An Empirical Evaluation of Convolutional and Recurrent Neural Networks for Lip Reading

---

Kevin Heimbach 3825981

dr. A.J. Feelders
prof. dr. A.P.J.M. Siebes

May 31, 2018
Version: Final Version

Universiteit Utrecht

Department of Information and Computing Sciences
Faculty of Science
Graduate School of Natural Sciences

Master Thesis

An Empirical Evaluation of Convolutional and Recurrent Neural Networks for Lip Reading

Kevin Heimbach

1. Reviewer	dr. A.J. Feelders Department of Information and Computing Sciences Universiteit Utrecht
2. Reviewer	prof. dr. A.P.J.M. Siebes Department of Information and Computing Sciences Universiteit Utrecht
Supervisors	dr. Laura Astola

May 31, 2018

Kevin Heimbach

An Empirical Evaluation of Convolutional and Recurrent Neural Networks for Lip Reading

Master Thesis, May 31, 2018

Reviewers: dr. A.J. Feelders and prof. dr. A.P.J.M. Siebes

Supervisors: dr. Laura Astola

Universiteit Utrecht

Graduate School of Natural Sciences

Faculty of Science

Department of Information and Computing Sciences

Princetonplein 5, De Uithof

3584CC and Utrecht

Abstract

The 3DCNN and the LSTM are both suited for video classification because of their ability to take into account temporal information. However, the two models do this in a very distinct manner. The aim of this work is to investigate which of the two models is better suited for automatic lip reading. Moreover, we also tested which model is better suited for transfer learning. We conducted two groups of experiments in this work. The first group consisted of experiments in which the two models were tested under several conditions in which the models were trained from scratch. The second group was conducted to determine which of the two models is better suited for transfer learning. We used a pretrained 3DCNN and LSTM from the first group of experiments to verify whether the accuracy of a model trained on a different dataset improved, compared to when it was trained from scratch. From the first group of experiments, we concluded that the 3DCNN is better suited for automatic lip reading because it achieves a higher test set accuracy than the LSTM. However, the 3DCNN takes a lot longer to train than the LSTM. From the second group of experiments, we can conclude that overall the 3DCNN is better suited for transfer learning. On the basis of all the experiments conducted, we conclude that overall the 3DCNN seems to be better suited for use in automatic lip reading in many different conditions.

Contents

1 Introduction

The number of problems in which automation is being applied has been in steep ascent in recent years. Automating processes that formerly required a human's input is getting increasingly more common. The type of automation we are discussing is the automation of thought, also called artificial intelligence. One reason for the increase of this form of automation is the rapid development of research in artificial intelligence, and more specifically, machine learning. Machine learning is the process of a machine gaining knowledge from past experiences and using this knowledge to form a prediction model which is used to evaluate or predict forthcoming events. Ever since industries began to profit from advances in machine learning, we see an increasing number of applications in everyday life. Because of developments in machine learning, several problems that were extremely difficult to solve a decade ago are seen as relatively uncomplicated today. One example of this is the fact that machines are now able to classify images with greater accuracy than humans [25].

Machine learning is being applied in many fields. In this work, we will focus on the field of Automatic Lip Reading (ALR). Specifically, we will focus on the use of neural networks for the classification of videos in which words are being vocalized. The main focus of this research is to determine what type of neural network is best suited for this purpose.

The structure of this thesis is as follows. In Chapter 2, we will describe the relevance of this work and state the research questions. In Chapter 3, we will provide the background information for automatic lip reading. In this chapter, we will describe the relevance of ALR, the history of ALR, and some of the latest developments in ALR research. In Chapter 4, we will describe neural networks by providing an elaborate description of the mechanisms within several types of neural networks. Moreover, we will describe the history behind neural networks and the latest development in neural network research. Subsequently in Chapter 5, we will discuss the experiments conducted and their results. Finally in Chapter 6, we will discuss the results and their implications more elaborately, and we will discuss their relevance to the research questions stated in Chapter 2.

2 Problem Statement

> *Computer science is no more about computers than astronomy is about telescopes.*
>
> — **Edsger Dijkstra**

In video data, there is a temporal interdependence between successive frames. The decision of what action is taking place in a video depends on the information contained in several frames and the relation between these frames. Because of this, it is often useful to take into account information contained in and spread over many successive frames and the chronological order of events in the frames. This interdependence exists in the short-term, as well as in the long term. In ALR, for example, it is useful to see the exact motions of the mouth in frames that succeed each other directly (short-term interdependence), and, it is useful to see what groups of mouth motions follow one another to decide what phonemes are likely to occur in the vocalization (long-term interdependence). There are multiple ways of capturing the information spread out over many frames. The LSTM and 3DCNN, for example, are very suitable for such problems.

Despite the advances in machine learning research, certain general problems and limitations remain. The wide availability of data does not imply that all classification problems are trivial to solve nowadays. One problem with data could be that it is unlabeled. Training a model to classify images of dogs and cats, for example, is relatively uncomplicated because images of cats and dogs are vastly available. However, some domains of image classification are still difficult to solve. These domains are the ones that are very specific, and in which not a lot of data is readily available. This makes it very difficult, perhaps impossible, to train a model to one of these domains. Gathering a large dataset would be the solution, but this is often not feasible because the process of data collection is a long and expensive process. However, this problem could be overcome with the use of transfer learning. A relatively small dataset could be sufficient to train a model when transfer learning is applied [4, 47, 48].

Another limitation is the availability of computing power. Researchers wish to process increasingly more data and data of higher dimensionality, which forms a limitation because it creates a demand for increasingly more powerful computers. Training a complex model on a large dataset takes a lot of computing power. For example,

the game of Go is seen as the computationally most complex board game. Recently, the world champion of Go has been defeated by Google's AlphaGo, which was a revolutionary accomplishment. One implementation of AlphaGo was on multiple machines, exploiting 1202 CPUs and 176 GPUs in total [61]. This example illustrates how the need for computational resources increases when the problem complexity also increases. It is evident that such resources are not available to the masses of researchers, and thus, they are not able to train a model to solve the most complex problems. For now, solving problems such as the game of Go within a reasonable amount of time is only reserved for those with great financial resources.

Both the LSTM and the 3DCNN are capable of taking into account the interdependence between video frames. We want to investigate which model is better for the application of ALR, an application where the short-term and the long-term interdependency is present. However, the question of which model is 'better' is rather ambiguous. For this reason, it is useful to compare the LSTM and 3DCNN in the context of ALR on several criteria. The main criteria we chose is training time, and more specifically, the difference in training time between the models to achieve a similar accuracy.

Aside from this, we want to investigate the models' suitability for transfer learning. The motivation for this is twofold. First, as stated before, gathering a large dataset for a classification problem is sometimes not feasible and therefore a model has to be trained on a small dataset. Using transfer learning can be beneficial in that case. Second, training on large datasets is computationally expensive and sometimes transfer learning can be of help to achieve good results in a shorter time.

The aim of the current paper is to investigate the differences between the two type of neural networks in the context of automatic lip reading and to investigate their suitability with respect to transfer learning.

2.1 Research Questions

Specifically, the current paper's main focus is the empirical evaluation of the LSTM and 3DCNN in the context of ALR. We aim to train an LSTM and a 3DCNN on an ALR dataset and compare performance and training time, taking into account the computing resources and time available. Thus, the first research question is:

What is the difference in training time between a CNN and an RNN when achieving a similar accuracy in the context of automatic lipreading, given the limitation that we do not know whether our parameter values are optimal?

Subsequently, we want to take the optimized models and explore how well their parameter values transfer to other datasets. We would like to investigate their transfer learning capabilities to a dataset in the same domain and a dataset in a different domain. Thus, the second research question is:

What is the difference in training time between a model that was trained from scratch and a model in which parameter value transfer was applied when achieving a similar accuracy in the context of automatic lipreading, given the limitation that we do not know whether our parameter values are optimal? And between a CNN and an RNN, which benefits most when parameter transfer is applied w.r.t. training time and accuracy?

3 Automatic Lip Reading

> *Innovation distinguishes between a leader and a follower.*
>
> **— Steve Jobs**

"Lip reading, also known as lipreading or speechreading, is a technique of understanding speech by visually interpreting the movements of the lips, face and tongue when sound is not available, relying also on information provided by the context, knowledge of the language, and any residual hearing" [41]. We will refer to machine LR by using the terminology ALR. In this chapter, we will discuss what the relevance is of ALR and how research in ALR can aid humans, and we will discuss the history of ALR. Finally, we will discuss what the current state is of research in ALR and we will discuss its limitations. Throughout this thesis the following terminology will be used: a *phoneme* is a sound that differentiates a word from another word, and a *viseme* is the visual equivalent of a phoneme. However, there is not a one to one correlation from phonemes to visemes. In the English language, for example, multiple phonemes correspond to a single viseme. The phonemes /p/, /b/, and /m/ correspond to the single viseme /p/, and the phonemes /t/, /d/, /n/,and /l/ correspond to the single viseme /t/. The motion of the mouth when these phonemes are vocalized looks very similar, if not identical. Moreover, it is the reason why the words 'men' and 'pet' are very difficult to differentiate by human lip readers.

In modern ALR research, usually the 'overlapped speakers' and 'unseen speakers' accuracy is reported. In the 'overlapped speakers' condition, a single speaker's set of word vocalizations is distributed among the test set *and* the training set. In the 'unseen speaker' condition, speakers in the test set and the speakers in the training set are completely disjoint. Thus, in the unseen speaker condition, there is naturally less resemblance between testing and training samples and this condition will thus be harder for a model to perform well on. Moreover, ALR of previously unseen speakers is also a challenging task, because of the speakers' large variation in lip shapes and because of the lack of large, tracked datasets of visual features [1]. However, we hypothesize that the unseen speaker accuracy correlates more with real-world usability, because the model will have to deal with unseen speakers in a real-world application, by definition. We will see that the first advances in ALR research are made with overlapped speaker conditions and, later, research progresses

to such an extent where it is able to achieve positive performance on unseen speaker conditions as well. Throughout this thesis, we will refer to ALR only capable of word classification on overlapped speakers as 'speaker-dependent' ALR and ALR capable of also word classification on unseen speakers as 'speaker-independent' ALR.

