics. Johannes Plapp designed experiments and implemented our ideas brilliantly. There are many other lab members who made the past three years an unforgettable experience and I thank each and every one of them.

Special thanks go to Dr. Silvia Rohr and Leonard Latter for their excellent suggestions to this document.

And of course, there was a life beyond my PhD studies. Thank you Thorsten, Silvia, Sarah, Jacob, Neal and all others for the great time!

My dear parents, Ulrike and Tilman, thank you for the way I am. I love you.

PUBLICATION PREFACE

The contribution of this dissertation is based on the following first-author publications:

Philip Haeusser, Alexander Mordvintsev and Daniel Cremers
Learning by Association — A Versatile Semi-Supervised Training Method for Neural Networks
2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
DOI: 10.1109/CVPR.2017.74

Philip Haeusser, Thomas Frerix, Alexander Mordvintsev and Daniel Cremers
Associative Domain Adaptation
2017 IEEE International Conference on Computer Vision (ICCV)
DOI: 10.1109/ICCV.2017.301

Philip Haeusser, Johannes Plapp, Vladimir Golkov, Elie Aljalbout and Daniel Cremers
Associative Deep Clustering: Training a Classification Network with no Labels
Under review at the time of submission[1]

1 The paper was accepted at *Proceedings of the German Conference on Pattern Recognition (GCPR).*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRODUCTION

1 NEURAL NETWORKS AND DEEP LEARNING

In the field of computer vision, researchers teach computers to make sense of images and videos. For a human, it is easy to generalize from the specific to the broad, e.g. from a single photo of a cat to other instances of cats. Conversely, all a computer sees is an array of numbers representing colors and intensities that seem unrelated to the numbers representing a second photo of a cat. It remains unclear to the machine which pixels are important and which pixels belong together. Understanding the concept of an image or a video is a surprisingly difficult task for a computer but it is the key to progress in any vision-related technology such as robotics, autonomous driving or biomedical engineering.

Researchers have built a plethora of models to tackle the problem of automated understanding of images and videos. One of the most powerful sets of models is the family of *convolutional neural networks (CNNs)*. These models are composed of multiple *layers* that learn hierarchical representations of the raw pixel input. Due to this structure, this field of research is also referred to as *deep learning*. It is possible to train CNNs such that structures in the training data are discovered automatically. From the high level representations, a wide range of outputs can be generated: a classification prediction, a de-blurred image or a segmentation mask.

CNNs are now applied to many tasks in computer vision and beyond. They have revolutionized many realms of computer vision as their predictions are often dramatically more accurate than those of conventional (non-deep-learning) methods [35, 54, 63], more robust against noise [87] and faster by orders of magnitude [18] as, in

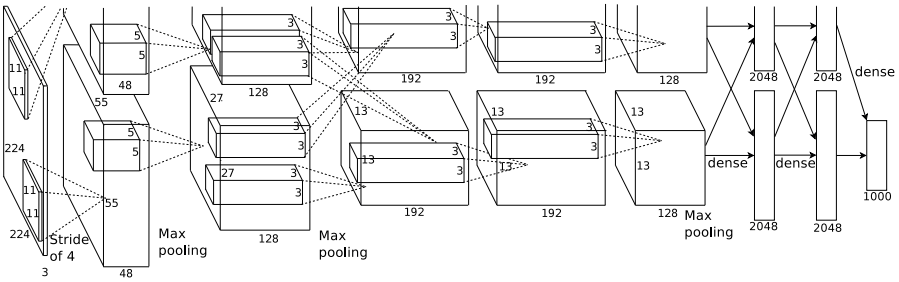


Figure 1.1: Illustration of the AlexNet architecture (2012) [54], the first deep convolutional neural network to outperform all competitors on the Imagenet image classification competition [84] by a large margin. Image taken from the original publication.

many cases, no optimization is necessary once the model is trained. Although neural networks were first conceived in the 1940s [67], it was only in the 2000s that they became computationally tractable and hence very popular. Today, neural networks constitute an entirely new field of research.

In this chapter, we present an introduction to these models, tricks of the trade used today and finally, chances and challenges which arise from deep learning approaches.

1.1 A BRIEF HISTORY OF NEURAL NETWORKS

For a long time, the problem of recognizing patterns in images and videos has been approached with hand-engineered feature extractors such as the *scale-invariant feature transform* (SIFT) [64] or the *histogram of oriented gradients* HOG [13]. The idea is to transform the raw input data to representations (features) that are more useful for the actual task. On a coarse level, this might be already extracting edges in an image by computing gradients (orientations and magnitudes). For more high-level tasks, however, manual transformation rules require precise domain knowledge and do not necessarily result in most efficient representations [21].

The idea of deep neural networks is for the computer to *learn* hierarchical feature representations that extract important information from a dataset and are able to generalize to unseen data. As compelling as this sounds, this is a very under-specified problem. In some cases, even humans struggle to agree on the properties needed to describe and separate one object from another in pixel space.

In 1943, McCulloch and Pitts developed the first computational model for the most basic building block of such a network: an artificial neuron [67]. They constructed an object that receives binary input signals (cf. Figure 1.2). If the sum of the inputs exceeds a threshold $\theta \in \mathbb{R}$, a 1 is output, otherwise 0. It was possible to model simple logic operations such as "AND", "OR" or "NOT" for two binary inputs. However, the thresholds had to be manually chosen.

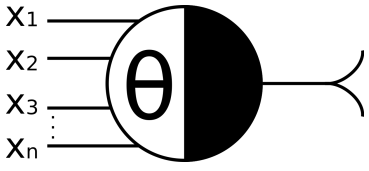


Figure 1.2: McCulloch-Pitts cell. One of the first mathematical formulations of a neuron. Many binary input signals are thresholded. The output signal is binary again. Image source: [56]

In order to model *learning* in artificial neurons, Donald Hebb took inspiration from the neural plasticity mechanism, the ability of neu-

rons to change their connections over time [36]. He proposed connecting multiple neurons together and to multiply their connections with weights $\boldsymbol{W}$. The idea was that neurons that fire together should also be connected with a non-zero weight.

In 1958, Rosenblatt proposed the *perceptron* [81], an algorithm for supervised *learning* of binary classifiers. This can be seen as the archetype of a modern neural network. The model consists of only one single neuron with adjustable weights for its inputs and a threshold that could be trained to convergence if the input data is linearly separable.

In order to deal with more entangled input data that is not linearly separable in the input space and that may contain more than a binary class, Rumelhart, McClelland and Hinton [83] proposed in 1986 the combination of multiple layers of such neurons to allow for a non-linear transformation of the input before the classification. This setup is sometimes referred to as a *feedforward neural network*, although some people consider it to be a subclass thereof. The key factor making such a model possible was the *backpropagation algorithm* which was discovered and developed by multiple researchers in parallel in the 1960's and 1970's [5, 6, 20, 50, 94].

The backpropagation algorithm is a rule that allows to "propagate" the error made by a model back from the output through the model in order to adjust its parameters for better predictions. This will be explained in more detail in Section 1.5

The model that is widely recognized as the first *convolutional neural network* is *LeNet* (Figure 1.3), which was invented by Yann LeCun in 1990 [60] and further developed in the following years [61]. LeCun's work builds upon Fukushima's *neocognitron* from 1979 [22] where *pooling* and *convolution* operations were proposed for hierarchical, multi-layer neural networks. A convolution can be seen as the re-use of weights connecting neurons from one layer to those of the following. Consequently, less parameters are needed, which makes the training of larger models possible. At the same time, convolutions have an intuitive purpose: they *filter* the input for relevant informa-

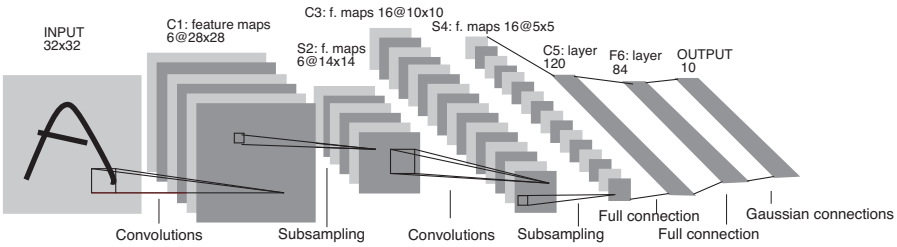


Figure 1.3: Illustration of the LeNet architecture (1998) [62], one if the first convolutional neural networks to be used for handwritten digit recognition. Image taken from the original publication.

tion. The mechanism of a convolution is outlined in the following section.

In 2006, Geoffrey Hinton introduced *Deep Belief Networks* [38], a class of neural networks with many layers (hence, *deep*) that could be trained layer-wise. Most modern deep learning systems are based on this seminal approach, for example *AlexNet* [54] (see Figure 1.1) which outperformed existing methods on the ImageNet image recognition challenge [84] by a large margin. It is named after the inventor Alex Krizhevsky. The important original aspect of his work was the combination of building blocks together with a training scheme that allowed for a significant improvement over the previous (non-deep-learning) state of the art. We discuss these and other components in the next sections.

1.2 BUILDING BLOCKS OF A NEURAL NETWORK

In this section we formally introduce the mathematical framework of a neural network. Vector quantities are printed in boldface.

The goal is to find a model function $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^c$ (parametrized by the model parameters $\boldsymbol{\theta} \in \mathbb{R}^p$) that maps input data $\boldsymbol{x_i}$ to a desired output $\boldsymbol{y_i}$. In the case of RGB-image classification, for example, d would be [3 (number of channels) $\times$ width $\times$ height] and c the number of classes. The predicted class is then the index of the vector $\boldsymbol{y_i}$ with the highest value. Note that here, $\boldsymbol{\theta}$ subsumes *all* parameters such as the previously-mentioned weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$. Here, *bias* is equivalent to a negative threshold value since the bias will be added to the weighted sum of inputs to a neuron.

In the case of feedforward networks, which we consider here, the model function is a nested function: $f(\boldsymbol{x}) = f^{(3)}\left[f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right]$, where each $f^{(l)}$ corresponds to the l-th layer of the network.

Such a layer could, for example, be formed of neurons as defined above: $f^{(l)}(\boldsymbol{z}) = g(\boldsymbol{W}^T\boldsymbol{z} + \boldsymbol{b})$ with weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$. Each neuron's output is usually fed through a piecewise non-linear function $g(z)$. Examples for such non-linear *activation functions* are Sigmoid ($g(z) = (1 + e^{-z})^{-1}$) or ReLu ($g(z) = \max(0, z)$).

We now have the most general formulation of a feedforward neural network, where all neurons of a layer are connected with all neurons of the following one. This setup is called *fully-connected*. From a computational point of view, this is not very scalable since the number of parameters grows exponentially with the number of neurons. One of the big advantages of *convolutional neural networks (CNNs)* [59] is that weights are shared among neurons in a layer, resulting in less parameters for the model. Convolutional neural networks have sparser connections compared to fully-connected networks. That is, rather than multiplying a layer's output with a weight matrix of size $|d_l| \times |d_{l+1}|$ (d_l being the number of units in the l-th layer), we now choose smaller

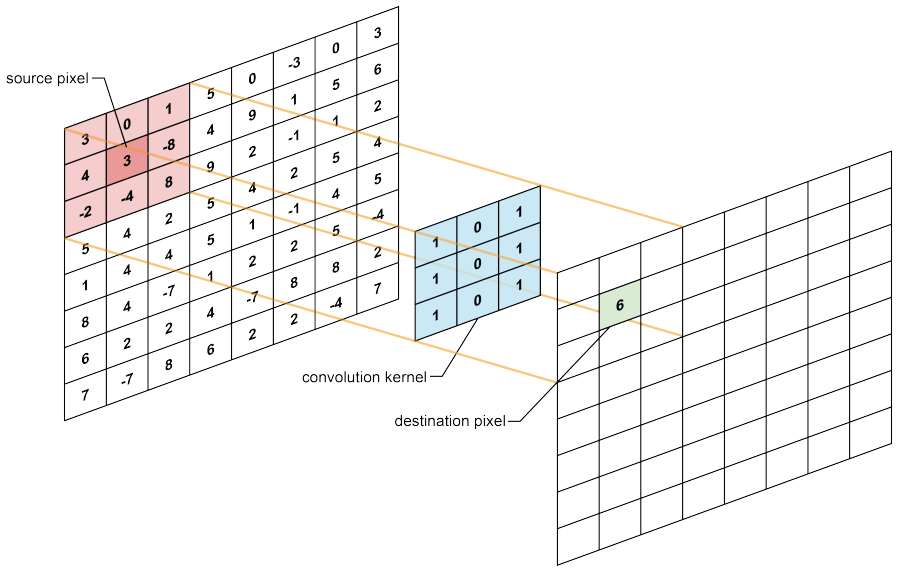


Figure 1.4: Schematic of a convolution. A window about a source pixel (red) is matrix multiplied with a convolution kernel (blue) yielding the destination pixel value (green). In a *deconvolution* (Section 1.4.4), the direction is reversed. The green pixel would be the scalar source which is multiplied with a kernel, resulting in the values for the destination window (red).

kernels of size $k \times k$. Matrix multiplications now happen with kernels shifted over the input resulting in an *activation map* for this kernel. Figure 1.4 illustrates such a convolution. Besides sparse interactions and shared parameters, convolutional networks have a third advantage. Since the same kernel is re-applied multiple times at different locations in the input, a translation invariance is introduced.

One of the first popularized convolutional neural networks was *LeNet* [62] which is depicted in Figure 1.3. Figure 1.1 shows the *AlexNet* architecture which won the 2012 ImageNet Large Scale Visual Recognition Competition 2012.

1.3 PROPERTIES OF NEURAL NETWORKS

One of the most important properties is that neural networks are trainable by adjusting the parameters according to an optimization schedule as will be described in Section 1.5. It is possible to discover structures in a dataset without the need for manually engineered features. Conversely, neural networks are able to extract a hierarchical representation of data [97].

From this follows the capability to approximate functions as stated in the universal approximation theorem [12, 40]: Any such network

with a linear output layer and at least one hidden layer with an activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.

ces are technically built in, such as translation invariance due to the sliding-window nature of convolutions. Moreover, neural networks are able to generalize so as to become invariant against scale, illumination changes, color variations or deformations [45, 54]. The system's adaptability to these variances highly depends on their presence in the training data. This is one reason why training deep neural networks in general requires large (often annotated) datasets.

1.4 ADDITIONAL COMPONENTS

Since deep learning is a very active field of research, there are many contributions whose aim is to make the training of neural networks more efficient and to allow for new applications.

We focus here on some selected contributions, which will play a major role in the rest of this work.

1.4.1 *Dropout*

Dropout was proposed in 2012 by Hinton et al. [39] as a form of regularization. The key idea is to randomly set neural activations to zero with probability p. This results in a different set of connections for each pass through the network, which can be seen as a form of ensemble learning. The network cannot "rely" on single connections since they might "drop out", but has to form multiple ways to obtain a useful representation for input data. Dropout is computationally cheap and was shown to lead to more robust models that are less prone to overfitting [87]. Recently, Gal and Ghahramani [24] showed that dropout applied before every weight layer is mathematically equivalent to an approximation to the *probabilistic deep Gaussian process* [14]. This made it possible to estimate uncertainty by running the same input through the network with active dropout multiple times and then considering the variance of the predictions.

1.4.2 *Batch normalization*

At each layer in a network, input data is non-linearly transformed resulting in output data that potentially has different statistics than the input. Since the parameters of the transformations θ change over time, the statistics of the data passing through the network also change. The output of one layer is the input to the next. Therefore, such changes can cause this shift to grow. This change in the distribution of a func-

tion's domain is called *covariate shift* and can hinder the convergence of a training process. The idea of batch normalization [43] is to normalize data before each layer which reduces the covariance shift. A network equipped with batch normalization is more stable with respect to parameter initialization and can be trained with a higher learning rate.

1.4.3 *Pooling*

Pooling is a form of dimensionality reduction. Pooling layers can be inserted at multiple locations in a neural network. A pooling layer reduces its input data to a sort of summary. This summary can, for example, be the maximum activation of a specific window in the input data. This is called *max-pooling* [61]. The benefit of pooling that less data is necessary to store relevant information. Depending on the task, it is sometimes however beneficial to maintain a high level of spatial information, for example in pixel-wise regression tasks such as segmentation or optical flow estimation. Other tasks, like classification, benefit from strong spatial abstraction.

Pooling is often implemented similarly to the convolution operation (Figure 1.4): For each window in the input data (red), the pooling operation is carried out (e.g. choosing the maximum) and the result is put in the respective location of the destination (green).

The same effect as pooling can be obtained by choosing a stride $s > 1$ for the convolutional layers as shown by Springenberg et al. [86]. The stride s is the number of pixels by which the convolution kernel is shifted across the input.

1.4.4 *Deconvolution and upconvolution*

The opposite operation of a convolution is a *deconvolution*. Input pixels are convolved with a kernel. This results in a scaled copy of the kernel, which is then copied to the output blob [29]. Figure 1.4 illustrates this procedure. An alternative approach to achieve the same result is presented by Dosovitskiy, Springenberg, and Brox [18] where the input is first upsampled and then convolved. The latter will be referred to as *upconvolution*. Both deconvolution and upconvolution can be used to upsample data, for example for pixel-wise classification or regression tasks, e.g. Dosovitskiy et al. [19] or Mayer et al. [66]. Springenberg et al. [86] and Zeiler and Fergus [97] use deconvolutions to visualize concepts and patterns learned by neurons in higher layers by backprojecting activations from the latent space to input (RGB) space.

1.5 OPTIMIZATION

The modules described in the previous section are the building blocks of most neural networks. They define the model structure. Once a model is set up, the goal is to train it for a specific task. The main principle behind training neural networks is known as *empirical risk minimization*[1]. This means that we want to change the parameters $\boldsymbol{\theta}$ of a model so as to optimize its performance on a defined task such as *classification*. This performance is approximated with a cost function that should be minimized by the network. Usually, a cost function is a sum over the cost of all N single training examples $\boldsymbol{x_i}$:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i}^{N} L(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\theta}) \tag{1}$$

A popular way to optimize a model f_θ in order to minimize J is *gradient descent*. It is an iterative procedure where in each step the parameters $\boldsymbol{\theta}$ are changed incrementally in the direction that decreases the cost J:

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\ &= \boldsymbol{\theta} - \eta \frac{1}{N} \sum_{i}^{N} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\theta}) \end{aligned} \tag{2}$$

Here, η is the *learning rate*. Since N might be a very large number, it is usually more practical to *estimate* the true gradient by computing the derivatives over a mini-batch of size N', drawn uniformly from the N training examples [4]. This procedure is also referred to as *stochastic gradient descent* since using only a mini-batch of size $N' \ll N$ introduces stochasticity.

Depending on the form of the cost function, it is possible to formulate *fully-supervised*, *semi-supervised* and *unsupervised* training objectives where all, some or no training examples $\boldsymbol{x_i}$ have a corresponding target value $\boldsymbol{y_i}$, respectively. Throughout this work, we will examine strategies for all three cases.

Once the cost function is set up, gradient descent (Equation 2) is used to update the weights in order to minimize the cost. Naïvely, one would compute the gradient for each component of $\boldsymbol{\theta}$ individually. This is computationally intractable. For this reason, it has been for a long time impossible to successfully train larger models until an efficient training scheme was discovered: backpropagation (cf. Section 1.1). The idea behind backpropagation is to compute the gradient layer-wise, beginning at the last layer before the cost function. Then,

1 or structural risk minimization when regularization is used

by the application of the chain rule, the gradient is computed for the previous layers.

If the mini-batch size N' is too small to capture enough statistically relevant structure of the entire dataset, the optimization will fail since the gradients will contain too much noise. Even with larger N', the "journey" through parameter space during optimization can become erratic. A useful method to avoid too many jumps in parameter space is to introduce a *momentum* to the gradient [70]. There is a physical intuition behind this formulation. Imagine the cost function landscape as a physical landscape with hills and valleys. The goal is to get to the lowest point. Random initialization of the parameters $\boldsymbol{\theta}$ is equivalent to placing a particle at a random spot on this landscape. Gradient descent now means letting the particle move downhill, in the opposite direction of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. In a physical world, the particle would gain momentum if the gradient is consistent for a few subsequent steps (i.e, the particle is moving down on one side of a large hill). The idea is to introduce now a variable $\boldsymbol{v}$ that can be interpreted as the velocity vector of the particle. It is initialized with zero and gets updated as illustrated in the following formulation:

$$\boldsymbol{v} \leftarrow \mu \boldsymbol{v} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{v} \tag{4}$$

The momentum parameter μ is usually chosen in the interval $[0.5, 0.99]$ ([49]). Technically, μ should rather be called *friction* since the velocity vector is effectively damped such that the particle actually comes to rest in the bottom of a valley while at the same time being less prone to get stuck on a saddle point. There exist more sophisticated formulations of momentum such as *Nesterov momentum*, where the correction to a relatively big jump according to $\boldsymbol{v}$ is anticipated.

Also the learning rate η is a hyper parameter requiring manual choice. Recent variants of the gradient descent optimization scheme have attempted to automate the selection of these optimization hyper parameters and to cope with noisy gradients. One of the most popular optimizers is *Adam* [52] which adaptively estimates learning rates with a concept similar to momentum.

1.6 CHANCES AND CHALLENGES

Deep neural networks have dramatically improved the state of the art in many fields of research by discovering hierarchical representations of data. In the past decade, they have been successfully applied to tasks like detection [27], segmentation [63] and object recognition [17]. Beyond these classical computer vision problems, new applications were made possible such as artistic style transfer [25, 82] or image-to-image translation [44, 98].

With highly efficient implementations for graphics processing units (GPUs) becoming more and more available, the duration of inference can often be reduced to a fraction of a second, making many applications real-time capable.

One of the main criticisms has been that simple gradient descent can get stuck in poor local minima. However, it was shown that neural networks nearly always reach solutions of very similar quality [9, 15]. A more serious issue is the *saddle point problem* [15]: Since the gradient near saddle points is almost zero, naïve gradient descent optimizers fail to escape these areas in reasonable time. There are many more saddle points than local minima. It is subject of current research to develop strategies for more efficient saddle point avoidance [47].

Another question has been the reliability of neural networks. As shown in Nguyen, Yosinski, and Clune [73], for example, minimal perturbation of images can lead to dramatic changes in a model's predictions. It is, therefore, the subject of current research to quantify a model's uncertainty and to devise ways to make them more robust against noise or adversarial domain shifts [23, 73].

1.7 MOTIVATION OF THIS THESIS

As long as there is enough training data to sufficiently represent the underlying distribution, neural networks are the model of choice for any machine learning task. Obtaining training data, however, can be a very costly process, in particular if data has to be manually annotated. Since unlabeled data is often abundantly available, the future of deep learning highly depends on new methods to leverage unlabeled data or to reduce the amount of labeled data for training. This thesis presents a method to tackle this problem. We introduce *associative learning*, an approach that facilitates embedding learning by associating labeled and unlabeled data in embedding space.

2 METHODS

2.1 DATA

Deep learning has been extremely data-driven. In conventional setups, deep neural networks are trained on vast amounts of data in order to learn patterns and rules that allow for the model to be applied to new data. For image classification, for example, a training set is required that usually consists of image-label pairs. The more classes there are, the more training examples are needed. Additionally, more variance within the classes (*intra class variance*) necessitates more training data.

Data set name	# labeled training exampes	# classes
MNIST [57]	60k	10
ILSVRC2012* [84]	1.2M	1,000
CIFAR-10 [53]	50k	10
CIFAR-100 [53]	50k	100
SVHN [71]	73k	10

Table 2.1: Examples for image classification data sets with their sizes (rounded) and the number of classes. *) Task 1. Sometimes briefly referred to as the *Imagenet data set*.

It is generally desirable that the training data set for a given deep learning task is *balanced*. That is, each class should contain more or less the same number of examples [8, 69, 91]. The reason for this is simple: If the cost function penalizes wrong classification estimates and 90% of the training examples are from one particular class, the model will just learn this *class prior* and always predict that class, hence achieving a 90% accuracy on the training set without having learned anything about the actual content of the data and the underlying distribution.

If the data set is imbalanced, one strategy is to sample training examples by their class membership in order to ensure that each class is represented equally during training. This method was employed in the works of this thesis for the labeled parts of the data sets [31, 32]. The sampling mechanism is described in detail in Chapter 3. For unlabeled data, this procedure is obviously not applicable. An alternative approach for this case is to assign weak labels to unlabeled examples, e.g. based on the current model estimates [41, 76, 95].

A machine learning model works best when it is applied to or tested on data that has the same distribution as the data that it was trained with [69]. If a neural network is only presented with black cats at training time, it might fail to correctly classify brown cats at testing time. The *color distribution* of training examples is only one of many aspects to be considered. Others are *orientation of objects*, *deformations* or *light conditions*. If these variations are not sufficiently present in the training data set, the model might not *generalize* well. In other words, we assume that there is a true data distribution which we do not have access to but from which we can draw a small subset: the training set.

Depending on what kind of variations are to be expected in the true data distribution, there are various ways to cope with the problem that the distribution of the training data set might not be sufficiently representative. One of the most common strategies is *data augmentation* which will be discussed below.

A discrepancy can not only arise from the fact that the sample size used for training is not large enough, as described above, but also from a situation where training data comes from a different distribution than the data processed at test time. In this case, a *domain adaptation* method can be useful to apply the model trained on one domain or distribution to another one. This is detailed in Chapter 4 where a deep-learning strategy for domain adaptation is presented.

Such discrepancies between data sets can be measured in different ways. One family of methods looks at the distance between probability distributions. The Kullback–Leibler (KL) divergence $D_{KL}(P\|Q)$ between the distributions P and Q is such a quantity [69]:

$$D_{KL}(P\|Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)} \tag{5}$$

The question is, however, how to turn a data set into an object that can be interpreted as a (probability) distribution. Neural networks offer an intuitive way for this. They have been shown to produce useful embeddings for data [74]. The activations of the last few layers for example can be used as the embeddings of an input image. This way, a high-dimensional image can be mapped to a low-dimensional vector. Embeddings of a data set obtained like this can be normalized so as to fulfill the properties of a probability distribution and then be used in a metric such as the KL divergence.

Another measure is the maximum mean discrepancy (MMD) [30], defined as the distance between the mean embeddings of two probability distributions in a reproducing kernel Hilbert space $\mathcal{H}_k$ with a characteristic kernel k. The detailed formulation is described in Chapter 4 where this metric is used to evaluate a novel domain adaptation approached proposed within the scope of this thesis.

2.1.1 *Representation*

In computer vision, the most common data modality is the image. Images are usually represented as objects in $\mathbb{R}^{H \times W \times 3}$ where H and W are the height and width and 3 is the number of color channels, e.g. red, green and blue [46]. Image sequences (videos) are represented analogously, with an additional time axis [72].

Data sets come in various formats. Some are a zipped directory of JPEG files [84], some are stored as one single binary file [57] and others are already prepared in a database format such as HDF5 [92]. Here, a trade-off has to be made between convenience and performance. Small datasets such as MNIST can be easily loaded into memory all at once. Other datasets are too large and need to be accessed in a sequential manner. In particular in the latter case, it is desirable to have the data stored in a format that minimizes the time required to make the information accessible for the learning module. Loading and decompressing single JPEG files, for example, is computationally more costly than preparing a binary database that may take up more space on the disk but can serve data faster in a format that does not need to be decompressed.

2.1.2 *Preparation*

In deep learning, we usually deal with artificial neurons that have a non-linearity around 0 (Section 1.2). Therefore, it makes sense to prepare the data such that the pixel values are centered at 0 and have approximately unit variance. This procedure is called *normalization* [69] and can be achieved in various ways.

A simple approach is to normalize data according to the expected range. For example, images that are represented with pixel values in the interval $[0, 255]$ can be normalized like so: $x' = (x - 127.5)/127.5$. The new pixel range is now $[-1, 1]$. However, depending on the distribution of pixel brightnesses and the precision of the data type, one can lose information here. If most of the pixel values are more or less distributed around the middle value 127.5, this normalization technique has the effect that the largest portion of pixels now lies in an interval $[-\epsilon_1, \epsilon_2]$ where $\epsilon_{1,2} \ll 1$.

A more sophisticated approach is to use the statistics of the data set in order to normalize the data. The normalization rule could then be $x' = (x - \mathbb{E}[x])/\mathrm{var}[x]$ where $\mathbb{E}$ is the *expectation value* and var the *variance* of the dataset. Both statistical moments can be taken over all pixels of all channels or taken per channel or even per pixel location (i.e. an *average image* is computed). This approach is also referred to as *standardization* [51]. The transformed data will have zero mean and unit variance.

2.1.3 *Augmentation*

In order to model the variations in a data set, various *augmentation* schemes can be applied to the input images [28]. Some examples are given in Figure 2.1. An augmentation operation usually takes one or more parameters, e.g. the angle for a rotation. These parameters can be randomly sampled for each training iteration and technically allow for an infinite number of unique training examples – at least in terms of pixel values. Of course, the semantic content of an image and its slightly rotated copy is quite the same.

Most augmentation methods fall in the following categories.

Affine transformations

The general idea of *affine transformations* is to model the fact that objects in the real world can be captured from different perspectives and/or have different deformations and sizes [68]. A computationally relatively efficient way to do this is by applying an affine transformation $F(\boldsymbol{x}) = A\boldsymbol{x} + B$ with $F : \mathbb{R}^2 \to \mathbb{R}^2$. The pixel intensity values located at position $\boldsymbol{x}$ in an input image are mapped to new coordinates in an output image. A is a matrix in $\mathbb{R}^{2\times 2}$ and B is a vector in $\mathbb{R}^2$. This allows for the application of translation, rotation, isotropic scaling and shearing in one step.

Chromatic transformations

When an image is captured, optical phenomena can alter the appearance greatly although the object has not changed. In order to model these variances, one can algorithmically change the image brightness, hue and saturation. It is important to carefully choose the ranges from which parameters for these transformations are sampled. Distortions that are too strong can cause *clipping*, where pixels take values larger or greater than the allowed values and hence local contrast is not preserved and image information is lost.

Other transformations

To model defocused images or the effects of moving objects, images can be blurred, for example by convolving with a Gaussian kernel. Often, images are additionally horizontally flipped (*mirrored*), which, of course, only makes sense if the result is still a valid training example. In the case of written characters for instance, this is usually not the case. The more one knows about the data generation process, the better one can model variances. This can be the simple addition of noise to random pixels or more sophisticated rendering of virtual fog.

Figure 2.1: Examples for different types of data augmentation. Left: The "Lena" image [79], middle and right: examples from Imagenet and SVHN, respectively

2.1.4 *Ingestion*

At some point in the training process, data has to be loaded from the hard disk. Regardless of the format in which the data is stored, it is often useful to carefully design data input pipelines in order to avoid bottlenecks caused by slow data loading processes. Figure 2.2 shows an example what a typical input pipeline setup could look like.

It is common practice to use *prefetching threads* [16], processes that continuously load data from the disk even if the training program is not requiring new data at the moment. The training examples are then cached in a queue from which they can be loaded quickly into the main program without the need for it to pause until data is loaded from disk.

Since it is usually desirable to shuffle the training data set, it is useful to set up the prefetching module so that data is randomly loaded already. For example, if the data set is represented as a list of paths to image files, this list can be loaded entirely into memory, shuffled and then the respective image files are continuously loaded into the input queue.

Most recent deep learning frameworks have existing data input pipeline modules for data sets represented as lists of paths or databases. Another task these modules usually take care of is *batching*. Data is then loaded into a big queue and popped as batches of a given size. The dimensionality of a training batch is usually $N \times H \times W \times C$ where N is the *batch size*, H and W are the image *height* and *width* and C is the *number of channels*.

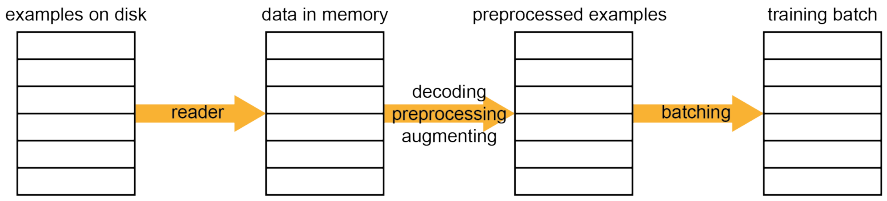


Figure 2.2: Example for a data input pipeline. White blocks stand for data, either on disk or in queues in memory. Yellow arrows denote operations. Training data is stored on disk in the form of multiple single files or as one coherent database that can be accessed randomly. A reader selects single examples, potentially in a shuffled order, and loads the data into the memory. The following module decodes the data if necessary, i.e. transforms a list of bytes to an image tensor. At this point, further preprocessing steps can be taken such as augmentation or whitening. The preprocessed examples are now kept in a queue from which batches are formed to be sent to the actual training process.

2.1.5 *GPU usage*

Employing *graphics processing units* (GPUs) has demonstrated tremendous gains in processing speed due to their optimization for matrix operations compared to pure CPU implementations [53]. Most current deep learning frameworks offer GPU implementations of the operations which form the building blocks of a neural network. Custom operations have to be implemented individually by the user. The predominant programming language is *CUDA* which can be used to program GPUs from NVIDIA. Recently, also other chip manufacturers are entering the deep learning hardware market [26]. Google announced their own *tensor processing unit (TPU)* hardware [48].

When creating a new model setup, it is recommended to do the preprocessing in a separate thread on the CPU and to use the GPU exclusively for training [90]. The reason for this is that preprocessing usually takes less computational resources and can be done in parallel to the training. Data is then ready to be fetched as one mini-batch by the actual training routine. This can save time.

2.2 TRAINING FRAMEWORKS

Today, there is a plethora of different frameworks for deep learning. One of the earlies that emerged was caffe [46]. Here, a neural network is described as a computational graph which is defined in a text file. Each layer and its parameters have to be defined before execution. Also a solver describing the optimization parameters is defined in the same way. The main training process uses both configuration files to instantiate the operations and to run the optimization. Operations can be implemented in C++ (to be executed on the CPU) or in CUDA (for execution on the GPU).

One of the drawbacks of this early framework was that the user could not communicate with the training process while it was running. If one wanted to change the learning rate, for example, it was necessary to kill the process, change the configuration file and restart it again. This is not very elegant nor efficient since only the last snapshot of the model parameters is saved, hence a certain number of iterations is lost. Moreover, reinstantiating the model takes some extra time.

As a part of this work, we adapted the caffe code to alleviate this problem. Changes were made such that the solver process monitors updates in the configuration file while it is running and adapts its parameters accordingly on the fly. This improvement was used in Dosovitskiy et al. [19] and Mayer et al. [66] to facilitate training of state-of-the-art networks.

One more improvement was made to address another shortcoming of the caffe training framework: the insufficient abilities for the user

to monitor training progress beyond values of cost function and test accuracy values. To this end, we designed the *Vizmaster*, a visualization tool for real-time analysis of loss gradients in the different layers and current parameters (convolution kernels). A key insight from this was that it is useful to react when the gradients in specific layers go towards zero or infinity (*vanishing* ir *exploding* gradients). Monitoring the aforementioned quantities allows to notice this behavior sooner than by just watching the overall loss. One strategy that helps in many cases of vanishing or exploding gradients is to reduce the learning rate, to decrease the number of layers or to employ batch normalization.

The other major framework that was used in the scope of this thesis is *TensorFlow* [1]. This framework was developed at Google and open-sourced in 2016. Like in caffe, an acyclic graph of operations is created to carry out the computations in a neural network. One key difference to caffe is that the graph is created programatically and can be altered at runtime. TensorFlow also comes with a visualization suite called *TensorBoard*. Specific quantites in the graph can be configured to be logged for later visualization in TensorBoard.

Both frameworks have an active open-source community and are continuously improved. This short summary is a snapshot of the time at which the code was checked out. More improvements are to be expected.

For completeness, we would like to present an overview of some of the widely used deep learning frameworks in Table 2.2

Name	Initial release	Developed at
Torch [11]	2002	IDIAP
Theano [2]	2010	Université de Montréal
Caffe [46]	2014	UC Berkeley
TensorFlow [1]	2016	Google
PyTorch [75]	2016	Facebook, Uber
Caffe2 [80]	2017	Facebook

Table 2.2: Non-comprehensive overview of deep learning frameworks that are currently widely used.

2.3 TRAINING PROCEDURE

The goal of every training is to apply an optimization rule such as *Adam* in order to minimize the cost function (Section 1.5). At the same time, a multitude of hyper parameters have to be chosen: the network architecture (e.g. number and type of layers and their param-

eters such as convolution kernel sizes), the learning rate, the type of optimizer and its parameters or model-specific settings.

itting. One usually tries many different sets of hyper parameters at the same time in order to find the best one [10]. During training, it is then interesting to plot the current loss and the performance on a validation set. It is also useful to visualize the values of the parameters θ and their gradients in order to find out whether a layer's gradients are vanishing or exploding. This can be done automatically or with human interaction as described above.

In the works presented in this thesis, we did a randomized hyper parameter search. We define ranges from which to sample each hyper parameter. They are either linearly spaced (e.g. kernel size) or exponentially (e.g. learning rate). One can now discretize these ranges and do the cross product of all ranges in order to obtain all possible sets of hyper parameters. Since this number is usually very large, a potentially better approach is to randomly sample each parameter individually. This way, more volume in the hyper parameter space is looked at, compared to a cross-product based grid search with less values for each hyper parameter. On the other hand, there is no guarantee that a specific combination of hyper parameters is tested.

2.3.1 *Distributed training*

The size of available memory, e.g. on the GPU, is one hard limitation. It limits the size of network architectures and the size of mini-batches. More complex tasks often require larger architectures and generally, a larger mini-batch size yields a less noisy estimate of the true gradient. It is therefore desirable to make as much memory available

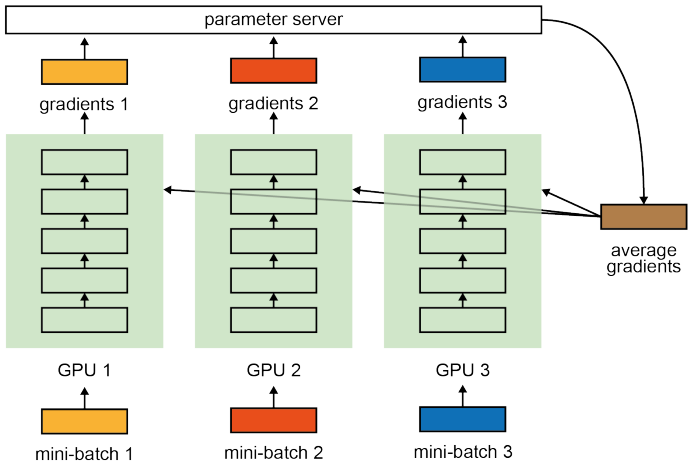


Figure 2.3: Training with distributed data. The same architecture is replicated on multiple GPUs.

as possible. This can be achieved by connecting multiple GPUs and distributing the training process. There are at least two approaches: distributing data or distributing operations [16].

Distributing data

In this approach, the same network is replicated multiple times on multiple GPUs as shown in Figure 2.3. Each replica receives an individual mini-batch and computes the gradients for the parameter update. These gradients are then sent to a parameter server that averages all incoming gradients and broadcasts the update to be applied in each replica. If the bottleneck is that mini-batches did not fit on the GPU, this is a useful approach as the effective mini-batch size scales with the number of replicas. As the mini-batch occupies less memory, the model architecture can be larger. On the other hand, one has to introduce additional infrastructure to communicate the gradients between the various replicas. The network bandwidth could be one limitation. Another problem to be aware of is that the learning rate has to be adapted when this type of distributed training is employed. A rule of thumb is to divide the learning rate that lead to convergence on a single GPU by the number of replicas.

Distributing operations

The alternative approach to distribute training is to split the graph across multiple GPUs. This concept is illustrated in Figure 2.4. The first few layers are computed on GPU1, the following layers on GPU2 et cetera. The same mini-bath is then passed from one GPU to the next. The gradients are back-propagated accordingly and the parameter updates take place on the respective GPUs. While this approach is theoretically very straight-forward, the user still has to make the

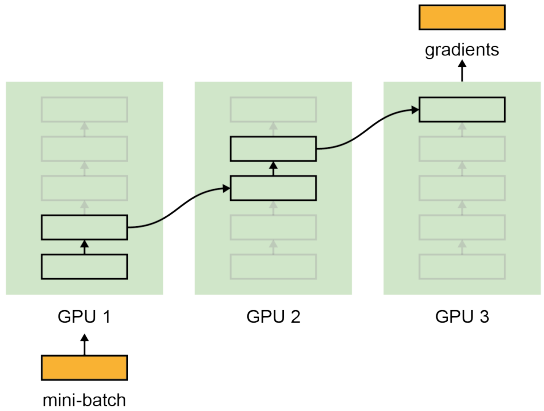


Figure 2.4: Training with distributed operations. The architecture is split across multiple GPUs.

decision where to place which operations. In a sophisticated setup, the (processed) mini-batches are passed from queues to queues so that each GPU is kept busy. Otherwise, only one GPU is being used while the others are waiting for new data.

2.4 EVALUATION OF TRAINING RESULTS

Quantitative analysis

The evaluation of a trained model highly depends on the task. For classification, for example, the average accuracy on a test set is usually reported. The test set contains images that were not used for training. It is good practice to entirely hold out this set until the end in order not to *leak* information from the test set at training time [7]. If, for example, one keeps evaluating a model on a test set and changes hyper-parameters until the test error improves, one has implicitly used the test set at training time.

The average accuracy [53] is usually computed as the expectation value $\mathbb{E}_{x\sim D_T}[\mathbb{1}(f(x) == y)]$ where a test example x from the test set D_T with ground truth label y is compared against the network's prediction $f(x)$. The indicator function $\mathbb{1}$ is 1 if the argument is true and 0 otherwise.

For the problem of classification, it may also make sense to report the confusion matrix [88], which gives more detailed insights into the performance of the machine learning model on the respective classes. An element of a confusion matrix M_{ij} tells how often an example from class i was classified as class j or vice versa.

Other metrics such as ROC curves or precision-recall are rather rarely reported in recent deep learning papers.

Other tasks may have other metrics. For example, in optical flow estimation, the average end-point error is reported [19]. This is the average over the pixel-wise L1 distance between the prediction and the ground truth for non-occluded pixels.

In order to prove the efficiency of a newly proposed model empirically, researchers usually report these quantitative results on different data sets that are made for the same task. A model that performs well on structurally different data sets can be assumed to generalize well. The other extreme would be a model that is specifically designed for one data set and that fails when it is applied to another data set with a different data distribution.

Qualitative analysis

Qualitative analyses are as various as human creativity. Here are just a few examples which are partially also employed in the scope of this work.