The process of ALR typically consists of three steps: face localization, mouth localization, and classification (Fig. 3.1). There are many methods for the first two steps [53, 72, 58, 59, 77]. In this thesis, however, we will focus on the final step, namely the classification task.

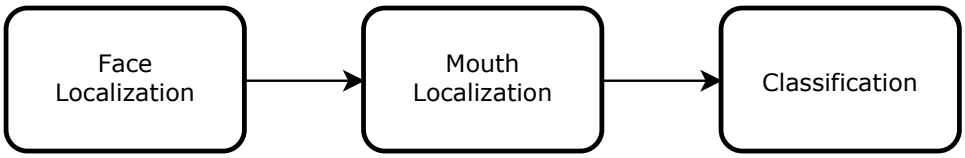

Fig. 3.1: The three steps involved when performing automatic lip reading from recorded video material.

3.1 Relevance

Speech recognition is often seen as a purely auditory process. However, the contribution of visual information to speech recognition cannot be underestimated. In [42], it was shown that human speech recognition accuracy significantly decreases when the word viseme of the speaker shown to the subject does not correlate with the word phoneme of the perceived sound coming from the shown speaker (e.g. 'ga-ga' is being vocalized in a shown video but the phoneme 'pa-pa' is auditorily superimposed on the video). This effect was called the McGurk effect. This effect was even greater in adults of age 18 to 40 than their other two experimental groups that contained children of age 3 to 5 and 7 to 8. Thus, it seems that people learn to rely more on visual information in recognizing speech as they get older. Moreover, children as young as 10 weeks are aware of the correspondence between lip movements and vocal sounds [15]. The underlying mechanism of the McGurk effect bears resemblance to findings in ALR research: the addition of processing visual information in parallel to regular auditory signal processing results in more accurate speech recognition predictions [46, 43, 67].

The usage of ALR can also be of aid in the transcription of video with no, or only low-quality audio available. By the same reasoning of the previous paragraph, ALR can be used in ensemble with auditory speech recognition to transcribe video fragments in the case where only low-quality audio is available. In the case where there is no audio available, it might be possible to rely on ALR alone to transcribe videos. In [18], for example, it was said that ALR was used to aid the analysis of Hitler's

privately recorded silent films, which gave the public an insight to how Hitler was in his private life.

Recently it has been shown that ALR outperforms LR by humans [26]. They used several types of classification conditions: using either appearance (i.e. regular) recordings or only the facial landmark position movements (called 'shape' by the authors) (Fig. 3.2), and testing either untrained or trained humans' LR performance. First, humans' LR performance benefited greatly from being able to see the appearance of a vocalization as opposed to seeing only the shape. Humans' accuracy in the shape-only LR test was 42.9%, as opposed to their 71.6% performance in the appearance condition test. ALR does not follow this trend: 74.3% accuracy in shape-only, and 75.2% in appearance. Second, humans do not seem to benefit from training significantly. Humans were tested on their ALR performance first, which became the measurements for the untrained condition, and afterwards followed a short training. For each of four consecutive days, subjects had to take a training of an hour long to increase their LR competence. However, their pre-train accuracy was 57.7% averages over several word types, while their post-train accuracy on the same tasks was 63.0%.

3.2 History

The current practice in ALR is that usually neural networks are used for research in ALR. However, researchers in the early days had to use other algorithms for their ALR models. This is because neural networks only became scalable to large datasets relatively recent.

One of the first attempts to ALR was made in [19]. The authors recorded 12 landmark position coordinates of the mouth during the articulation of one of 23 English consonant phonemes at 30 frames per second (fps) for one speaker. Of these 12 pairs of x and y positions, they calculated 14 distances between certain

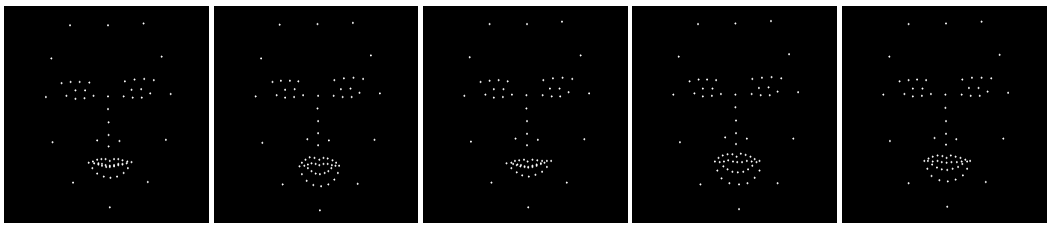

Fig. 3.2: Example frames of the extracted facial landmarks of a subject vocalizing the letter 'B'. The subject was instructed to begin and end the vocalization with a closed mouth. Thus, frame 1 and 5 contain the facial landmarks of a person with its mouth closed. The vocalization takes place in frames 2 to 4.

coordinate pairs as in Fig. 3.3. The speaker articulated each of the 23 phonemes twice: The first set of articulations was considered the training set and the second was considered the test set. The recognition procedure consisted of comparing the 14 distance measurement patterns of each sample in the test set to its equivalent in the training set. For this reason, an equivalence metric was calculated for each sample in the test set and each sample in the training set (i.e. one particular test example was compared to all training examples etc.) and the training example with the highest equivalence was considered the 'predicted' phoneme. This process recognized 9 out of 23 phonemes as the correct phoneme, or 39.1%. However, as stated at the beginning of this chapter, multiple phonemes correspond to the same viseme group. Taking this into account, 18 out of 23 phonemes (or 78.3%) were predicted in the correct 'viseme group'.

Another interesting research conducted in 1988 in [49] took a different approach. They used ALR to accommodate auditory speech recognition and aimed to classify utterances of digits and utterances of different letters separately. The comparison they made was between audio-only, vision-only, and audiovisual classification. From four speakers they extracted a 60 fps mouth region video and converted every frame into a binary image by thresholding the original grayscale frame. Examples of these obtained images can be seen in Fig. 3.4. As in the previously discussed research in [19], the focus here was on whether a recording of a phoneme was classified in the correct viseme group. Moreover, they compared the equivalence of each test sample to all training samples to determine their model's prediction. Their equivalence metric was a distance measure computed based on the comparison of the binary

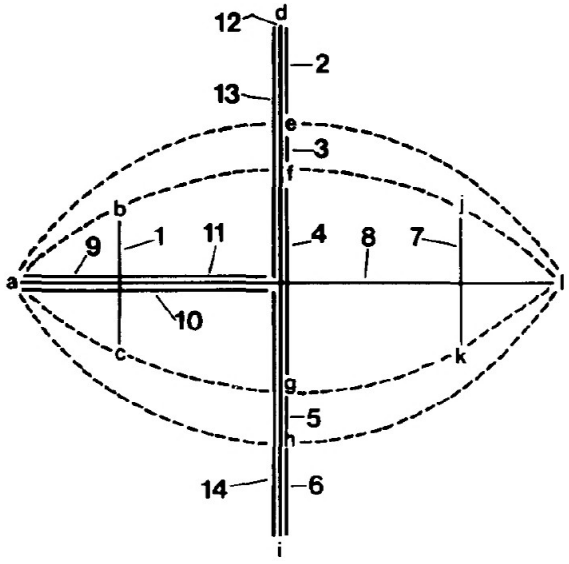

Fig. 3.3: Visualization of the landmark coordinate recordings and the distances calculated. Letters a to l indicate the 12 landmark positions recorded using a reflective substance and numbers 1 to 14 indicate the 14 distances calculated using these 12 landmark coordinates.

mouth images, which is a function of the size of the black region in two images, and the Hamming distance between two images. The Hamming distance of two images is the size of the exclusive disjunctive region between the images (i.e. the region that is black in image 1, but white in image 2, and vice versa). What they found was that audiovisual classification outperformed audio-only and vision-only classification, but also that vision-only outperformed audio-only classification. The audio-only classification was performed by AT&T's isolated word acoustic speech recognizer.

[71] was the first research to use a neural network for ALR. In contrast to today, computing power was scarce and thus images or videos were not given as input as pixel values, but as an array representing features. In this case, the authors extracted 17 features from each frame in the video. Thus, each input was a matrix of length 24, one for each of 24 timesteps, with each entry containing the 17 frame features. The network contained two hidden layers, classified five distinct vocalizations of phonemes, and contained 428 trainable parameters (Fig. 3.5). The performance of ALR was compared to auditory speech recognition and audiovisual speech recognition. What they found was that ALR achieved 54% accuracy, auditory speech recognition 62%, and both combined achieved 72% accuracy. These results, as well as the results from the previously discussed results in [49], strengthen the rationale of why advances in ALR are beneficial to speech recognition as a complement.

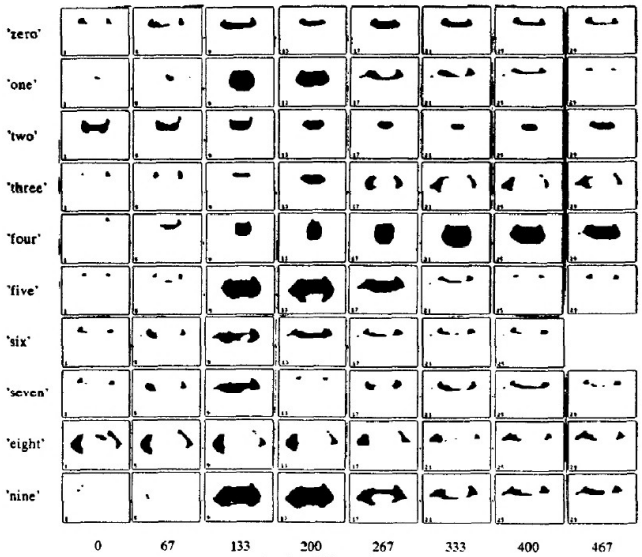

Fig. 3.4: Examples of frames extracted from a speaker. The y-axis indicates which number is vocalized and the x-axis indicates the number of milliseconds passed since the beginning of the vocalization. The first frames are pictured with 3 frames skipped after every frame.

In [31], a leave-one-out approach was taken with regards to testing ALR models. The leave-one-out method in the context of ALR consists of training a model on all but one speaker and testing the model on the speaker that was not included in the training phase. This process is repeated until all speakers have been 'left out' once and the mean accuracy is considered the accuracy achieved. Because this research was mainly concerned with audiovisual LR in auditory noisy environments, they focused on whether taking visual information into account improved speech recognition accuracy in several conditions in which the signal-to-noise ratio was changed. What they found was that audiovisual speech recognition improved audio-only speech recognition between 5% and 12%.

3.3 Current State

The current field of research uses mostly neural networks for ALR classification tasks. In this section, we will provide an overview of the many types of ALR models that have demonstrated state-of-the-art results or models that have recently provided a novelty in the field of ALR. In this section, it is assumed the reader has knowledge of the Convolutional Neural Network (CNN), Spatio-Temporal CNN (3DCNN), Recurrent Neural Network (RNN), and Long-Short Term Memory neural network (LSTM). A thorough explanation of these types of neural networks is

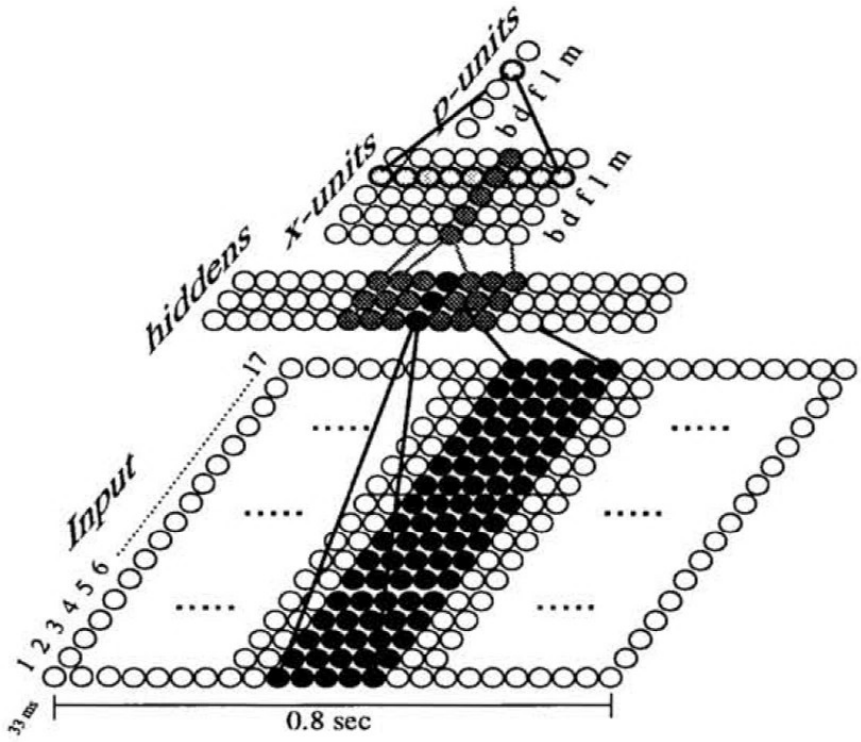

Fig. 3.5: Neural Network ALR

provided in the next chapter. Several datasets are used in ALR research, each designed for a specific classification task. We will first discuss these datasets before moving on to discussing the models that are used today.

A widely used word classification dataset is the Lip Reading in the Wild (LRW) dataset [7]. In this paper, the authors describe their pipeline for fully automatically collecting a large-scale lip-reading dataset from TV broadcasts. With this method, they collected a dataset of more than a million word instances, spread over 500 word classes and spoken by over a thousand people. Each video in the dataset is 1.28s long and the word is vocalized in the middle, as part of a continuous stream of speech. Before this dataset was available, the largest dataset was the GRID dataset [9]. This dataset contained 33,000 different data samples. However, all the sentences were syntactically identical and there was little variation between sentence classes. For example, there were only four different starting words, and the total number of different words was 51. Another widely used dataset, presented in [8], is the Lip Reading Sentences in the Wild (LRS) dataset. This dataset was collected in a similar way as the LRW dataset was collected and it is a significant contribution to the sentence-level ALR field. While the GRID dataset contained 33.000 sentences with a total number of 51 different words, the LRS dataset contained 118,000 sentences with a total number of 17,000 different words.

Another ALR dataset is the MIRACL-VC dataset, presented in 2014 by Rekik et al. [54]. This dataset contains 1500 word data samples and as many sentence data samples, each consisting of 15 speakers vocalizing a total of 10 words and 10 short sentences 10 separate times. Each data sample is accompanied by the depth measurements, recorded at 15fps by a Kinect sensor. The depth measurement consisted of an image with the distance to the camera as pixel values.

Together with the presentation of the LRW dataset in [7], Chung et al. conducted the first experiments on this dataset. They tested several network architectures, namely: 3DCNN with Early Fusion and 3DCNN with Multiple Towers [33], and, Early Fusion and Multiple Towers [35]. They achieve a 61.1% accuracy, by which they show that CNNs are very much suitable for word-level ALR. For future work, they suggested to investigate ALR on profile faces, and combining CNNs with LSTMs using a language model. Research of ALR on profile faces was done in [6] by the same authors and it was concluded that frontal view ALR performed best in comparison to several other angles (30°, 45°, 60°, and 90°). However, taking multiple views of a single vocalization as input increases ALR accuracy [51].

Furthermore, as suggested in [7], A combined 3DCNN with GRU was presented in [2]. It used three layers of combined 3D convolution and maxpooling, followed by 2 layers of bidirectional GRUs, and finally, a Connectionist Temporal Classification

(CTC) classifier, as presented in [23]. CTC is a method by which segmented data can be labeled without the need for labeling the segments explicitly beforehand. It takes as input a sequence of observations and gives as output a classification for each observation. For example, in a video in which a subject is vocalizing a word such as "dog", the classifier will learn by itself to recognize that the word "dog" is vocalized, and, it will learn automatically what the temporal boundaries are of each viseme, namely /t/ (viseme equivalent of the phoneme /d/), /o/, and /k/ (viseme equivalent of the phoneme /g/). In other words: it learns in what exact frames the viseme /t/ begins and ends. CTC eliminates the need to label each segment of a data sample exactly at the word boundaries, as well as the need to convert the output into a label sequence. They concluded that their model performed better than human lip readers and that the addition of 3DCNN feature extraction also increased performance. For future work, they suggest performing a similar experiment on the LRS dataset [8]. As a side note, CTC has been proven to be beneficial in, among other fields, handwritten character recognition [22], speech recognition [24], and machine translation [66].

In [64], a 3DCNN, a ResNet, and a bidirectional LSTM were sequentially combined to achieve an 83.0% accuracy on the LRW dataset. In this paper, they used the dataset's full-length sample video for their classification and do not make use of the word boundaries, i.e. the time in the video at which the word vocalization begins and ends. Later, in [65], the same authors do make use of the word boundaries, while testing out a slightly different model. They conduct two distinct experiments in this paper: they use a similar model as in [64] to perform the same closed-set word classification task as is common in ALR research on the LRW dataset, and, they extract word embeddings from 350 word classes to subsequently classify the other 150 words. The latter condition, called 'low-shot learning', means that the training set words and the test set word classes were two disjoint sets. In the closed-set classification experiment, they used a model similar to that in [64], but with the addition of word boundaries. With this new model, they achieve the current (i.e. at the time of writing) state-of-the-art accuracy of 88.08%.

In the low-shot learning experiment, Stafylakis et al. perform Probabilistic Linear Discriminant Analysis (PLDA) [30] to model a word embedding that summarizes certain information of the word class that is relevant to the classification task, while inhibiting irrelevant information. Specifically, PLDA is a linear dimensionality reduction method that extracts a low-dimensionality vector representation from the video data samples while maintaining the contextual closeness between words. In our case specifically, the context is related to visemes and, thus, the embeddings of words that look similar when vocalized, will look similar to each other as well (e.g. 'cancer' and 'against', 'makes' and 'means'). Thus, in this experiment, they aim to classify a subset of 150 word classes based on the word embeddings modeled on a

disjoint subset of 350 remaining word classes of the 500-class LRW dataset ('unseen word' condition). For comparison, they classify a number of words in the validation sets of all 500 classes based on word embeddings modeled on the test sets of the 500 classes ('seen word' condition). They varied the number of training instances per word embedding to a maximum of 16 words. At this maximum of 16 training examples per class, the model in the 'seen word' condition achieved 88.1% accuracy, and the model in the 'unseen word' condition achieved an impressive 82.7% accuracy. For future work, the authors express their ambition to use the same classification technique with sentence-level ALR.

Others have aimed to reconstruct an acoustic speech signal from audio-less videos. This process by which phonetic information is inferred from articulatory facial motion is called speechreading. Ephrat presented an encoder-decoder CNN in [16] that encodes raw video from the GRID dataset [9] and its optical flow into a vector representing the visual features. Subsequently, this vector is fed into a decoder, which consists of two fully connected layers that output a mel-scale spectrogram (i.e. a graph that has the spectrum of frequencies plotted as they vary with time, scaled in a way such that the distance between frequencies is perceptually realistic, instead of physically correct). Finally, this mel-scale spectrogram is fed into a post-processing layer which outputs a linear-scale spectrogram, which is used to generate a waveform. They achieve good results by evaluating their generated speech using several quantitative metrics but encourage the reader to watch and listen to the videos on their project web page [17], which, in their opinion, demonstrate the excellence of their method. They do provide, as a side note, the fact that the method is speaker-dependent and that creating a similar method that is speaker-independent is challenging.

Domain-adversarial training has also been applied in ALR. In [70], a single neural network was trained to classify words and the speaker simultaneously. This was done by copying the output of the second layer to another branch in the network (a two-stream neural network that has one input and two outputs). This secondary branch was trained to classify which speaker was speaking. However, instead of performing backpropagation by gradient descent from this secondary branch, gradient *ascent* was used from this secondary branch. This caused the first layers, and thus the complete word classification branch of the network as well, to confuse the speakers instead of learning to implicitly use the speaker as a feature. In a speaker-independent classification task, the test set accuracy rose in all conditions where the domain-adversarial training method was applied in comparison to the regular training method.