Visualizations of the network parameters

One way to argue that a neural network has learned useful convolution kernels was proposed by Zeiler and Fergus [97]. They visualize features using a multi-layered deconvolutional network to project the feature activations back to the input pixel space. An example is shown in Figure 2.5. This works well for some layers and less for others. It can be useful to understand which concepts in the dataset the network has learned, e.g. that there is a prevalence of dogs in the Imagenet dataset. The hierarchical nature of deep neural networks become evident through visualizations like this since. Kernels from lower layers are shown to be activated by more rudimentary patterns than kernels from higher layers. However, when a network can not be visualized well with this method, it is impossible to trace back the problem so as to optimize training or to draw conclusions on the network's performance on a specific task.

Visualizations of nearest neighbors

Whenever a neural network is trained, it is possible to identify a specific layer's activations with the *embedding* of the input. Often, this is done with the last or second to last layer as they typically have the smallest dimensionality, at least in classification networks. In this *embedding space* one can look for embeddings of other inputs and select the nearest neighbors, e.g. by computing the pairwise Euclidean distances. This method allows to explore patterns that a network found in the data. Manual inspection can reveal structure of the data set which was used to optimize the cost function. For example, if the task is classification, one would expect nearest neighbors to belong to the same class. If there are wrongly classified examples, it might be possible to figure out which shapes confused the network and, for example, think of data augmentation strategies that resolve the problem. Examples for this method are given in Section 4.2 of Chapter 3.

t-SNE embeddings

The notion of embeddings described in the previous paragraph can be visualized directly with dimensionality reduction methods such as t-SNE [65]. The key idea is to project high-dimensional vectors in 2 or 3 dimensions with the constraint that pairwise distances in the high-dimensional space should be preserved relatively in the low-dimensional embedding. A reasonable sample of the data set should be used but it is not necessary (nor efficient) to use the entire set for this visualization technique. t-SNE is a useful method to visualize the structure of the embedding space. However, it requires a few hyperparameters to be set such as the *perplexity*. Depending on the choice of these parameters, completely different projections can be obtained. Therefore, the user has to carefully analyze the plots and tune the hy-

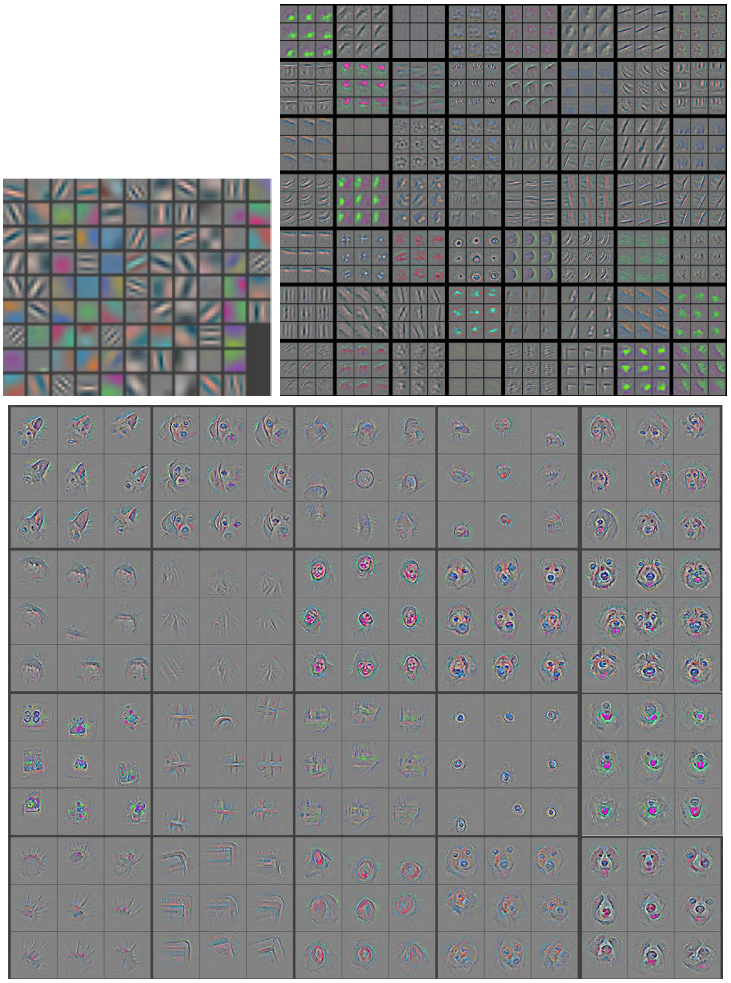


Figure 2.5: Visualizations of a neural network with the method of Zeiler and Fergus Zeiler and Fergus [97]. Shown are back-projected activations of kernels of layers 1 (top left), 2 (top right) and 4 (bottom) of AlexNet [54] after training on Imagenet.

perparameters before jumping to interpretations [93]. Examples for t-SNE plots are given in Section 3.4 of Chapter 4.

Part II

PUBLICATIONS

The task of classification is an example for the versatility of deep neural networks. These powerful models learn to generalize to a multitude of varieties present in real images, such as different perspectives, light conditions, deformations, color, scale and position to name just a few. In the past, a big challenge has been the acquisition of appropriate training data that sufficiently expresses these variances. Frequently, these datasets become very large. The imagenet dataset [84] contains 1.2 million training images from 1,000 classes. These images had to be manually labeled which is a costly task. In many cases, human annotation is intractable. It is therefore desirable to reduce the number of labeled examples dramatically. This part presents a novel approach to solve this problem: *learning by association*. The key idea is to "associate" data in embedding space. The implementation is described in detail in the first chapter along with the application of associative learning to the task of semi-supervised training, where only a small subset of the full dataset is labeled. Then, we apply this approach to domain adaptation. Here, knowledge about labels from one image domain is used to infer labels on a second domain. The following chapter introduces a training scheme where associative learning is used to train a neural network without any labels. In the last chapter, associations are made between two different modalities: images and text.

3

LEARNING BY ASSOCIATION

This chapter is based on Haeusser, Mordvintsev, and Cremers [31]. In many real-world scenarios, labeled data for a specific machine learning task is costly to obtain. Semi-supervised training methods make use of abundantly available unlabeled data and a smaller number of labeled examples. We propose *associative learning*, a new framework for semi-supervised training of deep neural networks inspired by learning in humans. "Associations" are made from embeddings of labeled samples to those of unlabeled ones and back. The optimization schedule encourages correct association cycles that end up at the same class from which the association was started and penalizes wrong associations ending at a different class. The implementation is easy to use and can be added to any existing end-to-end training setup. We demonstrate the capabilities of learning by association on several datasets and show that it can improve performance on classification tasks tremendously by making use of additionally available unlabeled data. In particular, for cases with few labeled data, our training scheme outperforms the current state of the art on SVHN.

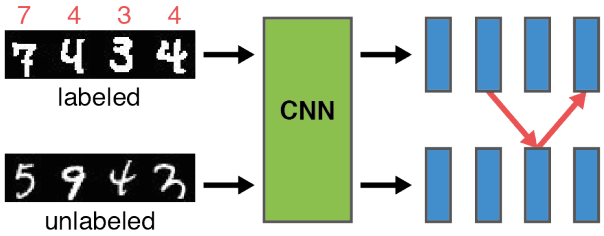


Figure 3.1: Learning by association - overview

CONTRIBUTIONS OF THE AUTHOR

The author of this dissertation was fully in charge of

- planning and carrying out the experiments
- evaluating the experiments
- writing the paper

2017 IEEE Conference on Computer Vision and Pattern Recognition

Learning by Association
A versatile semi-supervised training method for neural networks

Philip Haeusser[1,2] Alexander Mordvintsev[2] Daniel Cremers[1]

[1]Dept. of Informatics, TU Munich [2]Google, Inc.

{haeusser, cremers}@in.tum.de moralex@google.com

Abstract

In many real-world scenarios, labeled data for a specific machine learning task is costly to obtain. Semi-supervised training methods make use of abundantly available unlabeled data and a smaller number of labeled examples. We propose a new framework for semi-supervised training of deep neural networks inspired by learning in humans. "Associations" are made from embeddings of labeled samples to those of unlabeled ones and back. The optimization schedule encourages correct association cycles that end up at the same class from which the association was started and penalizes wrong associations ending at a different class. The implementation is easy to use and can be added to any existing end-to-end training setup. We demonstrate the capabilities of learning by association on several data sets and show that it can improve performance on classification tasks tremendously by making use of additionally available unlabeled data. In particular, for cases with few labeled data, our training scheme outperforms the current state of the art on SVHN.

1. Introduction

A child is able to learn new concepts quickly and without the need for millions examples that are pointed out individually. Once a child has seen one dog, she or he will be able to recognize other dogs and becomes better at recognition with subsequent exposure to more variety.

In terms of training computers to perform similar tasks, deep neural networks have demonstrated superior performance among machine learning models ([20, 18, 10]). However, these networks have been trained dramatically differently from a learning child, requiring labels for every training example, following a purely supervised training scheme. Neural networks are defined by huge amounts of parameters to be optimized. Therefore, a plethora of labeled training data is required, which might be costly and time consuming to obtain. It is desirable to train machine learning models without labels (unsupervisedly) or with only some fraction of the data labeled (semi-supervisedly).

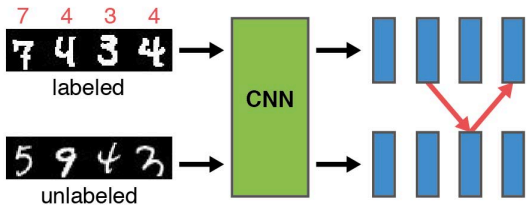


Figure 1. Learning by association. A network (green) is trained to produce embeddings (blue) that have high similarities if belonging to the same class. A differentiable association cycle (red) from embeddings of labeled (A) to unlabeled (B) data and back is used to evaluate the association.

Recently, efforts have been made to train neural networks in an unsupervised or semi-supervised manner yielding promising results. However, most of these methods require a trick to generate training data, such as sampling patches from an image for context prediction [6] or generating surrogate classes [7, 22, 13]. In other cases, semi-supervised training schemes require non trivial additional architectures such as generative adversarial networks [9] or a decoder part [39].

We propose a novel training method that follows an intuitive approach: learning by association (Figure 1). We feed a batch of labeled and a batch of unlabeled data through a network, producing embeddings for both batches. Then, an imaginary walker is sent from samples in the labeled batch to samples in the unlabeled batch. The transition follows a probability distribution obtained from the similarity of the respective embeddings which we refer to as an *association*. In order to evaluate whether the association makes sense, a second step is taken back to the labeled batch - again guided by the similarity between the embeddings. It is now easy to

1063-6919/17 $31.00 © 2017 IEEE
DOI 10.1109/CVPR.2017.74

check if the cycle ended at the same class from which it was started. We want to maximize the probability of consistent cycles, i.e., walks that return to the same class. Hence, the network is trained to produce embeddings that capture the essence of the different classes, leveraging unlabeled data. In addition, a classification loss can be specified, encouraging embeddings to generalize to the actual target task.

The association operations are fully differentiable, facilitating end-to-end training of arbitrary network architectures. Any existing classification network can be extended by our customized loss function.

In summary, our key contributions are:

- A novel yet simple training method that allows for semi-supervised end-to-end training of arbitrary network architectures. We name the method "associative learning".
- An open-source TensorFlow implementation[1] of our method that can be used to train arbitrary network architectures.
- Extensive experiments demonstrating that the proposed method improves performance by up to 64% compared to the purely supervised case.
- Competitive results on MNIST and SVHN, surpassing state of the art for the latter when only a few labeled samples are available.

2. Related Work

The challenge of harnessing unlabeled data for training of neural networks has been tackled using a variety of different methods. Although this work follows a semi-supervised approach, it is in its motivation also related to purely unsupervised methods. A third category of related work is constituted by generative approaches.

2.1. Semi-supervised training

The semi-supervised training paradigm has not been among the most popular methods for neural networks in the past. It has been successfully applied to SVMs [14] where unlabeled samples serve as additional regularizers in that decision boundaries are required to have a broad margin also to unlabeled samples.

One training scheme applicable to neural nets is to bootstrap the model with additional labeled data obtained from the model's own predictions. [22] introduce pseudo-labels for unlabeled samples which are simply the class with the maximum predicted probability. Labeled and unlabeled samples are then trained on simultaneously. In combination with a denoising auto-encoder and dropout, this approach yields competitive results on MNIST.

[1] https://git.io/vyzrl

Other methods add an auto-encoder part to an existing network with the goal of enforcing efficient representations ([27] [37] [39]).

Recently, [30] introduced a regularization term that uses unlabeled data to push decision boundaries of neural networks to less dense areas of decision space and enforces mutual exclusivity of classes in a classification task. When combined with a cost function that enforces invariance to random transformations as in [31], state-of-the-art results on various classification tasks can be obtained.

2.2. Purely unsupervised training

Unsupervised training is obviously more general than semi-supervised approaches. It is, however, important to differentiate the exact purpose. While semi-supervised training allows for a certain degree of guidance as to what the network learns, the usefulness of unsupervised methods highly depends on the design of an appropriate cost function and balanced data sets. For exploratory purposes, it might be desirable that representations become more fine grained for different suptypes of one class in the data set. Conversely, if the ultimate goal is classification, invariance to this very phenomenon might be more preferable.

[12] propose to use Restricted Boltzmann Machines ([33]) to pre-train a network layer-wise with unlabeled data in an auto-encoder fashion.

[11][19][39] build a neural network upon an auto-encoder that acts as a regularizer and encourages representations that capture the essence of the input.

A whole new category of unsupervised training is to generate surrogate labels from data. [13] employ clustering methods that produce weak labels.

[7] generate surrogate classes from transformed samples from the data set. These transformations have hand-tuned parameters making it non-trivial to ensure they are capable of representing the variations in an arbitrary data set.

In the work of [6], context prediction is used as a surrogate task. The objective for the network is to predict the relative position of two randomly sampled patches of an image. The size of the patches needs to be manually tuned such that parts of objects in the image are not over- or undersampled.

[34] employ a multi-layer LSTM for unsupervised image sequence prediction/reconstruction, leveraging the temporal dimension of videos as the context for individual frames.

2.3. Generative Adversarial Nets (GANs)

The introduction of generative adversarial nets (GANs) [9] enabled a new discipline in unsupervised training. A generator network (G) and a discriminator network (D) are trained jointly where the G tries to generate images that look as if drawn from an unlabeled data set, whereas D is supposed to identify the difference between real samples

and generated ones. Apart from providing compelling visual results, these networks have been shown to learn useful hierarchical representations [26].

[32] presents improvements in designing and training GANs, in particular, these authors achieve state-of-the-art results in semi-supervised classification on MNIST, CIFAR-10 and SVHN.

3. Learning by association

A general assumption behind our work is that good embeddings will have a high similarity if they belong to the same class. We want to optimize the parameters of a CNN in order to produce good embeddings, making use of both labeled and unlabeled data. A batch of labeled and unlabeled images (A_{img} and B_{img}, respectively) is fed through the CNN, resulting in embedding vectors (A and B). We then imagine a walker going from A to B according to the mutual similarities, and back. If the walker ended up at the same class as he started from, the walk is correct. The general scheme is depicted in Figure 1.

3.1. Mathematical formulation

The goal is to maximize the probability for correct walks from A to B and back to A, ending up at the same class. A and B are matrices whose rows index the samples in the batches. Let's define the **similarity between embeddings A and B** as

$$M_{ij} \coloneqq A_i \cdot B_j \tag{1}$$

Note that the dot product could in general be replaced by any other similarity metric such as Euclidean distance. In our experiments, the dot product worked best in terms of convergence. Now, we transform these similarities into **transition probabilities from A to B** by softmaxing M over columns:

$$\begin{aligned} P^{ab}_{ij} = P(B_j|A_i) &\coloneqq (\text{softmax}_{\text{cols}}(M))_{ij} \\ &= \exp(M_{ij}) / \sum_{j'} \exp(M_{ij'}) \end{aligned} \tag{2}$$

Conversely, we get the transition probabilities in the other direction, P^{ba}, by replacing M with M^T. We can now define the **round trip probability** of starting at A_i and ending up at A_j:

$$\begin{aligned} P^{aba}_{ij} &\coloneqq (P^{ab}P^{ba})_{ij} \\ &= \sum_k P^{ab}_{ik} P^{ba}_{kj} \end{aligned} \tag{3}$$

Finally, the **probability for correct walks** becomes

$$P(\text{correct walk}) = \frac{1}{|A|} \sum_{i \sim j} P^{aba}_{ij} \tag{4}$$

where $i \sim j \Leftrightarrow \text{class}(A_i) = \text{class}(A_j)$.

We define multiple losses that encourage intuitive goals. These losses can be combined, as discussed in Section 4.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{walker}} + \mathcal{L}_{\text{visit}} + \mathcal{L}_{\text{classification}} \tag{5}$$

Walker loss. The goal of our association cycles is consistency. A walk is consistent when it ends at a sample with the same class as the starting sample. This loss penalizes incorrect walks and encourages a uniform probability distribution of walks to the correct class. The uniform distribution models the idea that it is permitted to end the walk at a different sample than the starting one, as long as both belong to the same class. The walker loss is defined as the cross-entropy H between the uniform target distribution of correct round-trips T and the round-trip probabilities P^{aba}.

$$\mathcal{L}_{\text{walker}} \coloneqq H(T, P^{aba}) \tag{6}$$

with the uniform target distribution

$$T_{ij} \coloneqq \begin{cases} 1/\#\text{class}(A_i) & \text{class}(A_i) = \text{class}(A_j) \\ 0 & \text{else} \end{cases} \tag{7}$$

where $\#\text{class}(A_i)$ is the number of occurrences of $\text{class}(A_i)$ in A.

Visit loss. There might be samples in the unlabeled batch that are difficult, such as a badly drawn digit in MNIST. In order to make best use of all unlabeled samples, it should be beneficial to "visit" all of them, rather than just making associations among "easy" samples. This encourages embeddings that generalize better. The visit loss is defined as the cross-entropy H between the uniform target distribution V and the visit probabilities P^{visit}. If the unsupervised batch contains many classes that are not present in the supervised one, this regularization can be detrimental and needs to be weighted accordingly.

$$\mathcal{L}_{\text{visit}} \coloneqq H\left(V, P^{\text{visit}}\right) \tag{8}$$

where the visit probability for examples in B and the uniform target distribution are defined as follows:

$$P^{\text{visit}}_j \coloneqq \langle P^{ab}_{ij} \rangle_i \tag{9}$$

$$V_j \coloneqq 1/|B| \tag{10}$$

Classification loss. So far, only the creation of embeddings has been addressed. These embeddings can easily be mapped to classes by adding an additional fully connected

layer with softmax and a cross-entropy loss on top of the network. We call this loss classification loss. This mapping to classes is necessary to evaluate a network's performance on a test set. However, convergence can also be reached without it.

3.2. Implementation

The total loss $\mathcal{L}_{\text{total}}$ is minimized using Adam [16] with the suggested default settings. We applied random data augmentation where mentioned in Section 4. The training procedure is implemented end-to-end in TensorFlow [1] and the code is publicly available.

4. Experiments

In order to demonstrate the capabilities of our proposed training paradigm, we performed different experiments on various data sets. Unless stated otherwise, we used the following network architecture with batch size 100 for both labeled batch A (10 samples per class) and unlabeled batch B:

$$\begin{aligned} & C(32,3) \rightarrow C(32,3) \rightarrow P(2) \\ \rightarrow\ & C(64,3) \rightarrow C(64,3) \rightarrow P(2) \\ \rightarrow\ & C(128,3) \rightarrow C(128,3) \rightarrow P(2) \rightarrow FC(128) \end{aligned}$$

Here, $C(n,k)$ stands for a convolutional layer with n kernels of size $k \times k$ and stride 1. $P(k)$ denotes a max-pooling layer with window size $k \times k$ and stride 1. $FC(n)$ is a fully connected layer with n output units.

Convolutional and fully connected layers have exponential linear units (elu) activation functions [3] and an additional L2 weight regularizer with weight 10^{-4} applied.

There is an additional FC layer, mapping the embedding to the logits for classification after the last FC layer that produces the embedding, i.e., $FC(10)$ for 10 classes.

4.1. MNIST

The MNIST data set [21] is a benchmark containing handwritten digits for supervised classification. Mutual exclusivity regularization with transformations ([31]) have previously set the state of the art among semi-supervised deep learning methods on this benchmark. We trained the simple architecture mentioned above with our approach with all three losses from Section 3.1 and achieved competitive results as shown in Table 1. We have not even started to explore sophisticated additional regularization schemes that might further improve our results. The main point of these first experiments was to test how quickly one can achieve competitive results with a vanilla architecture, purely by adding our proposed training scheme. In the following, we explore some interesting, easily reproducible properties.

4.1.1 Evolution of associations

The untrained network is already able to make some first associations based on the produced embeddings. However, many wrong associations are made and only a few samples in the unsupervised batch (B) are visited: those most similar to the examples in the supervised batch (A). As training progresses, these associations get better. The visit loss ensures that all samples in B are visited with equal probability. Figure 2 shows this evolution. The original samples for a setup with 2 labeled samples per class are shown where A is green and B is red. Associations are made top-down. Note that the second set of green digits is equal to the first ("round-trip"). The top graphic in Figure 2 shows visit probabilities at the beginning of training. Darker lines denote a higher probability (softmaxed dotproduct). The bottom graphic in Figure 2 shows associations after training has converged. This took 10k iterations during which only the same 20 labeled samples were used for A and samples for B were drawn randomly from the rest of the data set, ignoring labels.