The final research we want to discuss is the audiovisual speech recognition method presented by Petridis et al. in [50]. In this paper, the authors designed a two-

stream neural network that in one stream takes as input a video and in the other stream an audio waveform. Each stream consists of a ResNet followed by two bidirectional LSTM layers. Subsequently, the output is concatenated and passes two more bidirectional LSTM layers. The difference between the vision and audio stream is that the visual ResNet consists of 34 layers and the audio stream ResNet consists of 18 layers, and, the video sequence first goes through a 3DCNN layer. They compare the performance of their model to audio-only and vision-only models in several signal-to-noise conditions. In a noiseless condition, the audiovisual speech recognizer outperforms both the audio-only and vision-only models (98.0% vs. 97.7% and 83.0%, respectively). In this comparison, the improvement from audio to audiovisual is only slight. In noisy conditions, however, the improvement is more notable. Specifically, in noisy conditions, the absolute improvement in test accuracy is between 1.3% and 14.1%, respectively.

4 Neural Networks

> *Users do not care about what is inside the box, as long as the box does what they need done.*
>
> **— Jef Raskin**

In the previous chapter, we have described the history and current state of ALR, which eventually became a field dominated by neural networks, as did many other fields that involved image or video classification. In this chapter, we will first discuss how the Multilayer Perceptron (MLP) works. Second, we will discuss the Convolutional Neural Network (CNN). And, third and final, we will discuss the Recurrent Neural Network (RNN). There are already many papers elaborating on the operations in neural networks, but our goal is to provide a more concise, intuitive, and comprehensible explanation.

Neural networks are *loosely* inspired by biological brains. The resemblance is often overstated, but there are two analogies to be made here. The first is that layer-wise processing takes place in both biological brains and neural networks. In humans' visual system, light received as raw light values in the retina is passed on to the primary visual cortex (V1). Subsequently, relatively low-level 'features' are 'extracted' by simple cells in V1, and complex cells and hypercomplex cells that are found in V1 and the secondary visual cortex (V2) [39]. And finally, after being processed by many different components of the visual system, we become conscious of the fact that we are indeed perceiving what we are looking at and the biological object recognition process is complete. Neural networks applied to computer vision, for example, receive raw pixel values (analogous to the retina receiving light) and subsequently processes this layer by layer until it is able to execute its task, for example classifying an image of a dog as a dog. As a side note; the human brain does not process the environment in a closed stimulus→response sequence manner as neural networks do. A human brain works via a feedback system called the sensorimotor loop [34]. This means that the brain is constantly reacting to its environment, and these reactions change the environment to which it responds to almost recursively, to put it in computer science terms.

Another analogy between neural networks and the human brain is that both are heavily interconnected. In an MLP for example, each node in an arbitrary layer is

given input by all nodes in the preceding layer, and it gives its output on to all nodes in the succeeding layer. A connection between two nodes is called a *weight*. A weight in a neural network has a value, which indicates the direction and strength of the influence of one node to the next. The value of a weight is a trainable *parameter*. This means that the value of a weight is constantly adjusted during the training to increase the performance of the network as a collective to increase the classification accuracy. Similarly, there is also a case of heavy interconnectivity between processing 'layers' in a biological brain, but the mechanism of a neural network is *by no means* neurologically plausible, despite this similarity [14, 68]. However, information is passed on to succeeding layers by a process in which the nodes in one layer are heavily interconnected with the nodes in the next layer in both the brain and a neural network. Each node in a neural network, or neuron in the brain, has a specific task, but no single node in neither a neural network or a biological brain is decisively influential [36, 45]; this property is called graceful degradation. It prevents that a faulty node or a relatively small group of faulty nodes causes the complete system to be faulty.

There are many applications of neural networks. In this chapter, however, we will discuss only the mechanism of neural networks used for classification tasks. To support the intuition behind our explanation in the following sections, we conduct several experiments documented in the next chapter and guide the reader through the inner workings of the CNN and RNN by analyzing these experiments.

4.1 Multi-Layer Perceptron

In this section, we will first discuss the general mechanism of a feed-forward neural network. Then, we will delve deeper into the concepts mentioned in this general introduction.

A perceptron, of which the foundation was introduced in [55], is an algorithm that computes a binary function based on its inputs. It was originally used to classify an image into one of two classes. It had no hidden layers, only the raw pixel values fed into a single node that performed the classification. Later came the Multilayer Perceptron neural network (MLP), the most basic type of neural networks used nowadays. Even the most elaborate and complicated neural networks contain a regular feed-forward neural network. It is characterized by having an input layer, an output layer, and one or more so-called 'hidden' layers (Fig. 4.1). Input is fed into the first layer and it is processed layer by layer until it reaches the output layer. Each node in the network (except the input nodes) also receive input from a bias node. The role of this bias input is to provide the individual nodes in the network with a

trainable constant value in addition to the input a node receives from the nodes in the preceding layer.

The activation in the last layer is decisive for the task; this is where the classification is performed. Each node in the output layer represents a class. For example, a network used to classify pictures based on what animal is depicted will have one output node for each animal.

The terminology used in this thesis is the following: a forward pass is the processing of an input in order to produce its corresponding output; a backward pass is the process by which the network calculates the loss function's gradient w.r.t its weights' by comparing the desired output of the network to the actual output. The gradient is the vector (set) of partial derivatives of the loss function w.r.t. the weights.

Activation Functions

So far we have mentioned activation functions and have discussed activation. But what exactly are activations and activation functions? Activation is the value that a node has during a forward pass. The activation of a node in the input layer

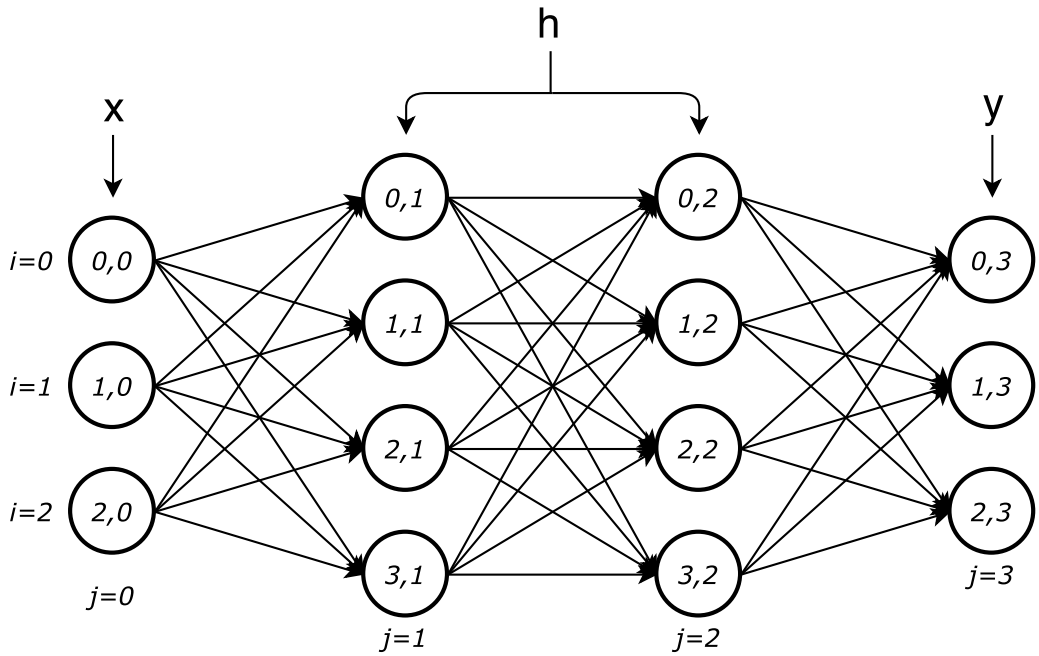

Fig. 4.1: A four-layer Multi-Layer Perceptron. A node is shown as a circle and weights are shown as arrows between nodes. x is the input layer, h are the hidden layers, and y is the output layer. Layers are numbered with indice j, and nodes in any arbitrary layer j are numbered with indice i. Nodes in any layer j except the input layer $j = 0$ receive their input from all nodes in the previous layer $j - 1$ and forward their output to all nodes in the next layer $j + 1$. Moreover, each node performs a certain activation function on the input before forwarding it to the next layer.

is the value of that particular element of the input vector. The reason a neural network needs an activation function is to make it able to learn a non-linear function. Without an activation function, a neural network could only learn a simple linear function. The activation y of an arbitrary node a at position i in layer j, $y_{a_{ij}}$, is computed by taking as input the output from the nodes in the preceding layer, $x_0 ... x_n$. Subsequently, node a_{ij} computes its activation based on the predetermined activation function and passes it on to all nodes in the succeeding layer $j+1$ (Fig. 4.2).

$$y_{a_{ij}} = f\left(\sum_{k=0}^{n} x_k w_{kj}\right) \tag{4.1}$$

where f is the activation function, n is the number of nodes in the layer $j-1$, x is the activation of node a_{kj-1} and w_{kj} is the weight from node a_{kj-1} to node a_{ij}. The activation function decides how an arbitrary node calculates its activation. There are countless activation functions able to be used in a neural network. However, we will discuss the activation functions most widely used today. We will also discuss their strengths and weaknesses.

The most basic activation function is the linear function, or identity function. It computes the dot product between all input nodes and the weights of these input nodes to the current node, as in Equation 4.1. Thus, the linear activation function is:

$$f(x) = x \tag{4.2}$$

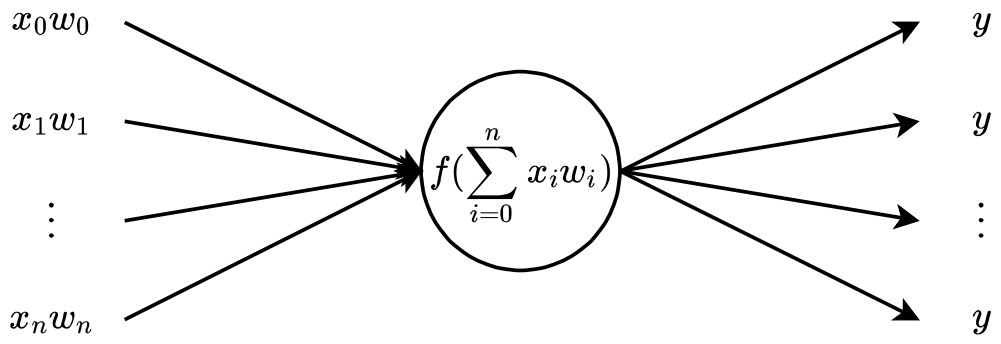

Fig. 4.2: Schematic visualization of the mechanics within a single node in an MLP. The node i in arbitrary layer j receives the activations from all nodes in the preceding layer $j-1$, $[x_0, x_1, ..., x_n]$ via their corresponding weights $[w_0, w_1, ..., w_n]$. Subsequently, node i computes its output, given a predefined activation function f.

Another widely used activation function is the logistic activation function, also called the sigmoid function. The sigmoid function takes a scalar value and converts it to a value in the range of $(0, 1)$:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

The hyperbolic tangent, or tanh, function converts a scalar value to a value in the range of $(-1, 1)$ and it is computed as follows:

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.4}$$

The Rectified Linear Unit (ReLU), however, is currently the most successful and widely used activation function in neural networks according to Ramachandran et al. in [52]. Basically, the ReLU thresholds the input at 0, thus converting a number to the range of $[0, \infty)$:

$$f(x) = max(0, x) \tag{4.5}$$

The last activation function we want to discuss is the SoftMax activation function. This activation function is mostly used in the output layer, where the classification is done. The activation of an arbitrary node a_{ij} depends on the input to node a_{ij} and the total input to layer j. The SoftMax activation is computed by taking the exponential function of the input to an arbitrary node a_{ij} and dividing it by the exponential function of the total input to layer j as follows:

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \text{ for } i = 1, ..., J, \tag{4.6}$$

where J is the number of output nodes. $f_i(\vec{x})$ can be interpreted as a probability of class J because the output is a probability distribution of J possible outcomes.

Initialization

Another point of consideration is the initialization of the parameters in the network. These parameters, or weights, decide how strong the connection is between two

nodes in two consecutive layers. The weights that connect nodes in consecutive layers are often referred to as kernel weights, and the connections between bias nodes and nodes in a certain layer are often called bias weights. A neural network's weights have to be initialized with a certain inequality within the parameters' values. This is to produce symmetry breaking. Symmetry breaking causes the different elements within a neural network such as individual weights and nodes to adopt different functions from one another. Consider a three-layer neural network with each layer containing an arbitrary number of nodes. If each parameter is initialized to the same value, the activations of all nodes in the first hidden layer will be equal. This is because each of these nodes takes as input all activations of the input, each multiplied by the same arbitrary number. Naturally, all the outputs will be equal as well, and thus, the gradients of weights will be equal per layer. This makes it impossible for a neural network to approximate a complex non-linear function.

A common solution to this is to initialize the kernel weights to random small values in the range of $-\epsilon$ and ϵ with mean 0. There are a lot of methods to initialize weights, many of which can be found in [5].

Loss Functions

The loss function is an important metric used to indicate the spread between the desired output of the network and the actual output. We will discuss two loss functions, namely the Mean Square Error (MSE) loss function, and the one we use in our experiments: Categorical Cross-Entropy (CCE). The MSE loss function is defined as:

$$E_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - t_i)^2, \tag{4.7}$$

where n is the number of output nodes, y_i is the output of node i, and t_i is the target output, or desired output. the CCE loss function, or log loss, is defined as follows:

$$E_{CCE} = -\sum_{i=1}^{n}(t_i \log(y_i) + (1 - t_i)\log(1 - y_i)) \tag{4.8}$$

CCE is a popular loss function in neural network research, but it has been suggested that other loss functions are more efficient and effective for some types of classification tasks [32].

Backpropagation

Backpropagation is the process by which the gradients of a loss function w.r.t. each of the parameters of a network is calculated. As stated before, the loss function of our choice was categorical cross entropy. Taking this into consideration, the process of backpropagation works as follows:

1. A forward pass is computed and the gradient (derivative) of E w.r.t. the output nodes must be computed in the following way:

$$\frac{\partial E}{\partial y} = \frac{y - t}{y(1 - y)}, \tag{4.9}$$

   where y is the output of a node, and t is its target output.

2. Then, by the chain rule, the gradient of E w.r.t. each weight in the preceding layers must be computed:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{i=1}^{k} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}}, \tag{4.10}$$

   where w_{ij} is the weight of node i in layer j. Here, j is the layer of weights that connects the last layer of nodes to the output nodes. The output of the second to last layer is indicated by y and the separate output of each node by y_k. This can be generalized in order to compute the gradients of E to nodes in all preceding layers (Fig. 4.3).

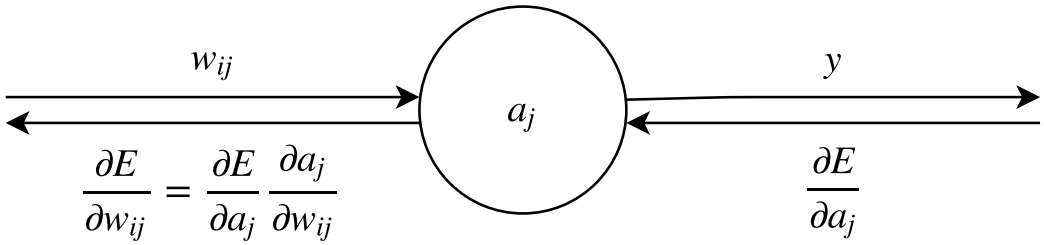

Fig. 4.3: A visualization of the chain rule. The arrows pointing right represent the values being forward propagated during a forward pass. The arrows pointing left represent the gradients that are computed during a backward pass. The gradient of E w.r.t. node a_j, and the gradient of node a_j w.r.t. weight w_{ij}, are assumed to be computed. The gradient of E w.r.t. w_{ij} can be computed by the chain rule as shown.

3. When all gradients have been computed, the weights are updates as follows:

$$w_t = w_{t-1} - \eta \frac{\partial E}{\partial w}, \tag{4.11}$$

   where t is a timestep in the training phase.

4. This process is repeated until a sufficiently low loss value has been reached. There are several choices one can make regarding when to stop training:

   - Stop training after a predefined number of epochs, or passes over the dataset.
   - Stop after the loss does not decrease within a predefined number of epochs from the last lowest achieved loss on the validation/training set.
   - Decrease the learning rate after the loss does not decrease within a predefined number of epochs from the last lowest achieved loss on the validation/training set.
   - Gradually decrease the learning rate over time regardless of the loss function.
   - A combination of two or more of the above.

The gradient of E w.r.t. a weight provides the information of what the influence is of a weight on the output. If the gradient is positive, increasing this weight causes the loss to increase and thus the weight has a negative influence on the network's performance. If the gradient is negative, increasing this weight causes the loss to decrease and thus the weight has a positive influence on the network's performance. Thus, in the first case the weight will be decreased, and in the latter case the weight will be increased.

Optimizers

After the gradients of the loss function w.r.t. the network's weights have been computed, a weight update needs to be made. Again, there are many ways to go about updating the weights. The different methods are called optimizers, or optimization algorithms. In this section, we will discuss two other optimizers, in addition to the optimizer denoted in Equation 4.11, which is the Stochastic Gradient

Descent (SGD) optimizer: Root Mean Square Propagation (RMSprop) [69], and Adaptive Moment Estimation (Adam) [37]. A more complete overview of choices of optimizers can be found in [56].

RMSprop is slightly more advanced than SGD. It is a gradient descent algorithm that adapts its learning rate based on an exponentially decaying average of the past gradients squared, v_t. This average is computed as follows:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) g_t^2, \tag{4.12}$$

where β_1 is the decay rate of the weight of the past average squared gradients in relation to the weight of the current squared gradient, g is the gradient of the loss function w.r.t. the weight, and v is the average of the past gradients squared. In our experiments we use $b_1 = 0.9$, as suggested in [69]. The weight at the current timestep, w_t, is subsequently computed as follows:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{v_t}} g_t \tag{4.13}$$

Adam also stores a running average of the gradients as in RMSprop. However, it is extended with the storage of another value, namely the exponentially decaying average of the past gradients, m_t:

$$m_t = \beta_2 m_{t-1} + (1 - \beta_2) g_t \tag{4.14}$$

Where β_2 is the decay rate of the weight of the past average gradients in relation to the weight of the current gradient. Moreover, in [37] it was observed that m_t and v_t tend to be biased towards zero. To counteract this, the authors correct their bias by computing the following 'bias-corrected' values:

$$\hat{v}_t = \frac{v_t}{1 - \beta_1} \tag{4.15}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_2} \tag{4.16}$$

These are used to update the weight in the following way:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}}\hat{m}_t \tag{4.17}$$

The authors suggest default values for β_1 and β_2: 0.9 and 0.999, respectively.

Regularization

Another important issue to consider in neural networks and machine learning in general is overfitting. To prevent overfitting, there is a process called regularization and the methods that do so are called regularizers. There are several types of regularizers, and we will discuss three of them: L1 regularization, L2 regularization, and dropout. In our experiments specifically, we will use dropout as a regularizer.

L1 and L2 regularization simply penalize E by adding a penalty value, based on the numbers of weights the network has and their values. L1 adds the sum of the absolute weight values in the network with a regularization factor strength λ:

$$E_{L1} = E + \lambda \sum_{i=1}^{k} |w_i|, \tag{4.18}$$

L2 regularization is relatively similar. However, it adds the sum of the squared weight values with a regularization factor strength λ:

$$E_{L2} = E + \lambda \sum_{i=1}^{k} w_i^2 \tag{4.19}$$

The reason why this works is as follows. Essentially, the task of a neural network is to learn a certain function such that it can perform well on the test set and desirably in the real world. When a neural network is overfitting, it is learning a function that can predict the training set well but does not perform well on the test set. A possible cause for this is that the network is learning a function that is too complex, which will learn the training set well but it will not learn to generalize its knowledge to the test set. By penalizing high weights and therefore obtaining smaller weights, the network learns a function with smaller slopes that is more fit to predict unseen data, i.e. the test data. Another reason why having smaller weights is better, is that a function with higher weights is more sensitive to noise and therefore might start to learn the samples from the training set better than the test set.