4.1.2 Confusion analysis

Even after training has converged, the network still makes mistakes. These mistakes can, however, be explained. Figure 3 shows a confusion matrix for the classification task. On the left side, all samples from the labeled set (A) are shown (10 per class). Those samples that are classified incorrectly express features that are not present in the supervised training set, e.g. "7" with a bar in the middle (mistaken for "2") or "4" with a closed loop (mistaken for "9"). Obviously, A needs to be somewhat representative for the data set, as is usually the case for machine learning tasks.

4.2. STL-10

STL-10 is a data set of RGB images from 10 classes [4]. There are 5k labeled training samples and 100k unlabeled training images from the same 10 classes and additional classes not present in the labeled set. For this task we modified the network architecture slightly as follows:

$$\begin{aligned} & C(32,3) \rightarrow C(64,3,\text{stride=2}) \rightarrow P(3) \\ \rightarrow\ & C(64,3) \rightarrow C(128,3) \rightarrow P(2) \\ \rightarrow\ & C(128,3) \rightarrow C(256,3) \rightarrow P(2) \rightarrow FC(128) \end{aligned}$$

As a preprocessing step, we apply various forms of data augmentation to all samples fed though the net. In particular, random cropping, changes in brightness, saturation, hue and small rotations.

We ran training using 100 randomly chosen samples per class from the labeled training set for A (i.e. we used only 20% of the labeled training images) and achieved an accuracy on the test set of 81%. As this is not exactly following

Method	# labeled samples		
	100	1000	All
Ladder, conv small Γ [28]	0.89 (0.50)	-	-
Improved GAN † [32]	0.93 (0.07)	-	-
Mutual Exclusivity + Transform. [31]	**0.55 (0.16)**	-	**0.27 (0.02)**
Ours	0.89 (0.08)	0.74 (0.03)	0.36 (0.03)

Table 1. Results on MNIST. Error (%) on the test set (lower is better). Standard deviations in parentheses. †: Results on permutation-invariant MNIST.

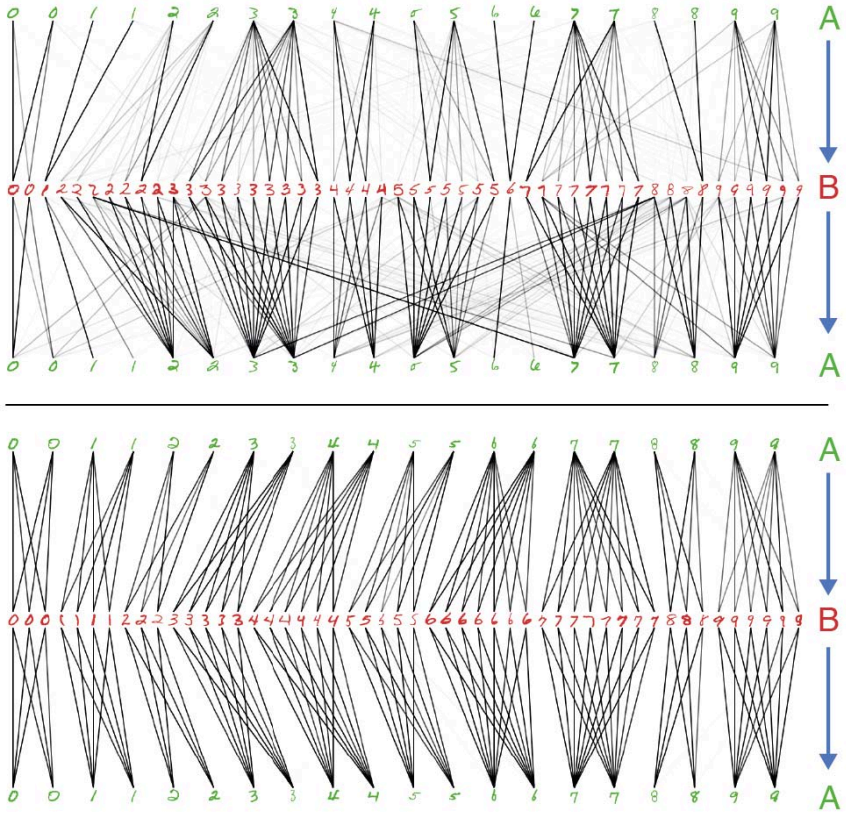


Figure 2. Evolution of associations. Top: in the beginning of training, after a few iterations. Bottom: after convergence. Green digits are the supervised set (A) and red digits are samples from the unsupervised set (B).

the testing protocol suggested by the data set creators, we do not want to claim state of the art for this experiment but do consider it a promising result. [13] achieved 76.3% following the proposed protocol.

The unlabeled training set contains many other classes and it is interesting to examine the trained net's associations with them. Figure 4 shows the 5 nearest neighbors (cosine distance) for samples from the unlabeled training set. The cosine similarity is shown in the top left corner of each association. Note that these numbers are not softmaxed. Known classes (top two rows) are mostly associated correctly, whereas new classes (bottom two rows) are associated with other classes, yet exposing interesting connections: The fin of a dolphin reminds the net of triangularly shaped objects such as the winglet of an airplane wing. A meerkat looking to the right is associated with a dog looking in the same direction or with a racoon with dark spots around the eyes. Unfortunately, embeddings of classes not present in the labeled training set do not seem to group together well; rather, they tend to be close to known class

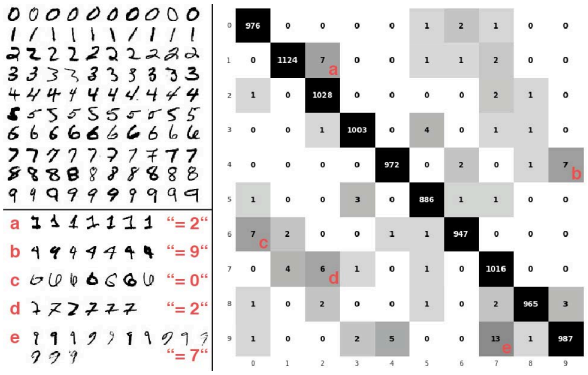


Figure 3. MNIST classification. Top left: All labeled samples that were used for training. Right: Confusion matrix with mistakes that were made. Test error: 0.96%. Bottom left: Misclassified examples from the test.

representations.

Figure 4. Nearest neighbors for samples from the unlabeled training set. The far left column shows the samples, the 5 other columns are the nearest neighbors in terms of cosine distance (which is shown in the top left corners of the pictures).

4.3. SVHN

The Street View House Numbers (SVHN) data set [25] contains digits extracted from house numbers in Google Street View images. We use the format 2 variant where digits are cropped to 32x32 pixels. This variant is similar to MNIST in structure, yet the statistics are a lot more complex and richer in variation. The train and test subsets contain 73,257 and 26,032 digits, respectively.

We performed the same experiments as for MNIST with the following architecture:

$$C(32,3) \rightarrow C(32,3) \rightarrow C(32,3) \rightarrow P(2)$$
$$\rightarrow C(64,3) \rightarrow C(64,3) \rightarrow C(64,3) \rightarrow P(2)$$
$$\rightarrow C(128,3) \rightarrow C(128,3) \rightarrow C(128,3) \rightarrow P(2) \rightarrow FC(128)$$

Data augmentation is achieved by applying random affine transformations and Gaussian blurring to model the variations evident in SVHN.

4.4. Effect of adding unlabeled data

In order to quantify how useful it is to add unlabeled data to the training process with our approach, we trained the same network architecture with different amounts of labeled and unlabeled data. For the case of no unlabeled data, only $\mathcal{L}_{\text{classification}}$ is active. In the other cases where labeled data is present, we optimize $\mathcal{L}_{\text{total}}$. We ran the nets on 10 randomly chosen subsets of the data and report median and standard deviation.

Table 3 shows results on SVHN. We used the (labeled) SVHN training set as data corpus from which we drew randomly chosen subsets as labeled and unlabeled sets. There might be overlaps between both of these sets, which would mean that the reported error rates can be seen as upper bounds.

Let's consider the case of fully supervised training. This corresponds to the far left column in Table 3. Not surprisingly, the more labeled samples are used, the lower the error on the test set gets.

We now add unlabeled data. For a setup with only 20 labeled samples (2 per class), the baseline is an error rate of 81.00% for 0 additional unlabeled samples. Performance deteriorates as more unlabeled samples are added. This setting seems to be pathological: depending on the data set, there is a minimum number of samples required for successful generalization.

In all other scenarios with a greater number of labeled samples, the general pattern we observed is that performance improves with greater amounts of unlabeled data. This indicates that it is indeed possible to boost a network's performance just by adding unlabeled data using the proposed associative learning scheme. For example, in the case of 500 labeled samples, it was possible to decrease the test error by 64.8% (from 17.75% to 6.25%).

A particular case occurs when all data is used in the labeled batch (last row in Table 3): Here, all samples in the unlabeled set are also in the labeled set. This means that the unlabeled set does not contain new information. Nevertheless, employing associative learning with unlabeled data improves the network's performance. $\mathcal{L}_{\text{walker}}$ and $\mathcal{L}_{\text{visit}}$ act as a beneficial regularizer that enforces similarity of embeddings belonging to the same class. This means that associative learning can also help in situations where a purely supervised training scheme has been used, without the need for additional unlabeled data.

4.5. Effect of visit loss

Section 3.1 introduces different losses. We wanted to investigate the effects of our proposed visit loss. To this end,

Method	# labeled samples		
	500	1000	2000
DGN [17]		36.02 (0.10)	
Virtual Adversarial [24]		24.63	
Auxiliary Deep Generative Model [23]		22.86	
Skip Deep Generative Model [23]		16.61 (0.24)	
Imporoved GAN [32]	18.44 (4.8)	8.11 (1.3)	6.16 (0.58)
Imporoved GAN (Ensemble) [32]		5.88 (1.0)	
Mutual Exclusivity + Transform.* [31]	9.62 (1.37)	**4.52 (0.40)**	**3.66 (0.14)**
Ours	**6.25 (0.32)**	5.14 (0.17)	4.60 (0.21)

Table 2. Results of comparable methods on SVHN. Error (%) on the test set (lower is better). Standard deviations in parentheses. *) Results provided by authors.

# labeled samples	# unlabeled samples			
	0	1000	20000	all
20	81.00 (3.01)	81.98 (2.58)	82.15 (1.35)	82.10 (1.91)
100	55.64 (6.54)	39.85 (7.19)	24.31 (7.19)	23.18 (7.41)
500	17.75 (0.65)	12.78 (0.99)	6.61 (0.32)	6.25 (0.32)
1000	10.92 (0.24)	9.10 (0.37)	5.48 (0.34)	5.14 (0.17)
2000	8.25 (0.32)	7.27 (0.43)	4.83 (0.15)	4.60 (0.21)
all	3.09 (0.06)	2.79 (0.02)	2.80 (0.03)	2.69 (0.05)

Table 3. Results on SVHN with different amounts of (total) labeled/unlabeled training data. Error (%) on the test set (lower is better). Standard deviations in parentheses.

we trained networks on different data sets and varied the loss weights for $\mathcal{L}_{\text{visit}}$ keeping the loss weight for $\mathcal{L}_{\text{classification}}$ and $\mathcal{L}_{\text{walker}}$ constant. Table 4 shows the results. Worst performance was obtained with no visit loss. For MNIST, visit loss is crucial for successful training. For SVHN, a moderate loss weight of about 0.25 leads to best performance. If the visit loss weight is too high, the effect seems to be over regularization of the network.. This suggests that the visit loss weight needs to be adapted according to the variance within a data set. If the distributions of samples in the (finitely sized) labeled and unlabeled batches are less similar, the visit loss weight should be lower.

4.6. Domain adaptation

A test for the efficiency of representations is to apply a model to the task of domain adaptation (DA) [29]. The general idea is to train a model on data from a source domain and then adapt it to similar but different data from a target domain.

In the context of neural networks, DA has mostly been achieved by either *fine-tuning* a network on the target domain after training it on the source domain ([36, 15]), or by designing a network with multiple outputs for the respective domains ([5, 38]), sometimes referred to as *dual outputs*.

As a first attempt at DA with associative learning, we tried the following procedure that is a mix of both fine-tuning and dual outputs: We first train a network on the source domain as described in Section 4. Then, we only exchange the unsupervised data set to the target domain data and continue training. Note that here, no labels from the target class are used at all at train time.

As a baseline example, we chose a network trained on SVHN. We fed labeled samples from SVHN (source domain) and unlabeled samples from MNIST (target domain) in the network with the architecture originally used for training on the source domain and fine-tuned it with our association based approach. No data augmentation was applied.

Initially, the network achieved an error of 18.56% on the MNIST test set which we found surprisingly low, considering that the network had not previously seen an MNIST digit. Some SVHN examples have enough similarity to MNIST that the network recognized a considerable amount of handwritten digits.

We then trained the network with both data sources as described above with 0.5 as weight for the visit loss. After

Data set	Visit loss weight 0	0.25	0.5	1
MNIST	5.68 (0.53)	1.17 (0.15)	**0.82** (0.12)	0.85 (0.04)
SVHN	7.91 (0.40)	**6.31** (0.20)	6.32 (0.07)	6.43 (0.26)

Table 4. Effect of visit loss. Error (%) on the resp. test sets (lower is better) for different values of visit loss weight. Reported are the medians of the minimum error rates throughout training with standard deviation in parentheses. Experiments were run with 1,000 randomly chosen labeled samples as supervised data set.

Data	Method	Domain (source → target) SVHN → MNIST
Source only	DA [8]	45.10
	DS [2]	40.8
	Ours	18.56
Adapted	DA [8]	26.15 (42.6%)
	DS [2]	17.3 (58.3%)
	Ours	**0.51 (99.3%)**
Target only	DA [8]	0.58
	DS [2]	0.5
	Ours	0.38

Table 5. Domain adaptation. Errors (%) on the target test sets (lower is better). "Source only" and "target only" refers to training only on the respective data set without domain adaptation. "DA" and "DS" stand for Domain-Adversarial Training and Domain Separation Networks, resp. The numbers in parentheses indicate how much of the gap between lower and upper bounds was covered.

9k iterations the network reached an accuracy of 0.51% on the MNIST test set, which is a higher accuracy than what we reached when training a network with 100 or 1000 labeled samples from MNIST (cf. Section 4.1).

For comparison, [2] has been holding state of the art for domain adaptation employing domain separation networks. Table 5 contrasts their results with ours. Our first tentative training method for DA outperforms traditional methods by a large margin. We therefore conclude that learning by association is a promising training scheme that encourages efficient embeddings. A thorough analysis of the effects of associative learning on domain adaptation could reveal methods to successfully apply our approach to this problem setting at scale.

5. Conclusion

We have proposed a novel semi-supervised training scheme that is fully differentiable and easy to add to existing end-to-end settings. The key idea is to encourage cycle-consistent association chains from embeddings of labeled data to those of unlabeled ones and back. The code is publicly available. Although we have not employed sophisticated network architectures such as ResNet [10] or Inception [35], we achieve competitive results with simple networks trained with the proposed approach. We have demonstrated how adding unlabeled data improves results dramatically, in particular when the number of labeled samples is small, surpassing state of the art for SVHN with 500 labeled samples. In future work, we plan to systematically study the applicability of Associative Learning to the problem of domain adaptation. Investigating the scalability to thousands of classes or maybe even completely different problems such as segmentation will be the subject of future research.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 4

[2] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. *arXiv preprint arXiv:1608.06019*, 2016. 8

[3] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 4

[4] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010. 4

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011. 7

[6] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015. 1, 2

[7] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 766–774, 2014. 1, 2

[8] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016. 8

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014. 1, 2

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1, 8

[11] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner. Early visual concept learning with unsupervised deep learning. *arXiv preprint arXiv:1606.05579*, 2016. 2

[12] G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010. 2

[13] C. Huang, C. Change Loy, and X. Tang. Unsupervised learning of discriminative attributes and visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5175–5184, 2016. 1, 2, 5

[14] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209, 1999. 2

[15] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 7

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[17] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling. Semi-supervised learning with deep generative models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc., 2014. 7

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[19] Q. V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013. 2

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[21] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998. 4

[22] D.-H. Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013. 1, 2

[23] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016. 7

[24] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing by virtual adversarial examples. *arXiv preprint arXiv:1507.00677*, 2015. 7

[25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, Spain, 2011. 6

[26] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 3

[27] M. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the 25th international conference on Machine learning*, pages 792–799. ACM, 2008. 2

[28] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015. 5

[29] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010. 7

[30] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Mutual exclusivity loss for semi-supervised deep learning. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1908–1912. IEEE, 2016. 2

[31] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. *arXiv preprint arXiv:1606.04586*, 2016. 2, 4, 5, 7

[32] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016. 3, 5, 7

[33] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986. 2

[34] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015. 2

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 8

[36] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014. 7

[37] J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012. 2

[38] Z. Yang, R. Salakhutdinov, and W. Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016. 7

[39] J. Zhao, M. Mathieu, R. Goroshin, and Y. Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015. 1, 2

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

4

ASSOCIATIVE DOMAIN ADAPTATION

In Chapter 3 we introduced *associative learning*, a novel training schedule for semi-supervised training of a classification network. In this chapter, which is based on Haeusser, Frerix, Mordvintsev, and Cremers [32], we propose *associative domain adaptation*, an extension that allows for end-to-end domain adaptation with neural networks, the task of inferring class labels for an unlabeled target domain based on the statistical properties of a labeled source domain. Our training scheme follows the paradigm that in order to effectively derive class labels for the target domain, a network should produce statistically domain invariant embeddings, while minimizing the classification error on the labeled source domain. We accomplish this by reinforcing associations between source and target data directly in embedding space. Our method can easily be added to any existing classification network with no structural and almost no computational overhead. We demonstrate the effectiveness of our approach on various benchmarks and achieve state-of-the-art results across the board with a generic convolutional neural network architecture not specifically tuned to the respective tasks. Finally, we show that the proposed association loss produces embeddings that are more effective for domain adaptation compared to methods employing maximum mean discrepancy as a similarity measure in embedding space.

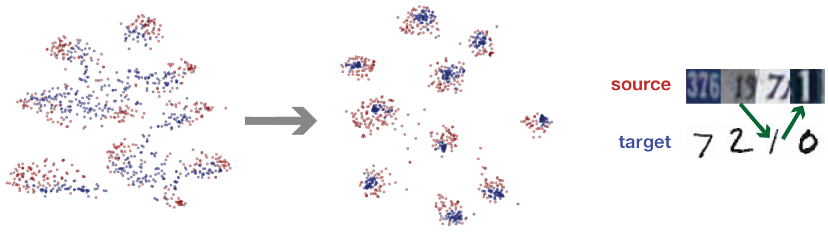


Figure 4.1: Associative domain adaptation - overview

CONTRIBUTIONS OF THE AUTHOR

The author of this dissertation was fully in charge of

- developing the key idea
- planning and carrying out the experiments
- evaluating experiments
- writing substantial parts of the paper

2017 IEEE International Conference on Computer Vision

Associative Domain Adaptation

Philip Haeusser[1,2] haeusser@in.tum.de
Thomas Frerix[1] thomas.frerix@tum.de
Alexander Mordvintsev[2] moralex@google.com
Daniel Cremers[1] cremers@tum.de

[1]Dept. of Informatics, TU Munich [2]Google, Inc.

Abstract

We propose associative domain adaptation*, a novel technique for end-to-end domain adaptation with neural networks, the task of inferring class labels for an unlabeled target domain based on the statistical properties of a labeled source domain. Our training scheme follows the paradigm that in order to effectively derive class labels for the target domain, a network should produce statistically domain invariant embeddings, while minimizing the classification error on the labeled source domain. We accomplish this by reinforcing* associations *between source and target data directly in embedding space. Our method can easily be added to any existing classification network with no structural and almost no computational overhead. We demonstrate the effectiveness of our approach on various benchmarks and achieve state-of-the-art results across the board with a generic convolutional neural network architecture not specifically tuned to the respective tasks. Finally, we show that the proposed association loss produces embeddings that are more effective for domain adaptation compared to methods employing maximum mean discrepancy as a similarity measure in embedding space.*

1. Introduction

Since the publication of LeNet [14] and AlexNet [13], a methodological shift has been observable in the field of computer vision. Deep convolutional neural networks have proved to solve a growing number of problems [28, 7, 29, 27, 6, 17]. On the downside, due to a large amount of model parameters, an equally rapidly growing amount of labeled data is needed for training, such as ImageNet [21], comprising millions of labeled training examples. This data may be costly to obtain or even nonexistent.

In this paper, we focus on an approach to train neural networks with a minimum of labeled data: domain adaptation. We refer to domain adaptation as the task to train a model on labeled data from a source domain while minimizing test error on a target domain, for which no labels are available at training time.

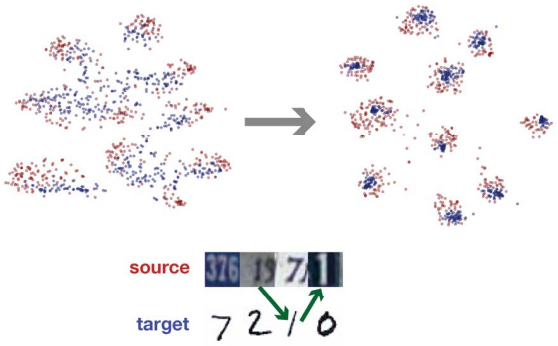


Figure 1: Associative domain adaptation. In order to maximize classification accuracy on an unlabeled target domain, the discrepancy between neural network embeddings of source and target samples (red and blue, respectively) is reduced by an associative loss ($\rightarrow$), while minimizing a classification error on the labeled source domain.

1.1. Domain adaptation

In more formal terms, we consider a source domain $\mathcal{D}_s = \{\boldsymbol{x}_i^s, y_i^s\}_{i=1,\dots,n_s}$ and a target domain $\mathcal{D}_t = \{\boldsymbol{x}_i^t, y_i^t\}_{i=1,\dots,n_t}$. Here, $\boldsymbol{x}_i^s \in \mathbb{R}^{N_s}, \boldsymbol{x}_i^t \in \mathbb{R}^{N_t}$ are the data vectors and $y_i^s \in \mathcal{C}, y_i^t \in \mathcal{C}$ the respective labels, where the target labels $\{y_i^t\}_{i=1,\dots,n_t}$ are *not* available for training. Note that for domain adaption it is assumed that source and target domains are associated with the same label space, while $\mathcal{D}_s$ and $\mathcal{D}_t$ are drawn from distributions $\mathbb{P}_s$ and $\mathbb{P}_t$, which are assumed to be *different*, i.e. the source and target distribution have different joint distributions of data $\boldsymbol{X}$ and labels $\boldsymbol{Y}$, $\mathbb{P}_s(\boldsymbol{X}, \boldsymbol{Y}) \neq \mathbb{P}_t(\boldsymbol{X}, \boldsymbol{Y})$.

The value of domain adaptation has even more increased with generative tools producing synthetic datasets. The idea is compelling: rather than labeling vast amounts of real-world data, one renders a similar but synthetic dataset that is automatically labeled. With an effective method for domain adaptation it becomes possible to train models without the need for one single labeled target example at training time.

2380-7504/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCV.2017.301

In order to combine labeled and unlabeled data for a predictive task, a variety of notions has emerged. To be clear, we explicitly distinguish *domain adaptation* from related approaches. For semi-supervised learning, labeled source data is leveraged by unlabeled target data drawn from the *same* distribution, i.e. $\mathbb{P}_s = \mathbb{P}_t$. In transfer learning, not only source and target domain are drawn from different distributions, also their label spaces are generally different. An example of supervised transfer learning is training a neural network on a source domain and subsequently fine-tuning the model on a labeled target domain for a different task [33, 5].

The problem of domain adaptation was theoretically studied in [2], relating source and target error with a statistical similarity measure of the respective domains. Their results suggest that a good domain adaptation method should be based on features that are as similar as possible for source and target domain (*assimilation*), while reducing the prediction error in the source domain as much as possible (*discrimination*). These effects are opposing each other since source and target domains are drawn from different distributions. This can be formulated as a cost function that consists of two terms:

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \mathcal{L}_{\text{sim}} \ , \tag{1}$$

Here, the classification loss, $\mathcal{L}_{\text{classification}}$ encourages discrimination between different classes, maximizing the margin between clusters of embeddings that belong to the same class. We define the second term as a generic similarity loss $\mathcal{L}_{\text{sim}}$, which enforces statistically similar latent representations.

Intuitively, for similar latent representations of the source and target domain, the target class labels can be more accurately inferred from the labeled source samples.

In the following, we show how previous methods approached this optimization and then propose a new loss for $\mathcal{L}_{\text{sim}}$.

1.2. Related work

Several works have approached the problem of domain adaptation. Here, we mainly focus on methods that are based on deep learning, as these have proved to be powerful learning systems and are closest to our scheme.

The CORAL method [24] explicitly forces the covariance of the target data onto the source data (*assimilation*). The authors then apply supervised training to this transformed source domain with original labels (*discrimination*). This idea is extended to second order statistics of features in deep neural networks in [25].

Building on the idea of adversarial training [10], the authors of [9] propose an architecture in which a class label and a domain label predictor are built on top of a general feature extractor. While the class label predictor is supposed to correctly classify the labeled training examples (*discrimination*), the domain label predictor for all training samples is used in a way to make the feature distributions similar (*assimilation*). The authors of [3] use an adversarial approach to train for similarity in data space instead of feature space. Their training scheme is closer to standard generative adversarial networks [10], however, it does not only condition on noise, but also on an image from the source domain.

Within the paradigm of training for domain invariant features, one popular metric is the maximum mean discrepancy (MMD) [11]. This measure is the distance between the mean embeddings of two probability distributions in a reproducing kernel Hilbert space $\mathcal{H}_k$ with a characteristic kernel k. More precisely, the mean embedding of a distribution $\mathbb{P}$ in $\mathcal{H}_k$ is the unique element $\mu_k(\mathbb{P}) \in \mathcal{H}_k$ such that $\mathbb{E}_{x\sim\mathbb{P}}[f(x)] = \langle f(x), \mu_k(\mathbb{P})\rangle_{\mathcal{H}_k}, \forall f \in \mathcal{H}_k$. The MMD distance between source and target domain then reads $d_{\text{MMD}}(\mathbb{P}_s, \mathbb{P}_t) = ||\mu_k(\mathbb{P}_s) - \mu_k(\mathbb{P}_t)||_{\mathcal{H}_k}$. In practice, this distance is computed via the kernel trick [31], which leads to an algorithm with quadratic runtime in the number of samples. Linear time estimators have previously been proposed [15].

Most works, which explicitly minimize latent feature discrepancy, use MMD in some variant. That is, they use MMD as $\mathcal{L}_{\text{sim}}$ in order to achieve *assimilation* as defined above. The authors of [15] propose the Deep Adaptation Network architecture. Exploiting that learned features transition from general to specific within the network, they train the first layers of a CNN commonly for source and target domain, then train individual task-specific layers while minimizing the multiple kernel maximum mean discrepancies between these layers.

The technique of task-specific but coupled layers is further explored in [20] and [4]. The authors of [20] propose to individually train source and target domains while the network parameters of each layer are regularized to be linear transformations of each other. In order to train for domain invariant features, they minimize the MMD of the embedding layer. On the other hand, the authors of [4] maintain a shared representation of both domains and private representations of each individual domain in their Domain Separation architecture.

As becomes evident in these works, the MMD minimizes domain discrepancy in some abstract space and requires a choice of kernels with appropriate hyperparameters, such as the standard deviation of the Gaussian kernel. In this work, we propose a different loss for $\mathcal{L}_{\text{sim}}$ which is more intuitive in embedding space, less computationally complex and better suitable to obtain effective embeddings.

1.3. Contribution

We propose the association loss $\mathcal{L}_{\text{assoc}}$ as an alternative discrepancy measure ($\mathcal{L}_{\text{sim}}$) within the domain adaptation paradigm described in Section 1.1. The reasoning behind our approach is the following: Ultimately, we want to minimize the classification error on the target domain $\mathcal{D}_t$. This is not directly possible since no labels are available at training time. Therefore, we minimize the classification error on the source domain $\mathcal{D}_s$ as a proxy while enforcing representations of $\mathcal{D}_t$ to have similar statistics to those of $\mathcal{D}_s$. This is accomplished by enforcing *associations* [12] between feature representations of $\mathcal{D}_t$ with those of $\mathcal{D}_s$ that are in the same class. Therefore, in contrast to MMD as $\mathcal{L}_{\text{sim}}$, this approach also leverages knowledge about labels of the source domain and hence avoids unwanted *assimilation* across class clusters. The implementation is simple yet powerful as we show in Section 2. It works with any existing architecture and, unlike most deep learning approaches for domain adaptation, does not introduce a structural and almost no computational overhead. In fact, we used the same generic and simple architecture for *all* our experiments, each of which achieved state-of-the-art results.

In summary, our contributions are:

- A straightforward training schedule for domain adaptation with neural networks.
- An integration of our approach into the prevailing domain adaptation formalism and a detailed comparison with the most commonly used explicit $\mathcal{L}_{\text{sim}}$: the maximum mean discrepancy (MMD).
- A simple implementation that works with arbitrary architectures[1].
- Extensive experiments on various benchmarks for domain adaptation that outperform related deep learning methods.
- A detailed analysis demonstrating that associative domain adaptation results in effective embeddings in terms of classifying target domain samples.

2. Associative domain adaptation

We start from the approach of learning by association [12] which is geared towards semi-supervised training. Labeled and unlabeled data are related by associating their embeddings, i.e. features of a neural network's last layer before the softmax layer. Our work generalizes this approach for domain adaptation. For the new task, we identify labeled data with the source domain and unlabeled data with the target domain. Specifically, for $\boldsymbol{x}_i^s \in \mathcal{D}_s, \boldsymbol{x}_i^t \in \mathcal{D}_t$ and the embedding map $\phi : \mathbb{R}^{N_0} \to \mathbb{R}^{N_{L-1}}$ of an L-layer neural network, denote by $A_i \coloneqq \phi(\boldsymbol{x}_i^s), B_j \coloneqq \phi(\boldsymbol{x}_j^t)$ the respective embeddings of source and target domain. Then, similarity is measured by the embedding vectors' dot product as $M_{ij} = \langle A_i, B_j \rangle$.

If one considers transitions between the parts $(\{A_i\}, \{B_j\})$ of a bipartite graph, the intuition is that transitions are more probable if embeddings are more similar. This is formalized by the transition probability from embedding A_i to embedding B_j:

$$P_{ij}^{ab} = \mathbb{P}(B_j|A_i) \coloneqq \frac{\exp(M_{ij})}{\sum_{j'} \exp(M_{ij'})} \,. \tag{2}$$

The basis of associative similarity is the two-step round-trip probability of an imaginary random walker starting from an embedding A_i of the labeled source domain and returning to another embedding A_j via the (unlabeled) target domain embeddings B,

$$P_{ij}^{aba} \coloneqq \left(P^{ab} P^{ba}\right)_{ij} \,. \tag{3}$$

The authors of [12] observed that higher order round trips do not improve performance. The two-step probabilities are forced to be similar to the uniform distribution over the class labels via a cross-entropy loss term called the *walker loss*,

$$\mathcal{L}_{\text{walker}} \coloneqq H\left(T, P^{aba}\right) \,, \tag{4}$$

where

$$T_{ij} \coloneqq \begin{cases} 1/|A_i| & \text{class}(A_i) = \text{class}(A_j) \\ 0 & \text{else} \end{cases} \tag{5}$$

This means that all association cycles within the same class are forced to have equal probability. The walker loss by itself could be minimized by only visiting target samples that are easily associated, skipping difficult examples. This would lead to poor generalization to the target domain. Therefore, a regularizer is necessary such that each target sample is visited with equal probability. This is the function of the *visit loss*. It is defined by the cross entropy between the uniform distribution over target samples and the probability of visiting some target sample starting in any source sample,

$$\mathcal{L}_{\text{visit}} \coloneqq H(V, P^{\text{visit}}) \,, \tag{6}$$

where

$$P_j^{\text{visit}} \coloneqq \sum_{\boldsymbol{x}_i \in \mathcal{D}_s} P_{ij}^{ab}, \quad V_j \coloneqq \frac{1}{|B|} \,. \tag{7}$$

Note that this formulation assumes that the class distribution is the same for source and target domain. If this is not the case, using a low weight for $\mathcal{L}_{\text{visit}}$ may yield better results.

[1] https://git.io/vyzrl

Together, these terms form a loss that enforces associations between similar embeddings of both domains,

$$\mathcal{L}_{\text{assoc}} = \beta_1 \mathcal{L}_{\text{walker}} + \beta_2 \mathcal{L}_{\text{visit}} \ , \tag{8}$$

where β_i is a weight factor. At the same time, the network is trained to minimize the prediction error on the labeled source data via a softmax cross-entropy loss term, $\mathcal{L}_{\text{classification}}$.

The overall neural network loss for our training scheme is given by

$$\mathcal{L} = \mathcal{L}_{\text{classification}} + \alpha \mathcal{L}_{\text{assoc}} \ . \tag{9}$$

We want to emphasize once more the essential motivation for our approach: The association loss enforces similar embeddings (*assimilation*) for the source and target samples, while the classification loss minimizes the prediction error of the source data (*discrimination*). Without $\mathcal{L}_{\text{assoc}}$, we have the case of a neural network that is trained conventionally [13] on the source domain only. As we show in this work, the (scheduled) addition of $\mathcal{L}_{\text{assoc}}$ during training allows to incorporate unlabeled data from a different domain improving the effectiveness of embeddings for classification. Adding $\mathcal{L}_{\text{assoc}}$ enables an arbitrary neural network to be trained for domain adaptation. The neural network learning algorithm is then able to model the shift in distribution between source and target domain. More formally, if $\mathcal{L}_{\text{assoc}}$ is minimized, *associated* embeddings from both source and target domain become more similar in terms of their dot product.

In contrast to MMD, $\mathcal{L}_{\text{assoc}}$ incorporates knowledge about source domain classes and hence prevents the case that source and target domain embeddings are statistically similar, but not class discriminative. We demonstrate this experimentally in Section 3.4.

We emphasize that not every semi-supervised training method can be adapted for domain adaptation in this manner. It is necessary that the method explicitly models the shift between the source and target distributions, in order to reduce the discrepancy between both domains, which is accomplished by $\mathcal{L}_{\text{assoc}}$.

In this respect, associative domain adaptation parallels the approaches mentioned in Section 1.2. As we demonstrate experimentally in the next section, $\mathcal{L}_{\text{assoc}}$ is employed as a compact, intuitive and effective training signal for *assimilation* yielding superior performance on all tested benchmarks.

3. Experiments

3.1. Domain adaptation benchmarks

In order to evaluate and compare our method, we chose common domain adaptation tasks, for which previous results are reported. Examples for the respective datasets are shown in Table 1.

Table 1: Dataset samples for our domain adaptation tasks. For three randomly chosen classes, the first row depicts a source sample, the second row a target sample. The datasets vary in difficulty due to differences in color space, variance of transformation or number of classes.

MNIST → MNIST-M We used the MNIST [14] dataset as labeled source and generated the unlabeled MNIST-M target as described in [9]. Background patches from the color photo BSDS500 dataset [1] were randomly extracted. Then the absolute value of the difference of each color channel with the MNIST image was taken. This yields a color image, which can be easily identified by a human, but is significantly more difficult for a machine compared to MNIST due to two additional color channels and more nuanced noise. The single channel of the MNIST images was replicated three times to match those of the MNIST-M images (RGB). The image size is 28×28 pixels. This is the only setting where we used data augmentation: We randomly inverted MNIST images since they are always white on black, unlike MNIST-M.

Synth → SVHN The Street View House Numbers (SVHN) dataset [19] contains house number signs extracted from Google Street View. We used the variant *Format 2* where images (32×32 pixels) are already cropped. Still, multiple digits can appear in one image. As a labeled source domain we use the Synthetic Digits dataset provided by the authors of [9], which expresses a varying number of fonts

and properties (background, orientation, position, stroke color, blur) that aim to mimic the distribution in SVHN.

SVHN → MNIST MNIST images were resized with bilinear interpolation to 32×32 pixels and extended to three channels in order to match the shape of SVHN.

Synthetic Signs → GTSRB The Synthetic Signs dataset was provided by the authors of [18] and consists of 100,000 images that were generated by taking common street signs from Wikipedia and applying various artificial transformations. The German Traffic Signs Recognition Benchmark (GTSRB) [23] provides 39,209 (training set) and 12,630 (test set) cropped images of German traffic signs. The images vary in size and were resized with bilinear interpolation to match the Synthetic Signs images' size of 40×40 pixels. Both datasets contain images from 43 different classes.

3.2. Training setup

3.2.1 Associative domain adaptation

Our formulation of *associative domain adaptation* is implemented[2] as a custom loss function that can be added to any existing neural network architecture. Results obtained by neural network learning algorithms often highly depend on the complexity of a specifically tuned architecture. Since we wanted to make the effect of our approach as transparent as possible, we chose the following generic convolutional neural network architecture for *all* our experiments:

$$\begin{aligned} & C(32,3) \rightarrow C(32,3) \rightarrow P(2) \\ \rightarrow\ & C(64,3) \rightarrow C(64,3) \rightarrow P(2) \\ \rightarrow\ & C(128,3) \rightarrow C(128,3) \rightarrow P(2) \rightarrow FC(128) \end{aligned}$$

Here, $C(n,k)$ stands for a convolutional layer with n kernels of size $k \times k$ and stride 1. $P(k)$ denotes a max-pooling layer with window size $k \times k$ and stride 1. $FC(n)$ is a fully connected layer with n output units. The size of the embeddings is 128. An additional fully connected layer maps these embeddings to logits, which are the input to a softmax cross-entropy loss for classification, $\mathcal{L}_{\text{classification}}$.

The detailed hyperparameters for each experiment can be found in the supplementary material. The most important hyperparameters are the following:

Learning rate We chose the same initial learning rate ($\tau = 1e^{-4}$) for all experiments, which was reduced by a factor of 0.33 in the last third of the training time. All trainings converged in less than 20k iterations.

Mini-batch sizes It is important to ensure that a mini-batch represents all classes sufficiently, in order not to introduce a bias. For the labeled mini-batch, we explicitly

[2] https://git.io/vyzrl

Method	Domains (source → target)			
	MNIST → MNIST-M	Syn. Digits → SVHN	SVHN → MNIST	Syn. Signs → GTSRB
Transf. Repr. [22]	13.30	-	21.20	-
SA [8]	43.10	13.56	40.68	18.35
CORAL [24]	42.30	14.80	36.90	13.10
ADDA [30]	-	-	24.00	-
DANN [9]	23.33 (55.87 %)	8.91 (79.67 %)	26.15 (42.57 %)	11.35 (46.39 %)
DSN w/ DANN [3]	16.80 (63.18 %)	8.80 (78.95 %)	17.30 (58.31 %)	6.90 (54.42 %)
DSN w/ MMD [3]	19.50 (56.77 %)	11.50 (31.58 %)	27.80 (32.26 %)	7.40 (51.02 %)
MMD [15]	23.10	12.00	28.90	8.90
DA_{MMD}	22.90	19.14	28.48	10.69
Ours (DA_{assoc} fixed params†)	**10.47 ± 0.28**	8.70 ± 0.2	4.32 ± 1.54	17.20 ± 1.32
Ours (DA_{assoc})	10.53 (85.94 %)	**8.14 (87.78 %)**	**2.40 (93.71 %)**	**2.34 (81.23)**
Source only	35.96	15.68	30.71	4.59
Target only	6.37	7.09	0.50	1.82

Table 2: Domain adaptation. Errors (%) on the target test sets (lower is better). *Source only* and *target only* refer to training only on the respective dataset (supervisedly [12], without domain adaptation) and evaluating on the target dataset. In the DA_{MMD} setting, we replaced $\mathcal{L}_{\text{assoc}}$ with MMD. The metric *coverage* is reported in parentheses, where available (cf. Section 3.3). We used the same network architecture for all our experiments and achieve state of the art results on all benchmarks. The row "DA_{assoc} fixed params†" reports results from 10 runs (± standard deviation) with an arbitrary choice of fixed hyper parameters (β_2 = 0.5, delay = 500 steps and batch size = 100) for all four domain pairs. The row below shows our results after individual hyper parameter optimization. No labels of the target domain were used at training time.

sample a number of examples per class. For the unlabeled mini-batch we chose the same overall size as for the labeled one, usually around 10-100 times the number of classes.

Loss weights The only loss weight that we actively chose is the one for $\mathcal{L}_{\text{visit}}$, β_2. As was shown in [12], this loss acts as a regularizer. Since it assumes the same class distribution on both domains, the weight needs to be lowered if the assumption does not hold. We experimentally chose a suitable weight.

Delay of $\mathcal{L}_{\text{assoc}}$ We observed that convergence is faster if we first train the network only with the classification loss, $\mathcal{L}_{\text{classification}}$, and then add the association loss, $\mathcal{L}_{\text{assoc}}$, after a number of iterations. This is implemented by defining α (Equation 8) as a step function. This procedure is intuitive, as the transfer of label information from source to target domain is most effective when the network has already learned some class structure and the embeddings are not random anymore.

Hyper parameter tuning We are aware that hyper parameter tuning can sometimes obscure the actual effect of a proposed method. In particular, we want to discuss the effect of small batch sizes on our algorithm. For the association loss to work properly, all classes must be represented in a mini-batch, which places a restriction on small batch sizes, when the number of classes is large. To further investigate this hyperparameter we ran the same architecture with an arbitrary choice of fixed hyper parameters and smaller batch size ($\beta_2 = 0.5$, delay = 500 steps and batch size = 100) for all four domain pairs and report the mean and standard deviation of 10 runs in the row "DA_{assoc} fixed params†". In all cases except for the traffic signs, these runs outperform previous methods. The traffic sign setup is special because there are $4.3\times$ more classes and with larger batches more classes are expected to be present in the unlabeled batch. When we removed the batch size constraint, we achieved a test error of 6.55 ± 0.59, which outperforms state of the art for the traffic signs.

Hardware All experiments were carried out on an NVIDIA Titan X (Pascal). Each experiment took less than 120 minutes until convergence.

3.2.2 Domain adaptation with MMD

In order to compare our setup and the proposed $\mathcal{L}_{\text{assoc}}$, we additionally ran all experiments described above with MMD instead of $\mathcal{L}_{\text{assoc}}$. We performed the same hyperparameter search for α and report the respectively best test errors. We used the open source implementation including hyperparameters from [26]. This setup is referred to as DA_{MMD}.

3.3. Evaluation

All reported test errors are evaluated on the target domain. To assess the quality of domain adaptation, we provide results trained on source and target only (SO and TO, respectively) as in [12], for associative domain adaptation (DA_{assoc}) and for the same architecture with MMD instead of $\mathcal{L}_{\text{assoc}}$. Besides the absolute accuracy, an informative metric is *coverage* of the gap between TO and SO by DA,

$$\frac{DA - SO}{TO - SO},$$

as it is a measure of how much label information is successfully transferred from the source to the target domain. In order to assess a method's performance on domain adaptation, one should always consider both coverage and absolute error on the target test set since a high coverage could also stem from poor performance in the SO or TO setting.