Dropout is a regularizer more specific to neural networks [63]. Dropout brings the activation of independent neurons in a certain layer and their effect on the neurons in the succeeding layer to zero at random with a probability factor p (Fig. 4.4). Effectively, a random different architecture is trained in each training epoch, and the forming of complex co-adaptation between nodes which would cause the network to overfit is prevented. Instead, each node learns a function that is independent of other nodes. Subsequently, when evaluating the model, dropout is not applied and the network can utilize all of its nodes and therefore its full capacity.

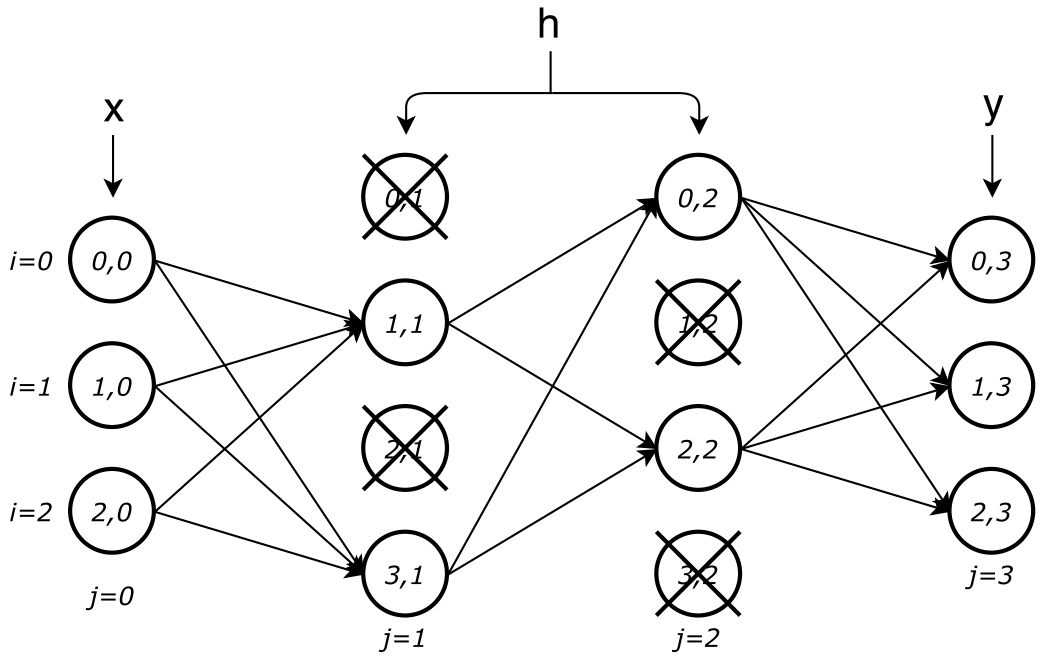

Fig. 4.4: Dropout with $p = 0.5$ applied to the two hidden layers in the MLP shown in Fig. 4.1. Note that when dropout is applied to a layer, the nodes chosen to be deactivated are randomly chosen every iteration.

4.2 Convolutional Neural Network

A CNN is a type of neural network of which the architecture is inspired by the architecture of the visual cortex. A neuron in the visual cortex responds to stimuli occurring in a restricted subsection in the field of vision. The specific region of space to which the individual neuron responds is called the receptive field. [29] showed that the output of cortical cells in certain layers in the visual system form the input to the receptive fields of cortical cells in subsequent layers in the visual system. Thus, small elementary receptive fields originating in the retina are phase-wise combined at following stages in the visual system to form large and complicated receptive fields that process complex visual information. As an analogy to this, neurons in a convolutional layer have a specific area in the preceding layer as their receptive field. A convolutional layer applies several filters to the visual input in order to

obtain feature maps. Usually, a CNN extracts an increasing amount of feature maps as its convolutional layers get deeper. However, the feature maps are reducing in size every layer it passes. By having multiple layers of convolution, increasingly complex features are extracted. Moreover, apart from convolutional layers, a CNN also contains pooling layers. The role of the pooling layers is to attain dimensionality reduction of the data by creating a summary of each subspace in the feature maps. Most often a CNN will consist of: several layers of convolution, each followed by a pooling layer, and several fully connected layers (Fig. 4.5).

The foundation of the modern CNN originates from the mechanism of a visual pattern recognition model called the Neocognitron, designed by [20]. It models how visual information passes through different layers of visual cortical cells. The structure is similar to the structure proposed by [29]. [20] modeled the visual system mechanism discovered by [29] containing, among other types of cells, simple and complex cells. In the Neocognitron, as well as in Hubel and Wiesel's model, information is processed at an increasingly higher level. That is: receptive fields of simple cells are linear and dependent on location and orientation and because they only respond to changes in gradient in specific orientations and locations, they take the role of edge detectors; complex cells' receptive fields are dependent on motion and direction and take the role of motion detectors, because they respond to shifting gradient changes in specific directions at specific locations; receptive fields of hypercomplex cells are dependent on orientation, motion, and direction, and take the role of angle detectors, because they combine complex cells' signals to produce information regarding the orientation of a certain motion [20, 29]. Cells at higher levels are able to take on their respective roles because they receive information from multiple adjacent cells in the preceding level. When comparing the model from [29], the Neocognitron from [20], and the CNN's general architecture in Fig. 4.5, a certain similarity exists.

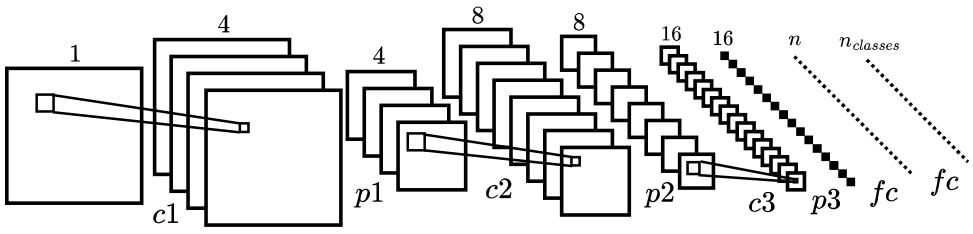

Fig. 4.5: A CNN with three layers of convolution and pooling, and two fully connected layers. The convolutional layers are indicated with $c1$, $c2$, and $c3$. The pooling layers are indicated with $p1$, $p2$, and $p3$. As illustrated, the convolutional layers extract multiple representations from their respective inputs, and the pooling layers reduce the size of said representations.

While the Neocognitron might have been the foundation of today's CNN, it was still very far from what is called a CNN today. The first modern CNN was LeNet-5, and the paper it was presented in coined the term 'convolutional neural network' [30]. It was the first neural network that used a number of alternating convolutional layers and pooling layers, followed by a number of fully connected layers to classify images. Despite this CNN performing very well, the first major performance breakthrough of CNNs came in 2012, when the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) [57] was won for the first time by a CNN, presented in [38]. In the ILSVRC, teams submit their classification algorithms, among other computer vision algorithms, and results to compete in an image classification challenge. The dataset of images to be classified is the ImageNet dataset [13]. In the following years, an increasing number of CNNs were among the top performing submissions in the ILSVRC. Currently, most of the competitors use some type of NN to compete Thus, a CNN or a variation on a CNN is the first choice for any image classification task in the present-day.

So far we have only discussed CNNs for classifying images. However, CNNs can also be adapted to take into account temporal information in order to process videos instead of images. By using 3D kernels for convolution instead of 2D kernels, 3D feature maps can be obtained of which the third dimension contains temporal information; and by using 3D pooling instead of 2D pooling, temporal features can be reduced in dimensionality together with spatial features [33]. In regular CNNs, the feature maps that are obtained after every layer of convolution are 2-dimensional. In 3D CNNs, the feature maps are 3-dimensional.

The CNN and neural networks that have a CNN component, have been applied to many other problems apart from image and video classification. For example: chemical data analysis [21], image colorization [76], sentence relation extraction [44], automatic game play [61], text sentiment analysis [60], image caption generation [73], and video caption generation [74].

Convolution

In Section 4.1, we have seen that regular neural networks consist of several layers of nodes and that all nodes in an arbitrary hidden layer j receive input from all nodes in the preceding layer $j-1$ and give input to all nodes in the succeeding layer $j+1$. In a CNN, nodes in consecutive layers are heavily interconnected as well. However, there are several constraints imposed on these connections in a CNN. These constraints make CNNs a better fit to perform classification on image and video data. The constraints are: local connectivity and parameter sharing.

Local connectivity means that an arbitrary neuron in an arbitrary convolutional layer j has a limited group of neurons in the preceding layer $j-1$ as its receptive field, as opposed to a node in an MLP, which has all nodes in the preceding layer $j-1$ as its receptive field (Fig. 4.1). Local connectivity is effective, because, in image data, features are locally dependent and globally independent (i.e. whether a certain feature is present at a location in an image only depends on a small group of spatially adjacent pixels).

Then, parameter sharing exploits a different characteristic of image data. As explained, an arbitrary neuron in an arbitrary convolutional layer is connected to only a small group of adjacent neurons in the preceding layer. Additionally, the weights that connect neurons from layer $j-1$ to convolutional layer j are dependent on each other. Each node i in convolutional layer j has a different group of neurons as its receptive field. However, each group of weights from any group of neurons in layer $j-1$ to its corresponding neuron in a convolutional layer j share the *same* parameters. Moreover, a weight update is imposed on each of these groups of weights evenly. The reason for this is that a certain feature can be present anywhere in the image.

The unique group of weights applied to each group of neurons in layer $j-1$ is called a filter. And, the already mentioned feature maps are activation maps generated by the application of such a filter to an input. Generally, multiple layers of convolution and pooling are applied to an input image or video. This part of a CNN consisting of multiple layers of convolution has the role of feature extractor. Subsequently, these features become the input to one or more fully connected layers. As the number of convolutional layers increases, more and more complex features can be extracted. However, this does not apply to every type of data and all datasets. Moreover, the first layer or layers extract low-level features (e.g. Gabor filters and color blobs) and the features become increasingly higher-level when the input is processed by deeper layers [75].

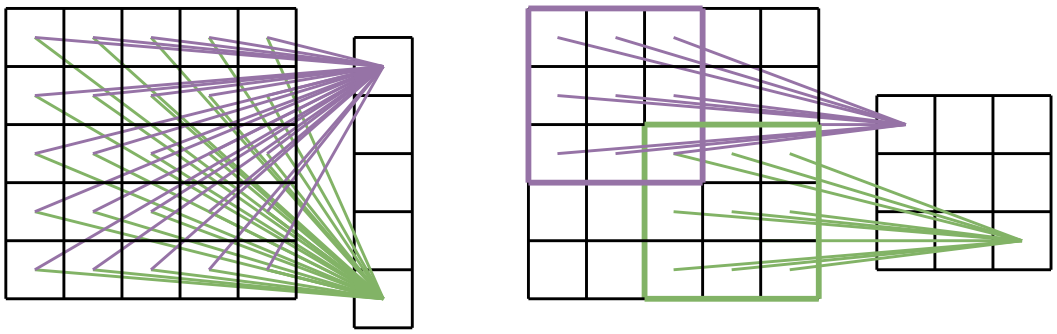

Fig. 4.6: A visualization of the connections in a convolutional layer (right), in relation to the connections in a fully connected layer (left).

So far we have only discussed CNNs for classifying images. However, CNNs can also be adapted to take into account temporal information in order to process videos instead of images. By using 3D filters for convolution instead of 2D filters, 3D feature maps can be obtained of which the third dimension contains temporal information; and by using 3D pooling instead of 2D pooling, temporal features can be reduced in dimensionality together with spatial features [33] (Fig. 4.6). In regular CNNs, the feature maps that are obtained after every layer of convolution are 2-dimensional. In 3D CNNs, the feature maps are 3-dimensional.

Pooling

Another important objective of a CNN is dimensionality reduction. This is performed by pooling layers. Usually, each layer of convolution in a CNN is followed by a layer of pooling. Pooling is basically a method of non-linear subsampling. In a pooling layer, the representation is downsized by a certain factor for each dimension (Fig. 4.7). If the factor is 2, a 3D representation is evenly divided in $2 \times 2 \times 2$ subdivisions. Depending on the pooling method, a single value is taken from these subdivisions and this results in a smaller representation. The reasoning behind this is that the precise location of a certain feature is not as important as its position in relation to other features, and it reduces the number of parameters in a network significantly. Two methods of pooling are most common: average pooling and max pooling. Max pooling takes the maximum value of the subdivision, and average pooling takes the average value.

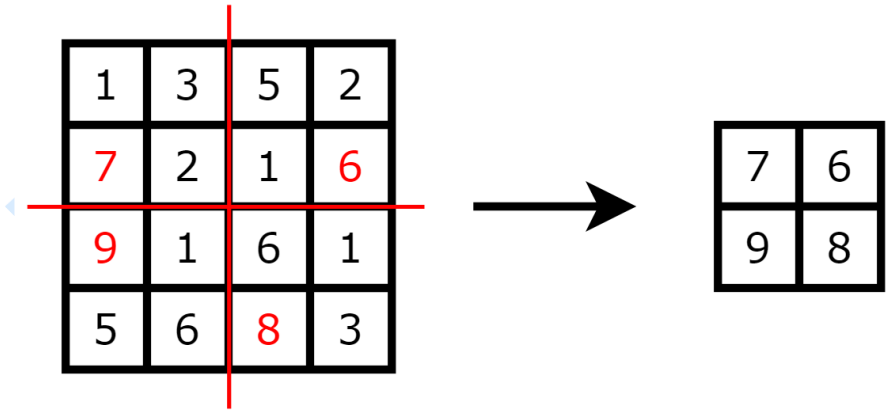

Fig. 4.7: A visualization of the max pooling operation. The pooling factor in both the x and y dimension the pooling operation is applied with a factor of 2.

Backpropagation

Because the general architecture is quite different from a regular feed-forward neural network with fully connected layers, the generality of our explanation of backpropagation does not fully apply in CNNs. In this section, we will discuss backpropagation through convolutional layers, and through pooling layers. Backpropagation through fully connected layers is already explained in Section 4.1.

Backpropagation through a layer is computed as follows. As we already explained, there are many weights in a convolutional layer, but only a few parameters. To compute the gradient of the loss function w.r.t. a filter, the gradient has to be computed for every connection in the layer. Subsequently, the gradients of the weights represented by a single parameter are summed.

In our experiments, we will use maxpooling of size $2 \times 2 \times 2$. For simplicity, however, we will discuss how backpropagation through a 2×2 maxpooling layer is computed. Consider a forward pass in which a 2×2 maxpooling operation is performed on the following subdivision:

$$\begin{matrix} 7 & 6 \\ 9 & 8 \end{matrix}$$

The output will be:

$$9$$

The position from which this 9 originated from has to be stored. Afterwards, during backpropagation, it is checked what position in the original input the 9 came from and this position is assigned a gradient of 1, while the other positions receive a gradient of 0:

$$\frac{\partial y}{\partial x} = \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}$$

4.3 Recurrent Neural Network

An RNN is a type of neural network that is designed to process sequential data. In sequential data, the chronological order of events is one of the key characteristics. This is because an event at an arbitrary timestep is dependent on the event that occurred before it. Examples of data for which this is true are time series data, human action recognition, speech recognition, and optical character recognition. In optical character recognition, the possible characters at position t depends on the characters at position t_z until $t-1$ where z is a non-negative integer. How large z is, and thus how far back the dependency goes, remains an empirical question. Moreover, RNNs take into account the chronological order of the input data by having an internal state that keeps track of the events it has seen before and the order in which it has seen said events.

In general, an RNN works as follows:

1. The network receives an input vector at timestep t_0, x_{t_0}, and updates it internal state at t_0, h_{t_0} by applying its activation function, the hyperbolic tangent function $tanh$, on its input as follows:

$$h_{t_0} = tanh(x_{t_0} w_x + b), \tag{4.20}$$

   where w_x is the input weight matrix.

2. Subsequently, it passes its internal state on as an input to the copy of itself at timestep t_1. The internal state h at timestep t_1 thus receives as input: h_{t_0}, and the input vector at timestep t_1, x_{t_1}. The internal state at an arbitrary timestep t_n is computed as follows:

$$h_{t_n} = tanh(x_{t_n} w_x + h_{t_{n-1}} w_h + b) \tag{4.21}$$

3. This process is repeated until the cell at the final timestep has its internal state h_{t_N} computed. This last internal state forms the output of the RNN cell and is passed onto the subsequent layer in the network:

$$y = h_{t_N} \tag{4.22}$$

or, when the RNN is set to output a sequence, the RNN gives its hidden state at every timestep as output:

$$y = [h_{t_0}, h_{t_1}, ..., h_{t_N}] \tag{4.23}$$

Backpropagation through time

The process through which the derivatives of E w.r.t. the weights in an RNN is calculated is the same as in a regular NN. However, the weight update is slightly different. The gradients are calculated for the complete unrolled network. Thus, each weight has one gradient in each timestep. For clarity, we will use a slightly different notation. We will use t to indicate a timestep in the training phase, and we will use T to indicate a timestep in the RNN. Recall that the weight update in a regular neural network is computed as denoted by Equation 4.11. A weight update in an RNN is computed as follows:

$$w_t = w_{t-1} - \eta \sum_{i=0}^{T} \frac{\partial E}{\partial w_i} \tag{4.24}$$

What happens is that, as usual, all the gradients in the network are computed. However, because the weights at multiple timesteps in the RNN are represented by one single value, there are more gradients than parameters. For example, weight matrix w_h is used at every timestep, but it is the exact same weight matrix at every timestep. Thus, the gradient of E w.r.t. w_h is computed at every timestep. These

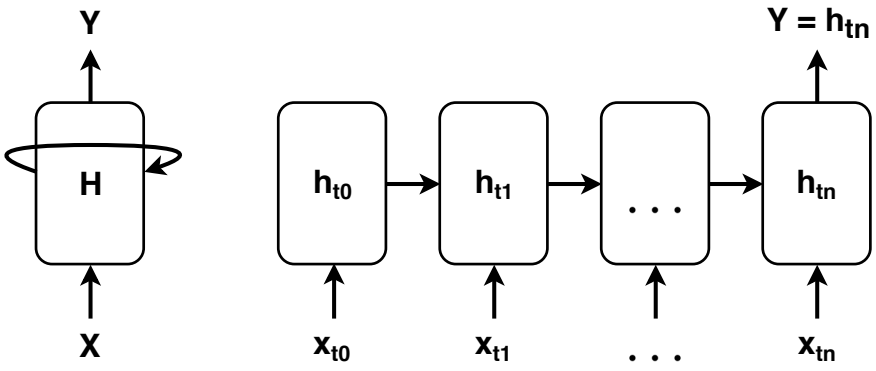

Fig. 4.8: The general architecture of an RNN. Both are visualizations of the same architecture: on the left the folded visualization is shown, and on the right the unfolded visualization is shown.

gradients must be summed and the outcome of that summation is the gradient of E w.r.t. w_h that is used for the weight update.

Long Short-Term Memory

The RNN described so far is more fit to handle sequential data than an MLP. Often, however, the sequences an RNN has to process contain long-term dependencies: not only are there directly successive temporal interdependencies, but the temporal interdependency between events can span over prolonged intervals. The RNN described is not fit to handle such data, because it suffers from a phenomenon called the vanishing and exploding gradient problem [3]. This means that during backpropagation through many layers, gradients might either increase out of proportion (exploding gradient) or decrease and approach near-zero values (vanishing gradient). In both cases, the network is unable to update its weight proportionally. If the gradient 'vanishes', the network will apply negligible weight changes or none at all. And, if the gradient 'explodes', the network will apply weight changes that are out of proportion.