Where available, we report the coverage of other methods (with respect to their own performance on SO and TO).

Table 2 shows the results of our experiments. In all four popular domain adaptation settings our method performs best. On average, our approach improves the performance by 87.17 % compared to training on source only (*coverage*). In order to make our results as comparable as possible, we used a generic architecture that was not handcrafted for the respective tasks (cf. Section 3.2).

3.4. Analysis of the embedding quality

As described in Section 1, a good intuition for the formalism of domain adaptation is the following. On the one hand, the latent features should cluster in embedding space, if they belong to the same class (*assimilation*). On the other hand, these clusters should separate well in order to facilitate classification (*discrimination*).

We claim that our proposed $\mathcal{L}_{\text{assoc}}$ is well suited for this task compared with maximum mean discrepancy. We use four points to support this claim:

- t-SNE visualizations show that employing $\mathcal{L}_{\text{assoc}}$ produces embeddings that cluster better compared to MMD.
- $\mathcal{L}_{\text{assoc}}$ simultaneously reduces the maximum mean discrepancy (MMD) in most cases.
- Lower MMD values do not imply lower target test errors in these settings.
- In all cases, the target domain test error of our approach is lower compared to training with an MMD loss.

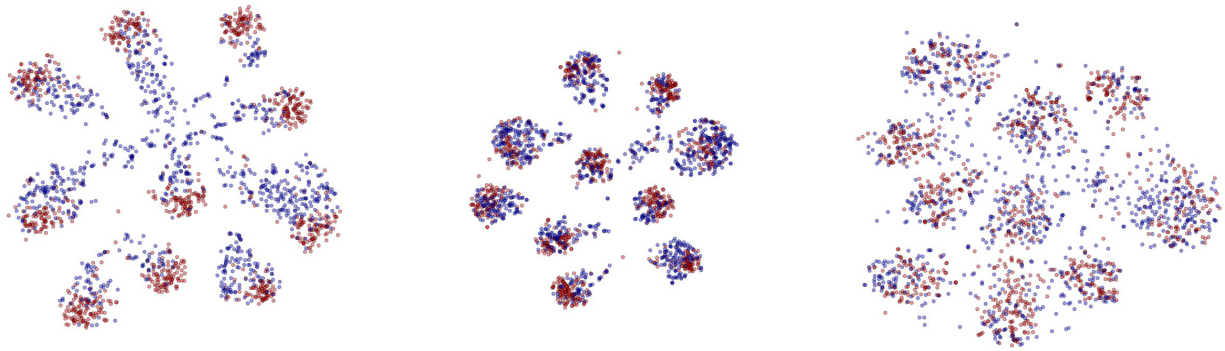


Figure 2: t-SNE embeddings with perplexity 35 of 1,000 test samples for Synthetic Digits (source, red) and SVHN (target, blue). **Left**: After training on *source only*. **Middle**: after training with *associative domain adaptation* ($\mathrm{DA}_{\mathrm{assoc}}$). **Right**: after training with *MMD loss* ($\mathrm{DA}_{\mathrm{MMD}}$). While the target samples are diffuse when embedded with the *source only* trained network, the class label information is successfully inferred after *associative domain adaptation*. When the network is trained with an *MMD loss*, the resulting distributions are similar, but less visibly class discriminative.

	Domains (source → target)			
	MNIST → MNIST-M	Syn. Digits → SVHN	SVHN → MNIST	Syn. Signs → GTSRB
Source only	0.1234 (35.96)	0.1010 (15.68)	0.0739 (30.71)	0.0466 (4.59)
$\mathrm{DA}_{\mathrm{assoc}}$	0.0504 (10.53)	0.0415 (8.14)	0.2112 (2.40)	0.0459 (2.34)
$\mathrm{DA}_{\mathrm{MMD}}$	0.0233 (22.90)	0.0166 (19.29)	0.0404 (34.06)	0.0145 (12.85)

Table 3: Maximum mean discrepancy (MMD) between embeddings of source and target domain, obtained with a network trained supervisedly on source only (SO), for the domain adaptation setting with $\mathcal{L}_{\mathrm{assoc}}$ ($\mathrm{DA}_{\mathrm{assoc}}$) and with an MMD loss ($\mathrm{DA}_{\mathrm{MMD}}$). Numbers in parentheses are test errors on the target domain from Table 2. Associative domain adaptation also reduces the MMD in some cases. Lower MMD values do not correlate with lower test errors. In fact, even though the MMD for training with the associative loss is higher compared with training with the MMD loss, our approach achieves lower test errors.

3.4.1 Qualitative evaluation: t-SNE embeddings

A popular method to visualize high-dimensional data in 2D is t-SNE [16]. We are interested in the distribution of embeddings for source and target domain when we employ our training scheme. Figure 2 shows such visualizations. We always plotted embeddings of the target domain test set. The embeddings are obtained with networks trained semi-supervisedly [12] on the source domain only (SO), with our proposed associative domain adaptation ($\mathrm{DA}_{\mathrm{assoc}}$) and with MMD instead of $\mathcal{L}_{\mathrm{assoc}}$ ($\mathrm{DA}_{\mathrm{MMD}}$, cf. Section 3.2).

In the SO setting, samples from the source domain fall into clusters as expected. Samples from the target domain are more scattered. For $\mathrm{DA}_{\mathrm{assoc}}$, samples from both domains cluster well and become separable. For $\mathrm{DA}_{\mathrm{MMD}}$, the resulting distributions are similar, but not visibly class discriminative.

For completeness, however, we explicitly mention that t-SNE embeddings are obtained via a non-linear, stochastic optimization procedure that depends on the choice of parameters like the perplexity ([16, 32]). We therefore interpret these plots only qualitatively and infer that associative domain adaptation learns consistent embeddings for source and target domain that cluster well with observable margins.

3.4.2 Quantitative evaluation: MMD values

While t-SNE plots provide qualitative insights into the latent feature representation of a learning algorithm, we want to complement this with a quantitative evaluation and compute the discrepancy in embedding space for target and source domains. We estimated the MMD with a Gaussian RBF kernel using the TensorFlow implementation provided by the authors of [26].

The results are shown in Table 3. In parentheses we copied the test accuracies on the respective target domains

from Table 2.

We observe that $\mathrm{DA}_{\mathrm{MMD}}$ yields the lowest maximum mean discrepancy, as expected, since this training setup explicitly minimizes this quantity. At the same time, $\mathrm{DA}_{\mathrm{assoc}}$ also reduces this metric in most cases. Interestingly though, for the setup SVHN $\rightarrow$ MNIST, we actually obtain a particularly high MMD. Nevertheless, the test error of the network trained with $\mathrm{DA}_{\mathrm{assoc}}$ is one of the best results. We ascribe this to the fact that MMD enforces domain invariant feature representations regardless of the source labels, whereas $\mathcal{L}_{\mathrm{assoc}}$ takes into account the labels of associated source samples, resulting in better separation of the clusters and higher similarity within the same class. Consequently, $\mathrm{DA}_{\mathrm{assoc}}$ achieves lower test error on the target domain, which is the actual goal of domain adaptation.

4. Conclusion

We have introduced a novel, intuitive domain adaptation scheme for neural networks termed *associative domain adaptation* that generalizes a recent approach for semi-supervised learning[12] to the domain adaptation setting. The key idea is to optimize a joint loss function combining the classification loss on the source domain with an association loss that imposes consistency of source and target embeddings. The implementation is simple, works with arbitrary architectures in an end-to-end manner and introduces no significant additional computational and structural complexity. We have demonstrated the capabilities of associative domain adaptation on various benchmarks and achieved state-of-the-art results for all our experiments. Finally, we quantitatively and qualitatively examined how well our approach reduces the discrepancy between network embeddings from the source and target domain. We have observed that, compared to explicitly modelling the maximum mean discrepancy as a cost function, the proposed association loss results in embeddings that are more effective for classification in the target domain, the actual goal of domain adaptation.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011. 4

[2] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010. 2

[3] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *arXiv:1612.05424*, 2016. 2, 5

[4] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems 29*, pages 343–351. 2016. 2

[5] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, 2014. 2

[6] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015. 1

[7] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014. 1

[8] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2960–2967, 2013. 5

[9] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(1):2096–2030, 2016. 2, 4, 5

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014. 2

[11] A. Gretton. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012. 2

[12] P. Haeusser, A. Mordvintsev, and D. Cremers. Learning by association - a versatile semi-supervised training method for neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 5, 6, 7, 8

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference of Neural Information Processing Systems*, 2012. 1, 4

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998. 1, 4

[15] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, pages 97–105, 2015. 2, 5

[16] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008. 7

[17] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016. 1

[18] B. Moiseev, A. Konev, A. Chigorin, and A. Konushin. Evaluation of traffic sign recognition methods trained on synthetically generated data. In *International Conference on Ad-*

vanced Concepts for Intelligent Vision Systems, pages 576–583. Springer, 2013. 5

[19] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. 4

[20] A. Rozantsev, M. Salzmann, and P. Fua. Beyond sharing weights for deep domain adaptation. *arXiv:1603.06432*, 2016. 2

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 1

[22] O. Sener, H. O. Song, A. Saxena, and S. Savarese. Learning transferrable representations for unsupervised domain adaptation. In *Advances in Neural Information Processing Systems*, pages 2110–2118, 2016. 5

[23] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011. 5

[24] B. Sun, J. Feng, and K. Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2058–2065, 2016. 2, 5

[25] B. Sun and K. Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision–ECCV 2016 Workshops*, pages 443–450, 2016. 2

[26] D. J. Sutherland, H.-Y. Tung, H. Strathmann, S. De, A. Ramdas, A. Smola, and A. Gretton. Generative models and model criticism via optimized maximum mean discrepancy. *arXiv:1611.04488*, 2016. 6, 7

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 1

[28] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013. 1

[29] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660, 2014. 1

[30] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. *Nips*, 2016. 5

[31] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995. 2

[32] M. Wattenberg, F. Vigas, and I. Johnson. How to use t-sne effectively. *Distill*, 2016. http://distill.pub/2016/misread-tsne. 7

[33] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 27*, 27:1–9, 2014. 2

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

5 ASSOCIATIVE DEEP CLUSTERING

In the previous chapters we have investigated *associative learning* for cases where labels are present. This chapter, which is based on Haeusser, Plapp, Golkov, Aljalbout, and Cremers [33], introduces an extension to the previous training scheme where *no labels* are used at all.

Humans are able to look at a large number of images, find similarities and group images together by an abstract understanding of the content. Researchers have been trying to implement such unsupervised learning schemes for a long time: Given a dataset, find a rule to assign each example to one of k clusters. We propose a novel training schedule for neural networks that facilitates fully unsupervised end-to-end clustering that is direct, i.e. outputs a probability distribution over cluster memberships.

A neural network maps images to embeddings. We introduce centroid variables with the same shape as image embeddings. These variables are jointly trained with the network's parameters. This is achieved by a cost function associating the centroid variables with the embeddings of input images. Finally, an additional layer maps embeddings to logits allowing for the direct estimation of the respective cluster membership. Unlike other methods, this does not require any additional classifier to be trained on the embeddings separately.

The proposed approach achieves state-of-the-art results in unsupervised classification and we provide an extensive ablation study to demonstrate its capabilities.

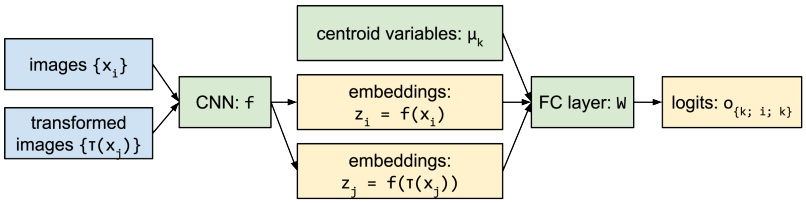


Figure 5.1: Associative deep clustering - overview

CONTRIBUTIONS OF THE AUTHOR

The author of this dissertation was fully in charge of

- developing the key idea
- planning the experiments
- partially evaluating experiments
- writing substantial parts of the paper

Associative Deep Clustering: Training a Classification Network with no Labels

Philip Haeusser [1] **Johannes Plapp** [1] **Vladimir Golkov** [1] **Elie Aljalbout** [1] **Daniel Cremers** [1]

Abstract

Humans are able to look at a large number of images, find similarities and group images together by an abstract understanding of the content. Researchers have been trying to implement such unsupervised learning schemes for a long time: Given a dataset, e.g. images, find a rule to assign each example to one of k clusters. We propose a novel training schedule for neural networks that facilitates fully unsupervised end-to-end clustering that is direct, i.e. outputs a probability distribution over cluster memberships.

A neural network maps images to embeddings. We introduce centroid variables that have the same shape as image embeddings. These variables are jointly trained with the network's parameters. This is achieved by a cost function that associates the centroid variables with the embeddings of input images. Finally, an additional layer maps embeddings to logits allowing for the direct estimation of the respective cluster membership. Unlike other methods, this does not require any additional classifier to be trained on the embeddings in a separate step.

The proposed approach achieves state-of-the-art results in unsupervised classification and we provide an extensive ablation study to demonstrate its capabilities.

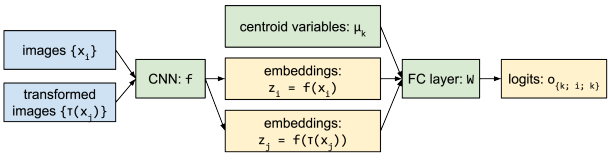


Figure 1. Associative deep clustering. Images (x_i) and transformations of them ($\tau(x_j)$) are sent through a CNN in order to obtain embeddings z. We introduce k centroid variables μ_k that have the same dimensionality as the embeddings. Our proposed loss function simultaneously trains these centroids and the network's parameters along with a mapping from embedding space to a cluster membership distribution.

1. Introduction

1.1. Towards direct deep clustering

Deep neural networks have shown impressive potential on a multitude of computer vision challenges (Szegedy et al., 2013; Eigen et al., 2014; Toshev & Szegedy, 2014; Szegedy et al., 2015; Dosovitskiy et al., 2015; Mayer et al., 2016). A fundamental limitation in many applications is that they traditionally require huge amounts of labeled training data. To circumvent this problem, a plethora of semi-supervised and unsupervised training schemes have been proposed (Doersch et al., 2015; Dosovitskiy et al., 2014; Lee, 2013; Huang et al., 2016). They all aim at reducing the number of labeled data while leveraging large quantities of unlabeled data.

It is an intriguing idea to train a neural network without any labeled data at all by automatically discovering structure in data given as minimal prior knowledge the number of classes. In many real-world applications beyond the scope of academic research, it is desired to discover structures in large unlabeled data sets. When the goal is to separate date into groups, this maneuver is called *clustering*. Deep neural networks are the model of choice when it comes to image understanding. In the past, however, deep neural networks have rarely been trained for clustering directly. A more common approach is *feature learning*. Here, a proxy task is designed to generate a loss signal that can be used to update the model parameters in an entirely unsupervised manner. An exhaustive comparison of previous works is collected in Section 1.2. All these approaches aim at transforming an input image x_i to a representation or embedding z_i that allows for clustering the data. In order to perform this last step, a mapping from embedding space to clusters is necessary, e.g. by training an additional classifier on the features, such as k-means (MacQueen et al., 1967) or an SVM (Suykens & Vandewalle, 1999) in a separate step.

A common problem in unsupervised learning is that there

[1]Department of Informatics, TU Munich, Germany. Correspondence to: Philip Haeusser <haeusser@in.tum.de>.

Under review for ECCV 2018

is no signal that tells the network to cluster different examples from the same class together although they look very different in pixel space. We call this the *blue sky problem* with the pictures of a flying bird and a flying airplane in mind, both of which will contain many blue pixels but just a few others that make the actual distinction. In particular auto-encoder approaches suffer from the blue sky problem as their cost function usually penalizes reconstruction in pixel space and hence favors the encoding of potentially unnecessary information such as the color of the sky.

1.2. Related work

Classical clustering approaches are limited to the original data space and are thus not very effective in high-dimensional spaces with complicated data distributions, such as images. Early deep-learning-based clustering methods first train a feature extractor, and in a separate step apply a clustering algorithm to the features (Goodfellow et al., 2014; Radford et al., 2015; Salimans et al., 2016).

Well-clusterable deep representations are characterized by high within-cluster similarities of points compared to low across-cluster similarities. Some loss functions optimize only one of these two aspects. Particularly, methods that do not encourage across-cluster dissimilarities are at risk of producing worse (or theoretically even trivial) representations/results (Yang et al., 2016a), but some of them nonetheless work well in practice. Other methods avoid trivial representations by using additional techniques unrelated to clustering, such as autoencoder reconstruction loss during pre-training or entire training (Huang et al., 2014; Xie et al., 2016; Yang et al., 2016a; Li et al., 2017).

Deep Embedded Clustering (DEC) (Xie et al., 2016) model simultaneously learns feature representations and cluster assignments. To get a good initialization, DEC needs autoencoder pre-training. Also, the approach has some issues scaling to larger datasets such as STL-10.

Variational Deep Embedding (VaDE) (Zheng et al., 2016) is a generative clustering approach based on variational autoencoders. It achieves significantly more accurate results on small datasets, but does not scale to larger, higher-resolution datasets. For STL-10, it uses a network pre-trained on the dramatically larger Imagenet dataset.

Joint Unsupervised Learning (JULE) of representations and clusters (Yang et al., 2016b) is based on agglomerative clustering. In contrast to our method, JULE's network training procedure alternates between cluster updates and network training. JULE achieves very good results on several datasets. However, its computational and memory requirements are relatively high as indicated by (Hsu & Lin, 2017).

Clustering convolutional neural networks (CCNN) (Hsu & Lin, 2017) predict clustering assignments at the last layer, and a clustering-friendly representation at an intermediate layer. The training loss is the difference between the clustering predictions and the results of k-means applied to the learned representation. Interestingly, CCNN yields good results in practice, despite both the clustering features and the cluster predictions being initialized in random and contradictory ways. CCNN requires running k-means in each iteration.

Deep Embedded Regularized Clustering (DEPICT) (Dizaji et al., 2017) is similar to DEC in that feature representations and cluster assignments are simultaneously learned using a deep network. An additional regularization term is used to balance the cluster assignment probabilities allowing to get rid of the pre-training step. This method achieved a performance comparable to ours on MNIST and FGRC. However, it requires pre-training using the autoencoder reconstruction loss.

In Categorical Generative Adversarial Networks (CatGANs) (Springenberg, 2015), unlike standard GANs, the discriminator learns to separate the data into k categories instead of learning a binary discriminative function. Results on large datasets are not reported. Another approach based on GANs is (Premachandran & Yuille, 2016).

Information-Maximizing Self-Augmented Training (IMSAT) (Hu et al., 2017) is based on Regularized Information Maximization (RIM) (Krause et al., 2010), which learns a probabilistic classifier that maximizes the mutual information between the input and the class assignment. In IMSAT this mutual information is represented by the difference between the marginal and conditional distribution of those values. IMSAT also introduces regularization via self-augmentation. This is achieved via an additional term to the cost function which assures that the generated data point and the original are similarly assigned. For large datasets, IMSAT uses fixed pre-trained network layers.

Several other methods with various quality of results exist (Chen, 2015; Lukic et al., 2016; Wang et al., 2016; Chen et al., 2017; Saito & Tan, 2017; Harchaoui et al., 2017).

In summary, previous methods either do not scale well to large datasets (Xie et al., 2016; Zheng et al., 2016; Springenberg, 2015), have high computational and/or memory cost (Yang et al., 2016b), require a clustering algorithm to be run during or after the training (most methods, e.g. (Yang et al., 2016b; Hsu & Lin, 2017)), are at risk of producing trivial representations, and/or require some labeled data (Hu et al., 2017) or clustering-unrelated losses (for example additional autoencoder loss (Huang et al., 2014; Xie et al., 2016; Yang et al., 2016a; Li et al., 2017; Zheng et al., 2016; Dizaji et al., 2017)) to (pre-)train (parts of) the network. In particular reconstruction losses tend to overestimate the importance of low level features such as colors.

In order to solve these problems, it is desirable to develop new training schemes that tackle the clustering problem as a *direct* end-to-end training of a neural network.

1.3. Contribution

In this paper, we propose *Associative Deep Clustering* as an end-to-end framework that allows to train a neural network directly for clustering. In particular, we introduce *centroid embeddings*: variables that look like embeddings of images but are actually part of the model. They can be optimized and they are used to learn a projection from embedding space to the desired dimensionality, e.g. logits $\in \mathbb{R}^k$ where k is the number of classes. The intuition is that the centroid variables carry over high-level information about the data structure (i.e. cluster centroid embeddings) from iteration to iteration. It makes sense to train a neural network directly for a clustering task, rather than a proxy task (such as reconstruction from embeddings) since the ultimate goal is actually clustering.

To facilitate this, we introduce a cost function consisting of multiple terms which cause clusters to separate while associating similar images. *Associations* (Haeusser et al., 2017b) are made between centroids and image embeddings, and between embeddings of images and their transformations. Unlike previous methods, we use clustering-specific loss terms that allow to directly learn the assignment of an image to a cluster. There is no need for a subsequent training procedure on the embeddings. The output of the network is a cluster membership distribution for a given input image. We demonstrate that this approach is useful for clustering images without any prior knowledge other than the number of classes.

The resulting learned cluster assignments are so good that subsequently re-running a clustering algorithm on the learned embeddings does not further improve the results (unlike e.g. (Yang et al., 2016b)).

To the best of our knowledge, we are the first to introduce a training scheme for direct clustering jointly with network training, as opposed to feature learning approaches where the obtained features need to be clustered by a second algorithm such as k-means (as in most methods), or to methods where the clustering is directly learned but parts of the network are pre-trained and fixed (Hu et al., 2017).

Moreover, unlike most methods, we use *only* clustering-specific and invariance-imposing losses and no clustering-unrelated losses such as autoencoder reconstruction or classification-based pre-training.

In summary, our contributions are:

- We introduce centroid variables that are jointly trained with the network's weights.
- This is facilitated by our clustering cost function that makes *associations* between cluster centroids and image embeddings. No labels are needed at any time. In particular, there is no subsequent clustering step necessary such as k-means.
- We conducted an extensive ablation study demonstrating the effects of our proposed training schedule.
- Our method outperforms the current state of the art on a number of datasets.
- All code is available as an open-source implementation in TensorFlow[1].

2. Associative Deep Clustering

In this section, we describe our setup and the cost function. Figure 2 depicts an overall schematic which will be referenced in the following.

2.1. Associative learning

Recent works have shown that *associations* in embedding space can be used for semi-supervised training and domain adaptation (Haeusser et al., 2017b;a). Both applications require an amount of labeled training data which is fed through a neural network along with unlabeled data. Then, an imaginary walker is sent from embeddings of labeled examples to embeddings of unlabeled examples and back. From this idea, a cost function is constructed that encourages consistent association cycles, meaning that two labeled examples are associated via an unlabeled example with a high probability if the labels match and with a low probability otherwise. More formally: Let A_i and B_j be embeddings of labeled and unlabeled data, respectively. Then a similarity matrix $M_{ij} := A_i \cdot B_j$ can be defined. These similarities can now be transformed into a transition probability matrix by softmaxing M over columns:

$$P_{ij}^{ab} = P(B_j|A_i) := (\text{softmax}_{\text{cols}}(M))_{ij} = \exp(M_{ij}) / \sum_{j'} \exp(M_{ij'}) \quad (1)$$

Analogously, the transition probabilities in the other direction (P^{ba}) are obtained by replacing M with M^T. Finally, the metric of interest is the probability of an association

[1]The code will be made available upon publication of this paper.

cycle from A to B to A:

$$P_{ij}^{aba} := (P^{ab}P^{ba})_{ij} = \sum_k P_{ik}^{ab} P_{kj}^{ba} \tag{2}$$

Since the labels of A are known, it is possible to define a target distribution where inconsistent association cycles (where labels mismatch) have zero probability:

$$T_{ij} := \begin{cases} 1/\#\text{class}(A_i) & \text{class}(A_i) = \text{class}(A_j) \\ 0 & \text{else} \end{cases} \tag{3}$$

Now, the associative loss function becomes $\mathcal{L}_{\text{assoc}}(A, B) := \text{crossentropy}(T_{ij}; P_{ij}^{ab})$. For more details, the reader is kindly referred to (Haeusser et al., 2017b).

2.2. Clustering loss function

In this work, we further develop this setup since there is no labeled batch A. In its stead, we introduce centroid variables μ_k that have the same dimensionality as the embeddings and that have surrogate labels $0, 1, \ldots, k-1$. With them and embeddings of (augmented) images, we can define clustering associations.

We define two associative loss terms:

- $\mathcal{L}_{\text{assoc,c}} := \mathcal{L}_{\text{assoc}}(\mu_k; z_i)$ where associations are made between (labeled) centroid variables μ_k (instead of A) and (unlabeled) image embeddings z_i
- $\mathcal{L}_{\text{assoc,aug}} := \mathcal{L}_{\text{assoc}}(z_i; f(\tau(x_j)))$ where we apply 4 random transformations τ to each image x_j resulting in 4 "augmented" embeddings z_j that share the same surrogate label. The "labeled" batch A then consists of all augmented images, the "unlabeled" batch B of embeddings of the unaugmented images x_i.

For simplicity, we imply a sum over all examples x_i and x_j in the batch. The loss is then divided by the number of summands.

The transformation τ randomly applies cropping and flipping, Gaussian noise, small rotations and changes in brightness, saturation and hue.

All embeddings and centroids are regularized with an L2 loss $\mathcal{L}_{\text{norm}}$ to have norm 1. We chose this value empirically to avoid too small numbers in the 128-dimensional embedding vectors.

A fully-connected layer W projects embeddings (μ_k, z_i and z_j) to logits $o_{k,i,j} \in \mathbb{R}^k$. After a softmax operation, each logit vector entry can be interpreted as the probability of membership in the respective cluster. W is optimized through a standard cross-entropy classification loss $\mathcal{L}_{\text{classification}}$ where the inputs are μ_k and the desired outputs are the surrogate labels of the centroids.

Finally, we define a transformation loss

$$\mathcal{L}_{\text{trafo}} := |1 - f(x_i)^T f(\tau(x_j)) - \text{crossentropy}(o_i, o_j)| \tag{4}$$

Here, we apply τ once and set it once to the identity such that the "augmented" batch contains each image and a transformed version of it. This formulation can be interpreted as a trick to obtain weak labels for examples since the logits yield an estimate for the class membership "to the best knowledge of the network so far". Particularly, this loss serves multiple purposes:

- The logit o_i of an image x_i and the logit o_j of the transformed version $\tau(x_j)$ should be similar for $i = j$.
- Images that the network thinks belong to the same cluster (i.e. their centroid distribution o is similar) should also have similar embeddings, regardless of whether τ was applied.

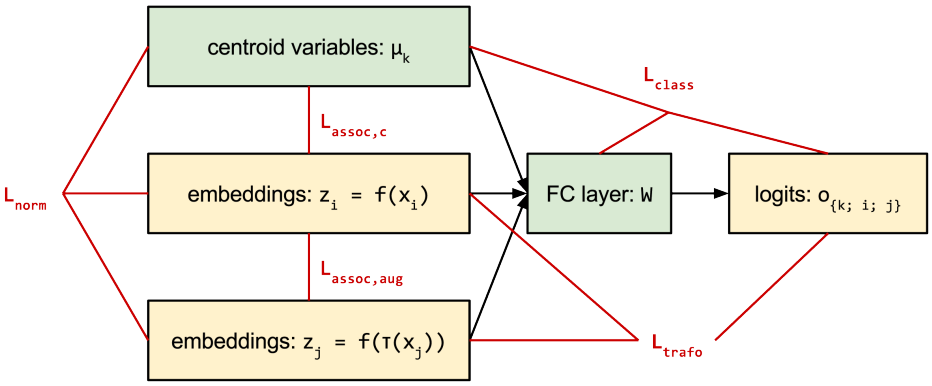


Figure 2. Schematic of the framework and the losses. Green boxes are trainable variables. Yellow boxes are representations of the input. Red lines connect the losses with their inputs. Please find the definitions in Section 2.2

	MNIST	FRGC	SVHN	CIFAR-10	STL-10
k-means on pixels	53.49	24.3	12.5	20.8	22.0
DEC (Xie et al., 2016)	84.30	37.8	-	-	35.9‡†
pre-train+DEC (Hu et al., 2017)	-	-	11.9 (0.4)†	46.9 (0.9)†	78.1 (0.1)†
VaDE (Zheng et al., 2016)	94.46	-	-	-	84.5†
CatGAN (Springenberg, 2015)	95.73	-	-	-	-
JULE (Yang et al., 2016b)	96.4	46.1	-	63.5‡‡	-
DEPICT (Dizaji et al., 2017)	96.5	**47.0**	-	-	-
DEPICT unsupervised*	-	-	18.6 (1.1)	12.4 (0.5)	22.8 (1.5)
IMSAT (Hu et al., 2017)	**98.4 (0.4)**	-	57.3 (3.9) ‡	45.6 (2.9)†	94.1†
IMSAT unsupervised*	-	-	23.2 (0.4)	19.9 (0.2)	24.3 (0.9)
CCNN (Hsu & Lin, 2017)	91.6	-	-	-	-
ours (VCNN): Mean	**98.7 (0.6)**	43.7 (1.9)	37.2 (4.6)	26.7 (2.0)	38.9 (5.9)
ours (VCNN): Best	99.2	46.0	43.4	28.7	41.5
ours (ResNet): Mean	95.4 (2.9)	21.9 (4.9)	**38.6 (4.1)**	**29.3 (1.5)**	**47.8 (2.7)**
ours (ResNet): Best	97.3	29.0	**45.3**	**32.5**	**53.0**
ours (ResNet): K-Means	93.8 (4.5)	24.0 (3.0)	35.4 (3.8)	30.0 (1.8)	47.1 (3.2)

Table 1. Clustering. Accuracy (%) on the test sets (higher is better). Standard deviation in parentheses where available. We report the mean and best of 20 runs for two architectures. For ResNet, we also report the accuracy when k-means is run on top of the obtained embeddings after convergence. †: Using features after pre-training on Imagenet. ‡: Using GIST features. ‡†: Using Histogram-of-oriented-gradients (HOG) features. ‡‡: Using 5k labels to train a classifier on top of the features from clustering. *: For DEPICT and IMSAT, we carried out additional experiments without pre-training, i.e. unsupervisedly and from scratch. We report mean and standard deviation from 10 runs.

- Embeddings of one image and a different image are allowed to be similar if the centroid membership is similar.
- Embeddings are forced to be dissimilar if they do not belong to the same cluster.

The final cost function now becomes

$$\mathcal{L} = \alpha\mathcal{L}_{\text{assoc,c}} \tag{5}$$
$$+ \beta\mathcal{L}_{\text{assoc,aug}} \tag{6}$$
$$+ \gamma\mathcal{L}_{\text{norm}} \tag{7}$$
$$+ \delta\mathcal{L}_{\text{trafo}} \tag{8}$$
$$+ \mathcal{L}_{\text{classification}} \tag{9}$$

with the loss weights $\alpha, \beta, \gamma, \delta$.

In the remainder of this paper, we present an ablation study to demonstrate that the loss terms do in fact support the clustering performance. From this study, we conclude on a set of hyper parameters to be held fixed for all datasets. With these fixed parameters, we finally report clustering scores on various datasets along with a qualitative analysis of the clustering results.

3. Experiments

3.1. Training procedure

For all experiments, we start with a warm-up phase where we set all loss weights to zero except $\beta = 0.9$ and $\gamma = 10^{-5}$. $\mathcal{L}_{\text{assoc,aug}}$ is used to initialize the weights of the network. After 5,000 steps, we find an initialization for μ_k by running k-means on the training embeddings. Then, we activate the other loss weights.

We use the Adam optimizer (Kingma & Ba, 2014) with (beta1=0.8, beta2=0.9) and a learning rate of 0.0008. This learning rate is divided by 3 every 10,000 iterations. Following (Goyal et al., 2017) we also adopt a warmup-phase of 2,000 steps for the learning rate.

We report results on two architectures: A vanilla convolutional neural net from (Haeusser et al., 2017b) (in the following referred to as VCNN) and the commonly used ResNet architecture (He et al., 2016). For VCNN, we adopt the hyper-parameters used in the original work, specifically a mini-batch size of 100 and embedding size 128. For ResNet, we use the architecture specified for CIFAR-10 by (He et al., 2016) for all datasets except STL-10. Due to the larger resolution of this dataset, we adopt the ImageNet variant, and modify it in the following way:

- The kernel size for the first convolutional layer becomes 3, with a stride of 1.
- The subsequent max-pooling layer is set to 2x2
- We reduce the number of filters by a factor of 2.

We use a mini-batch size of 128 images. For ResNet, we use 64-dimensional embedding vectors, as the CIFAR10-variant

only has 64 filters in the last layer. We use a block size of 3 for MNIST, FRGC and STL-10, and 5 for SVHN and CIFAR-10.

The visit weight for both association losses is set to 0.3.

In order to find reasonable loss weights and investigate the importance, we conducted an ablation study which is described in the following section.

3.2. Ablation study

We randomly sampled the loss weights for all previously introduced losses in the range [0; 1] except for the normalization loss weight γ and ran 1,061 experiments on MNIST. Then, we used this clustering model to assign classes to the MNIST test set. Following (Xie et al., 2016), we picked the permutation of class labels that reflects the true class labels best and summarized the respective accuracies in Figure 5. For each point in a plot, we held only the one parameter fixed and averaged all according runs. This explains the error bars which arise from variance of all other parameters. It can still be seen very clearly that each loss has an important contribution to the test accuracy indicating the importance of the respective terms in our cost function.

We carried out similar experiments for other datasets which allowed to choose one set of hyper parameters for all subsequent experiments, which is described in the next section.

In Table 2, we report the results of an ablation study of the different loss terms. It becomes evident that the contribution of each individual loss term is crucial.

3.3. Clustering performance

From the previous section, we chose the following hyper parameters to hold fixed for all datasets:

$$\alpha = 1; \beta = 0.9; \gamma = 10^{-5}; \delta = 2 \times 10^{-5}$$

3.4. Evaluation protocol

Following (Xie et al., 2016), we set k to be the number of ground-truth classes in each dataset, and evaluate the clustering performance using the unsupervised clustering accuracy (ACC) metric. For every dataset, we run our proposed algorithm 10 times, and report multiple statistics:

- Mean and standard deviation of all clustering accuracies.
- The maximum clustering accuracy of all runs.

3.5. Datasets

We evaluate our algorithm on the following, widely-used image datasets:

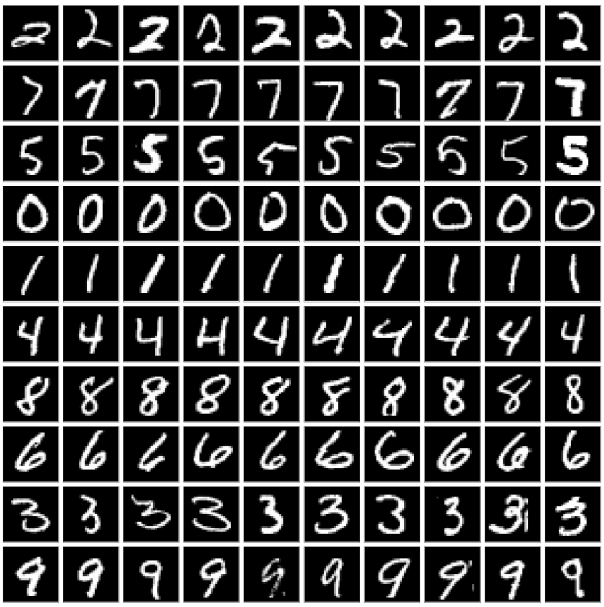


Figure 3. Clustering examples from MNIST. Each row contains the examples with the highest probability to belong to the respective cluster.

MNIST (LeCun, 1998) is a benchmark containing 70,000 handwritten digits. We use all images from the training set without their labels. Following (Haeusser et al., 2017b), we set the visit weight to 0.8. We also use 1,000 warm-up steps.

For **FRGC**, we follow the protocol introduced in (Yang et al., 2016b), and use the 20 selected subsets, providing a total of 2,462 face images. They have been cropped to 32×32px images. We use a visit weight of 0.1 and 1,000 warm-up steps.

SVHN (Netzer et al., 2011) contains digits extracted from house numbers in Google Street View images. We use the training set combined with 150,000 images from the additional, unlabeled set for training.

CIFAR-10 (Krizhevsky & Hinton, 2009) contains tiny images of ten different object classes. Here, we use all images of the training set.

STL-10 (Coates et al., 2011) is similar to CIFAR-10 in that it also has 10 object classes, but at a significantly higher resolution (96×96px). We use all 5,000 images of the training set for our algorithm. We do not use the unlabeled set, as it contains images of objects of other classes that are not in the training set. We randomly crop images to 64×64px during training, and use a center crop of this size for evaluation.

Table 1 summarizes the results. We achieve state-of-the-art results on MNIST, SVHN, CIFAR-10 and STL-10.

Figure 4. Clustering examples from STL-10. Each row contains the examples with the highest probability to belong to the respective cluster.

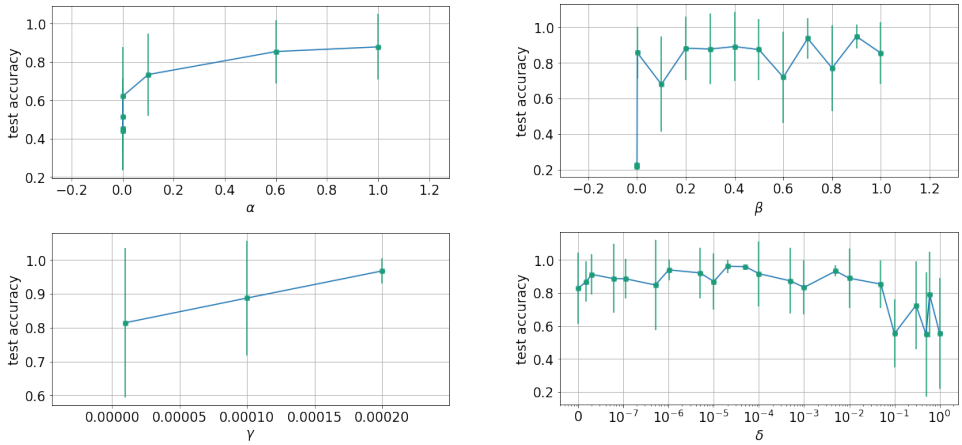


Figure 5. Ablation study for different hyper-parameters on MNIST.

Active loss terms:	$\mathcal{L}_{\text{assoc,aug}}$	$\mathcal{L}_{\text{assoc,c}} + \mathcal{L}_{\text{norm}}$	$\mathcal{L}_{\text{assoc,c}} + \mathcal{L}_{\text{norm}} + \mathcal{L}_{\text{assoc,aug}}$	all (cf. Table 1)
MNIST VCNN	27.6 (3.8) / 85.3 (6.4)	66.7 (8.7)	97.0 (3.8)	98.7 (0.6)
CIFAR-10 ResNet	19.4 (0.5) / 30.2 (1.2)	17.8 (0.8)	27.1 (0.9)	29.3 (1.5)
STL-10 ResNet	19.5 (2.7) / 42.0 (1.3)	16.2 (1.1)	45.6 (3.1)	47.8 (2.7)

Table 2. Ablation study for different loss terms. We report the mean classification test error mean and standard deviation of 10 runs. In the first column, as no centroids are trained, we report scores using direct classification (left), and using k-means on embeddings after the warm-up phase (right). In general, all loss terms contribute to the final results. Also, especially on more complex datasets such as STL-10, associations with augmented samples are important.

3.6. Qualitative analysis

Figure 3 and Figure 4 show images from the MNIST and STL-10 test set, respectively. The examples shown have the highest probability to belong to the respective clusters (logit argmax). The MNIST examples reveal that the network has learned to cluster hand-written digits well while generalizing to different ways how digits can be written. For example, *2* can be written with a little loop. Some examples of *8* have a closed loop, some don't. The network does not make a difference here which shows that the proposed training scheme does learn a high-level abstraction.

Analogously, for STL-10, nearly all images are clustered correctly, despite a broad variance in colors and shapes. It is interesting to see that there is no bird among the top-scoring samples of the *airplane*-cluster, and all birds are correctly clustered even if they are in front of a blue background. This demonstrates that our proposed algorithm does not solely rely on low-level features such as color (cf. the *blue sky problem*) but actually finds common patterns based on more complex similarities.

4. Conclusion

We have introduced *Associative Deep Clustering* as a novel, direct clustering algorithm for deep neural networks. The central idea is to jointly train centroid variables with the network's weights by using a clustering cost function. No labels are needed at any time and our approach does not require subsequent clustering such as many feature learning schemes. The importance of the loss terms were demonstrated in an ablation study and the effectiveness of the training schedule is reflected in state-of-the-art results in classification. A qualitative investigation suggests that our method is able to successfully discover structure in image data even when there is high intra-class variation. In clustering, there is no absolute right or wrong - multiple solutions can be valid, depending on the categories that a human introduces. We believe, however, that our formulation is applicable to many real world problems and that the simple implementation will hopefully inspire many future works.

References

Chen, Dongdong, Lv, Jiancheng, and Yi, Zhang. Unsupervised multi-manifold clustering by learning deep representation. In *Workshops at the 31th AAAI conference on artificial intelligence (AAAI)*, pp. 385–391, 2017. 2

Chen, Gang. Deep learning with nonparametric clustering. *arXiv preprint arXiv:1501.03084*, 2015. 2

Coates, Adam, Ng, Andrew, and Lee, Honglak. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011. 6

Dizaji, Kamran Ghasedi, Herandi, Amirhossein, and Huang, Heng. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. *arXiv preprint arXiv:1704.06327*, 2017. 2, 5

Doersch, Carl, Gupta, Abhinav, and Efros, Alexei A. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015. 1

Dosovitskiy, Alexey, Springenberg, Jost Tobias, Riedmiller, Martin, and Brox, Thomas. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 766–774, 2014. 1

Dosovitskiy, Alexey, Fischer, Philipp, Ilg, Eddy, Hausser, Philip, Hazirbas, Caner, Golkov, Vladimir, van der Smagt, Patrick, Cremers, Daniel, and Brox, Thomas. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766, 2015. 1

Eigen, David, Puhrsch, Christian, and Fergus, Rob. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pp. 2366–2374, 2014. 1

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014. 2

Goyal, Priya, Dollár, Piotr, Girshick, Ross, Noordhuis, Pieter, Wesolowski, Lukasz, Kyrola, Aapo, Tulloch, Andrew, Jia, Yangqing, and He, Kaiming. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5

Haeusser, P., Frerix, T., Mordvintsev, A., and Cremers, D. Associative domain adaptation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017a. 3

Haeusser, P., Mordvintsev, A., and Cremers, D. Learning by association - a versatile semi-supervised training method for neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017b. 3, 4, 5, 6

Harchaoui, Warith, Mattei, Pierre-Alexandre, and Bouveyron, Charles. Deep adversarial gaussian mixture autoencoder for clustering. 2017. 2

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In

Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016. 5

Hsu, Chih-Chung and Lin, Chia-Wen. CNN-based joint clustering and representation learning with feature drift compensation for large-scale image data. *arXiv preprint arXiv:1705.07091*, 2017. 2, 5

Hu, Weihua, Miyato, Takeru, Tokui, Seiya, Matsumoto, Eiichi, and Sugiyama, Masashi. Learning discrete representations via information maximizing self augmented training. *arXiv preprint arXiv:1702.08720*, 2017. 2, 3, 5

Huang, Chen, Change Loy, Chen, and Tang, Xiaoou. Unsupervised learning of discriminative attributes and visual representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5175–5184, 2016. 1

Huang, Peihao, Huang, Yan, Wang, Wei, and Wang, Liang. Deep embedding network for clustering. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pp. 1532–1537. IEEE, 2014. 2

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

Krause, Andreas, Perona, Pietro, and Gomes, Ryan G. Discriminative clustering by regularized information maximization. In *Advances in neural information processing systems*, pp. 775–783, 2010. 2

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009. 6

LeCun, Yann. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 6

Lee, Dong-Hyun. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, pp. 2, 2013. 1

Li, Fengfu, Qiao, Hong, Zhang, Bo, and Xi, Xuanyang. Discriminatively boosted image clustering with fully convolutional auto-encoders. *arXiv preprint arXiv:1703.07980*, 2017. 2

Lukic, Yanick, Vogt, Carlo, Dürr, Oliver, and Stadelmann, Thilo. Speaker identification and clustering using convolutional neural networks. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pp. 1–6. IEEE, 2016. 2

MacQueen, James et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pp. 281–297. Oakland, CA, USA., 1967. 1

Mayer, Nikolaus, Ilg, Eddy, Hausser, Philip, Fischer, Philipp, Cremers, Daniel, Dosovitskiy, Alexey, and Brox, Thomas. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4040–4048, 2016. 1

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011. 6

Premachandran, Vittal and Yuille, Alan L. Unsupervised learning using generative adversarial training and clustering. 2016. 2

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 2

Saito, Sean and Tan, Robby T. Neural clustering: Concatenating layers for better projections. 2017. 2

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016. 2

Springenberg, Jost Tobias. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015. 2, 5

Suykens, Johan AK and Vandewalle, Joos. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999. 1

Szegedy, Christian, Toshev, Alexander, and Erhan, Dumitru. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pp. 2553–2561, 2013. 1

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015. 1

Toshev, Alexander and Szegedy, Christian. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1653–1660, 2014. 1

Wang, Zhangyang, Chang, Shiyu, Zhou, Jiayu, Wang, Meng, and Huang, Thomas S. Learning a task-specific deep architecture for clustering. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 369–377. SIAM, 2016. 2

Xie, Junyuan, Girshick, Ross, and Farhadi, Ali. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pp. 478–487, 2016. 2, 5, 6

Yang, Bo, Fu, Xiao, Sidiropoulos, Nicholas D, and Hong, Mingyi. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *arXiv preprint arXiv:1610.04794*, 2016a. 2

Yang, Jianwei, Parikh, Devi, and Batra, Dhruv. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5147–5156, 2016b. 2, 3, 5, 6

Zheng, Yin, Tan, Huachun, Tang, Bangsheng, Zhou, Hanning, et al. Variational deep embedding: A generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016. 2, 5

Part III

CONCLUSION

6 DISCUSSION AND CONCLUSION

Deep learning models are extremely powerful tools that have revolutionized not only computer vision research [58]. One of the main problems has been the necessity of large amounts of training data.

In this thesis, we have proposed an approach to cope with this problem: *learning by association*. This framework is generally applicable for all types of *categorical* learning problems such as classification or clustering. The capabilities of the proposed method were assessed for semi-supervised, domain adaptation and unsupervised setups.

The key idea is to construct a cost function that "associates" training examples in embedding space. Associations can be made between labeled and unlabeled data (semi-supervised setup) or data from different domains (domain adaptation setup). In a third scenario, we investigated how associative learning can be applied in an unsupervised setup where new model variables ("centroids") are defined and optimized with associations.

With the *associative* training scheme, it is possible to obtain embeddings that are close if they belong to examples from the same class and distant otherwise. We have demonstrated that this novel training schedule allows to leverage unlabeled data in all four setups.

One of the key advantages of *associative learning* is the lean formulation which allows to add it to any existing classification network architecture at almost no performance cost. Modifications that have to be made to such an existing network so as to to integrate this method are of the order of ten lines of code. The training scheme is very intuitive and interpretable. It is easily possible to obtain graphs such as Figure 2 in Chapter 3 where walker probabilities from class examples to others are shown. This can be used to interpret which connections between classes the network has learned.

This work serves as a starting point for further research in interpretable deep learning that requires less annotated training data. It is possible to scale the *associative* training scheme up to problems with a large number of classes. However, a few optimizations would have to be made which was beyond the scope of this work.

Currently, the implementation of *associative learning* is optimized for maximum versatility and performance. The process is broken down to matrix multiplications and nonlinearities which can be carried out fast on a GPU. In particular, the current formulation requires

mini-batches to be large enough such that all classes are sufficiently represented in the unlabeled batch.

If the unlabeled training set is balanced, this can easily be achieved by sampling a large enough number of examples. The main limitation is how much memory is available for mini-batches of a particular size. With a hierarchical class representation [77, 78, 96] or caching mechanisms [89], it could be possible to scale this scheme up to an arbitrary number of classes. Another approach to tackle this problem is to make more use of heavily parallelized training such as described in Chapter 2: the same network is instantiated on multiple machines and gradients are handled centrally. This effectively increases the batch size.

If the unlabeled training set is unbalanced, one strategy could be to pre-sample the unlabeled subset by assigning *weak labels* [41, 76, 95]. These labels could come from predictions of the current network or from another model. The former case would have the disadvantage that if the model has already learned misconceptions, it could become more difficult to overcome them because of this feedback loop. One might have to weigh the *classification loss* higher in the beginning to avoid this. The unlabeled batch should also contain completely randomly samples examples in addition to the "smartly" sampled examples so that potentially wrongly pre-sampled examples do not dominate.

So far, *associative learning* has only been applied to tasks where an image as a whole was classified or clustered. Pixel-wise tasks such as image segmentation [63] could also be integrated in the *associative* framework. Rather than giving the entire image one label, semantically coherent subparts of the image are detected and labeled. by breaking down the image in patches whose embeddings are then associated. As long as a mapping from an input (patch or superpixel) to an embedding can be achieved, an *associative* training scheme can be applied. A naive approach could be to split the image in rectangular patches and then classify them. A more elaborate one could be the application of hypercolums [34] where an embedding for single pixels is obtained by considering activations of all CNN units above that pixel. Integrating such a method in the *associative learning* framework could be used for semi-supervised segmentation for example.

In this thesis we have introduced *associative learning* for semi- supervised classification, domain adaptation and clustering. The general scheme was to obtain useful embeddings for images. A new line of work is to obtain embeddings for images and *text*. Generic text embeddings are successfully used in a variety of tasks. However, they are often learned by capturing the co-occurrence structure from pure text corpora, resulting in limitations of their ability to generalize. In a follow-up work, we explored models that incorporate visual information into the text representation. This was published in the preprint

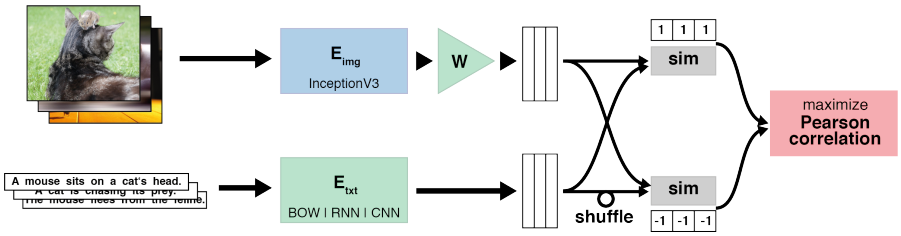


Figure 6.1: An overview of the multimodal association model [55]. Images are fed into a pre-trained CNN (E_{img}). Their representation is then transformed via W to match the dimension of the sentence embeddings. The sentences are encoded via a text embedding model (E_{txt}). Finally, the embeddings are paired in two ways: matching pairs and incorrect pairs. For each set of pairs, we compute the similarity. The loss signal comes from the Pearson correlation, which should be 1 for matching pairs and -1 for incorrect pairs. Only green shaded modules are trained.

Kurach, Gelly, Jastrzebski, Haeusser, Teytaud, Vincent, and Bousquet [55]. Figure 6.1 shows a high-level summary of the approach. Text and image data is encoded by two separate networks. The *association* step again happens in embedding space where text and imagery that belongs together, is *associated*. Based on comprehensive ablation studies, we proposed a conceptually simple, yet well performing architecture. It outperforms previous multimodal approaches on a set of well established benchmarks. We also improved the state-of-the-art results for image-related text datasets, using less data by orders of magnitude.

In summary, *learning by association* is promising for future deep learning research and industrial applications.

In particular in the field of self-driving cars where vast amounts of data from different sensoric modalities need to be processed [42], new methods to reduce the number of labeled training examples will pave the way for technological advancement. Recently, efforts have been made to solve the task of self-driving cars purely based on deep learning [3]. This case particularly requires smart strategies to transfer knowledge acquired under certain geographical or seasonal circumstances to others. Landscapes or traffic signs look differently in different parts of the world and winter scenery has different features than the outside world in summer. Profound domain adaptation methods will be very useful to overcome this problem.

In the domain of automated medical image analysis, manual expert annotation is prohibitively expensive. With the proposed methods, neural networks can be better used in this important area.

The idea of rendering synthetic data to (pre-)train neural networks has become popular recently [19, 66]. Synthetic training data has several benefits: It is possible to generate theoretically infinitely many training examples. Researchers can quantify variances and ensure

that sufficient variety is present in the training datasets. One can explicitly model the real world and implement knowledge about physical principles in the data rendering engine. Then, a domain adaptation setup such as described in Chapter 4 can be used to combine concepts learned in the controlled, synthetic environment to data from the real world. Although, in general, it is tremendously less expensive to build synthetic models than to manually annotate large quantities of data, it would be beneficial to reduce the overall amount of manual work necessary.

With recent advancements in the field of *generative adversarial networks (GANs)* [29], an interesting approach would be to *learn* the underlying generative distribution of the dataset, rather than explicitly modeling it manually. It shall be the subject of further research to determine whether the formulation of *associative learning* can be useful, for example, as part of a cost function for the *discriminator* of a GAN.

It will be interesting to quantify how realistic data for pre-training needs to be. There are several measures for the similarity of image statistics such as *maximum mean discrepancy (MMD)* [30], the *Inception Score* [85] or the *Fréchet distance Distance* [37]. These metrics have been mostly used for the training of GANs in the past but they might be applicable for benchmarking how much effort needs to be put into the generation of *realistic* synthetic training data.

A big topic in deep learning is the quantification of uncertainty. It will be crucial for industrial and scientific applications to develop a unified scheme to estimate how trustworthy a network's prediction is. Gal and Ghahramani [24] propose to use dropout to estimate the model uncertainty. The "dropout method" could easily be integrated in *associative learning* since this training scheme is independent of the network architecture.

With improvements like these, the presented method will hopefully be even more useful for both academic and industrial applications where manually labeled training data is costly or even impossible to obtain. It might get us one step closer to understanding how humans actually learn. Modeling the complex processes in the brain of a learning baby exploring the world, however, will still remain a challenge for a long time.

BIBLIOGRAPHY

[1] Martın Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." In: *arXiv preprint arXiv:1603.04467* (2016).

[2] James Bergstra et al. "Theano: A CPU and GPU math compiler in Python." In: *Proc. 9th Python in Science Conf*. Vol. 1. 2010.

[3] Mariusz Bojarski et al. "End to end learning for self-driving cars." In: *arXiv preprint arXiv:1604.07316* (2016).

[4] Léon Bottou, Frank E Curtis, and Jorge Nocedal. "Optimization methods for large-scale machine learning." In: *arXiv preprint arXiv:1606.04838* (2016).

[5] Arthur E Bryson. "A gradient method for optimizing multi-stage allocation processes." In: *Proc. Harvard Univ. Symposium on digital computers and their applications*. 1961, p. 72.

[6] Arthur Earl Bryson. *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[7] Gavin C Cawley and Nicola LC Talbot. "On over-fitting in model selection and subsequent selection bias in performance evaluation." In: *Journal of Machine Learning Research* 11.Jul (2010), pp. 2079–2107.

[8] Nitesh V Chawla. "Data mining for imbalanced datasets: An overview." In: *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 875–886.

[9] Anna Choromanska et al. "The loss surfaces of multilayer networks." In: *Artificial Intelligence and Statistics*. 2015, pp. 192–204.

[10] Marc Claesen and Bart De Moor. "Hyperparameter search in machine learning." In: *arXiv preprint arXiv:1502.02127* (2015).

[11] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. *Torch: a modular machine learning software library*. Tech. rep. Idiap, 2002.

[12] George Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989), pp. 303–314.

[13] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection." In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[14] Andreas Damianou and Neil Lawrence. "Deep gaussian processes." In: *Artificial Intelligence and Statistics*. 2013, pp. 207–215.

[15] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[16] Jeffrey Dean et al. "Large scale distributed deep networks." In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

[17] Jeff Donahue et al. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition." In: *International Conference in Machine Learning (ICML)*. 2014.

[18] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. "Learning to Generate Chairs with Convolutional Neural Networks." In: *CVPR*. 2015.

[19] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2758–2766.

[20] Stuart Dreyfus. "The numerical solution of variational problems." In: *Journal of Mathematical Analysis and Applications* 5.1 (1962), pp. 30–45.

[21] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. "Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT." In: (2014). pre-print, arXiv:1405.5769v1 [cs.CV].

[22] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[23] Yarin Gal and Zoubin Ghahramani. "Bayesian convolutional neural networks with Bernoulli approximate variational inference." In: *arXiv preprint arXiv:1506.02158* (2015).

[24] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning." In: *international conference on machine learning*. 2016, pp. 1050–1059.

[25] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "A neural algorithm of artistic style." In: *arXiv preprint arXiv:1508.06576* (2015).

[26] Dave Gershgorn. *Despite the hype, nobody is beating Nvidia in AI*. 2017.

[27] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." In: *CVPR*. 2014.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[29] Ian Goodfellow et al. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems 27* (2014), pp. 2672–2680.

[30] Arthur Gretton. "A Kernel Two-Sample Test." In: *Journal of Machine Learning Research* 13 (2012), pp. 723–773.

[31] Philip Haeusser, Alexander Mordvintsev, and Daniel Cremers. "Learning by Association - A Versatile Semi-Supervised Training Method for Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. © 2017 IEEE. Reprinted, with permission from the authors. 2017, pp. 89–98.

[32] Philip Haeusser et al. "Associative Domain Adaptation." In: *International Conference on Computer Vision (ICCV)*. Vol. 2. 5. © 2017 IEEE. Reprinted, with permission from the authors. 2017, p. 6.

[33] Philip Haeusser et al. "Associative Deep Clustering - Training a Classification Network with no Labels." In: *Proceedings of the German Conference on Pattern Recognition (GCPR)*. Reprinted, with permission from the authors. 2018.

[34] Bharath Hariharan et al. "Hypercolumns for Object Segmentation and Fine-grained Localization." In: *CVPR* (2015).

[35] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[36] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[37] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6629–6640.

[38] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[39] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors." In: *arXiv preprint arXiv:1207.0580* (2012).

[40] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural networks* 4.2 (1991), pp. 251–257.

[41] Chen Huang, Chen Change Loy, and Xiaoou Tang. "Unsupervised Learning of Discriminative Attributes and Visual Representations." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5175–5184.

[42] Brody Huval et al. "An empirical evaluation of deep learning on highway driving." In: *arXiv preprint arXiv:1504.01716* (2015).

[43] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[44] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks." In: *CVPR* (2017).

[45] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks." In: *Advances in Neural Information Processing Systems*. 2015, pp. 2017–2025.

[46] Yangqing Jia et al. "Caffe: Convolutional architecture for fast feature embedding." In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[47] Chi Jin et al. "How to Escape Saddle Points Efficiently." In: *arXiv preprint arXiv:1703.00887* (2017).

[48] Norm Jouppi. *Google supercharges machine learning tasks with TPU custom chip*. 2016.

[49] Andrej Karpathy. "Stanford cs231n." In: `http://cs231n.github.io/neural-networks-3` (2018).

[50] Henry J Kelley. "Gradient theory of optimal flight paths." In: *Ars Journal* 30.10 (1960), pp. 947–954.

[51] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. "Optimal whitening and decorrelation." In: *The American Statistician* (2018), pp. 1–6.

[52] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *ICLR*. 2015.

[53] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images." In: (2009).

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Proceedings of the 25th International Conference of Neural Information Processing Systems*. 2012.

[55] Karol Kurach et al. "Better Text Understanding Through Image-To-Text Transfer." In: *arXiv* (2017).

[56] Adrian Lange. "Diagram of a McCulloch-Pitts-Cell." In: *Wikimedia Commons* (2011).

[57] Yann LeCun. "The MNIST database of handwritten digits." In: *http://yann. lecun. com/exdb/mnist/* (1998).

[58] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[59] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551.

[60] Yann LeCun et al. "Handwritten digit recognition with a back-propagation network." In: *Advances in neural information processing systems*. 1990, pp. 396–404.

[61] Yann LeCun et al. "Efficient backprop." In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[62] Yann LeCun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[63] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *CVPR*. 2015.

[64] David G Lowe. "Distinctive image features from scale-invariant keypoints." In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[65] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[66] Nikolaus Mayer et al. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4040–4048.

[67] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[68] Joseph L Mundy, Andrew Zisserman, et al. *Geometric invariance in computer vision*. Vol. 92. MIT press Cambridge, MA, 1992.

[69] Nasser M Nasrabadi. "Pattern recognition and machine learning." In: *Journal of electronic imaging* 16.4 (2007), p. 049901.

[70] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)." In: *Doklady AN USSR*. Vol. 269. 1983, pp. 543–547.

[71] Yuval Netzer et al. "Reading digits in natural images with unsupervised feature learning." In: *NIPS workshop on deep learning and unsupervised feature learning*. Vol. 2011. Granada, Spain. 2011, p. 4.

[72] Jiquan Ngiam et al. "Multimodal deep learning." In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 689–696.

[73] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 427–436.

[74] Maxime Oquab et al. "Learning and transferring mid-level image representations using convolutional neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[75] Adam Paszke et al. *Pytorch*. 2017.

[76] Deepak Pathak et al. "Fully convolutional multi-class multiple instance learning." In: *arXiv preprint arXiv:1412.7144* (2014).

[77] Ravinder Prajapati, Arnav Bhavsar, and Anil Sao. "A hierarchical class-grouping approach, and a study of classification strategies for leaf classification." In: *Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG), 2015 Fifth National Conference on*. IEEE. 2015, pp. 1–4.

[78] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger." In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 6517–6525.

[79] Lawrence Roberts. "Picture coding using pseudo-random noise." In: *IRE Transactions on Information Theory* 8.2 (1962), pp. 145–154.

[80] A Rodriguez and N Sundaram. *Intel and Facebook collaborate to boost Caffe2 performance on Intel CPUs*. 2017.

[81] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[82] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. "Artistic style transfer for videos." In: *German Conference on Pattern Recognition*. Springer. 2016, pp. 26–36.

[83] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), p. 533.

[84] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[85] Tim Salimans et al. "Improved techniques for training gans." In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[86] Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net." In: *arXiv preprint arXiv:1412.6806* (2014).

[87] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[88] Stephen V Stehman. "Selecting and interpreting measures of thematic classification accuracy." In: *Remote sensing of Environment* 62.1 (1997), pp. 77–89.

[89] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. "End-to-end memory networks." In: *Advances in neural information processing systems*. 2015, pp. 2440–2448.

[90] TensorFlow. *Input Pipeline Performance Guide*. 2018.

[91] Jason Van Hulse, Taghi M Khoshgoftaar, and Amri Napolitano. "Experimental perspectives on learning from imbalanced data." In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 935–942.

[92] Sven Wanner, Stephan Meister, and Bastian Goldluecke. "Datasets and benchmarks for densely sampled 4d light fields." In: *VMV*. Citeseer. 2013, pp. 225–226.

[93] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. "How to Use t-SNE Effectively." In: *Distill* (2016). http://distill.pub/2016/misread-tsne.

[94] Paul John Werbos. "Beyond regression: New tools for prediction and analysis in the behavioral sciences." In: *Doctoral Dissertation, Applied Mathematics, Harvard University, MA* (1974).

[95] Jiajun Wu et al. "Deep multiple instance learning for image classification and auto-annotation." In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE. 2015, pp. 3460–3469.

[96] Zhicheng Yan et al. "HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2740–2748.

[97] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *ECCV*. 2014.

[98] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks." In: *arXiv preprint arXiv:1703.10593* (2017).