A solution to this problem is the Long Short-Term Memory neural network (LSTM), first presented in [27] in 1997. It maintains an internal state similar to the regular RNN. However, it has an additional internal state: its memory, which enables the LSTM to learn long-term temporal contingencies. The mechanism is slightly more complicated than the mechanism of the RNN. An LSTM cell receives the input at the current timestep, x_t, the hidden state output of the previous timestep's LSTM cell, h_{t-1}, and the memory state output of the previous timestep's LSTM cell, c_{t-1}. The internal states h_t and c_t are computed as follows (Fig. 4.9):

1. Several gating values f_t, i_t, and o_t, are computed:

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{4.25}$$

$$i_t = \sigma(w_x[h_{t-1}, x_t] + b_i) \tag{4.26}$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{4.27}$$

2. In parallel, the candidate memory state $\tilde{c}_t$ is computed:

$$\tilde{c}_t = tanh(w_o[h_{t-1}, x_t] + b_o) \tag{4.28}$$

3. Finally, the memory state c_t and hidden state h_t are computed:

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t \tag{4.29}$$

$$h_t = tanh(c_t) o_t \tag{4.30}$$

4.4 Transfer Learning

In machine learning, knowledge is conventionally learned in one domain only and used for one task only. A new prediction model is trained on a new task with different data from scratch when the domain or task changes. However, sometimes this is either: not possible, because of a lack of data in the new domain; not necessary, because knowledge learned from other tasks can reduce training time; or not optimal,

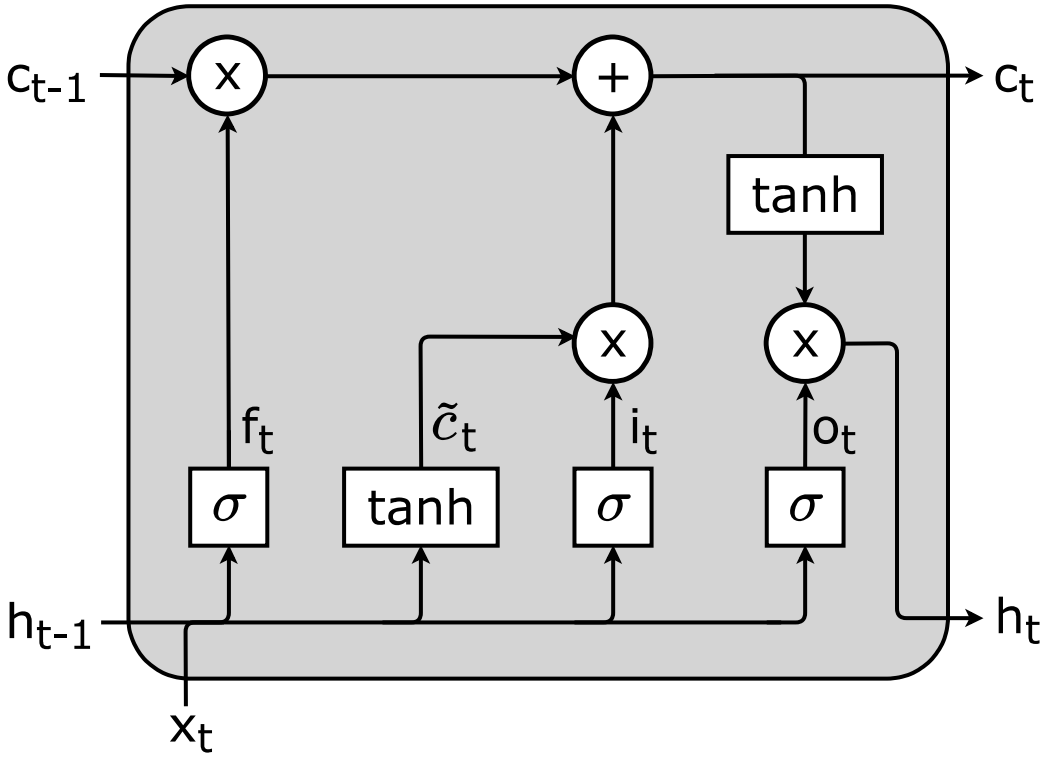

Fig. 4.9: The general architecture of an LSTM cell. The white circles indicate multiplication or addition operations, and the rectangles indicate activation functions.

because extracting knowledge from other prediction models can increase another model's performance. What can be used to bridge the seemingly incompatible knowledge is transfer learning. In the context of machine learning, transfer learning comprises of methods that aim to take learned knowledge from one domain or task, called the source, and use that knowledge in another domain or task, called the target. Transfer learning can be applied in many tasks, such as classification [35], regression [40], and clustering [11]. One motivation for this could be that the target does not have a large enough up-to-date dataset available from which a model can be optimized from scratch. [48] categorizes transfer learning into three different types: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning, which differ in what knowledge is transferred (Table 4.1). In inductive transfer learning, the source task and target task are different but related, while the domains are the same. In transductive transfer learning it is the opposite: the source task and target task are the same, but the source domain and the target domain are different but related. Finally, unsupervised transfer learning is similar to inductive transfer learning such that the source and target tasks are different but related, but unsupervised transfer learning focuses on cases where there is no labeled data available neither the source domain and target domain.

Another division that is made by [48] is the division of transfer learning approaches (Table 4.2). The first is instance-based transfer learning, in which certain knowledge learned in the source domain can be reused in the target domain by relabeling using the class labels in the source data to relabel the target data [10]. The second approach is feature representation transfer learning, in which the way features are represented in the source domain is transferred to the target domain [12]. This leads to features in the target domain being encoded more accurately which subsequently leads to an increase in target task performance. The third approach is parameter transfer learning, which relies on the assumption that parameters learned in the source task are useful to the target task. By transferring the parameters of the prediction model learned in the source task directly to the prediction model used for the target task, a significant performance increase and training time decrease can be achieved [35]. The fourth and final approach is relational knowledge transfer learning, which assumes that certain relations within the source domain data also apply in the target domain data. Moreover, not all transfer learning approaches can

Learning Setting	Domains	Tasks
Conventional Machine Learning	Same	Same
Inductive Transfer Learning	Same	Related
Transductive Transfer Learning	Related	Same
Unsupervised Transfer Learning	Related	Related

Tab. 4.1: Difference of conventional machine learning in relation to several transfer learning varieties.

be used for all transfer learning settings: transductive transfer learning can only make use of instance transfer learning and feature-representation transfer learning, and unsupervised transfer learning can only make use of feature representation transfer learning. Inductive transfer learning is the only setting in which all four approaches can be used.

The current paper, however, focuses solely on using transfer learning in neural networks. Specifically, we will focus on parameter transfer learning from a prediction model that contains knowledge learned on a source domain dataset, to a prediction model that will be optimized on a target domain dataset. The parameters of the model optimized on the source domain dataset are used as a starting point for optimizing the model on the target domain dataset. The reason for this is that the target domain dataset is not large enough to learn useful features from, which the neural network aims to extract and uses for classification. A typical transfer learning process in neural networks is to train a model from scratch on the source domain dataset, freeze (i.e. keep the parameters fixed) a number of layers, and then retrain the non-frozen layers, usually one or more layers in the end, on a different dataset as in [35, 47, 75]. Specifically, when transfer learning is used in neural networks, it is most often the case that knowledge from the first layer and n number of succeeding layers are transferred from the source task to the target task, where n is at most the number of layers in the source task's model minus 1.

Retraining means either initializing the layers' parameters as if they were trained from scratch, or using the formerly optimized parameters as a starting point. The latter case is often referred to as fine-tuning the layers. Another possible method is

Transfer Learning Approach	Description
Instance Transfer	Reweight labeled data from the source domain to train a model in the target domain.
Feature-representation Transfer	Features that are useful in the source domain are used in the target domain.
Parameter transfer	Model parameters that are beneficial in the source domain are used to train a model in the target domain.
Relational Knowledge Transfer	Define the relational knowledge in the target domain by using the already established knowledge in the source domain and the relation between the source and target domain.

Tab. 4.2: The different approaches to transfer learning.

training a number of starting layers on multiple different but related domains and training several sets of end layers separately on those domains, all using the same starting layers. This can be used in tasks where the separate domains share certain domain-general knowledge while being different to the extent where they require different end-layers, such as automatic language comprehension [28].

Transfer learning in neural networks can often be very useful, but the extent to which knowledge can be transferred depends on a few factors. As stated before, the first layers of a CNN process pixel-level patterns and the deeper into the network, the more specific and abstract the extracted information becomes. The type of knowledge extracted changes gradually from general knowledge being extracted in the first layers of a CNN and specific knowledge being extracted towards the last layers of the network. Thus, there seems to be a shift from general to specific knowledge processing through the network.

In [75], several methods of transfer learning were experimentally evaluated, namely whether knowledge was transferred from the bottom, middle, or top of the network. This was done to quantify the extent to which layers process specific or general information. They used the ImageNet dataset, which contains 1.3 million images distributed across 1000 classes. The dataset was split in half at random such that each half contains 500 classes, creating two datasets, named A and B. This served as the 'similar datasets' condition. They also split the dataset into the semantically dissimilar ImageNet parent classes: 551 man-made entity classes and 449 natural entity classes. This served as the 'dissimilar datasets' condition. In each condition, an 8-layer CNN was trained on both dataset halves separately. The numbers of layers from which knowledge was transferred were changed in each experiment and ranged from 1 to 8. They also experimented with either freezing or fine-tuning the transferred layers. Another experiment was that they used TL from task B to task B, to test whether co-adaptation between layers takes place. In the similar datasets condition, they made a few surprising discoveries, as well as confirming some expected occurrences. First, they expected that layers towards the end contain dataset-specific knowledge and thus it should not be useful to transfer these layers. This was indeed the case: the first two layers transfer seamlessly from task A to task B, transferring the layers till layer 3 showed a slight drop in performance, and the drop from layers 4 till the end showed a more significant decline in performance. Second, training a network on dataset A and fine-tuning a number of layers on dataset B provided a better performance than when a network is trained on dataset B directly. This was the case for any number of layers transferred. We suspect that this phenomenon occurred because of the similarity between the datasets and the increased number of training examples. Third, transferring three or more already learned layers from dataset B to a new model also trained on dataset B caused a drop in performance when the transferred layers were frozen and the rest of the

network was randomly reinitialized. This proves that the first network contained fragile co-adapted features that could only be extracted because successive layers were dependent on each other. This codependency could not be relearned when later layers are randomly reinitialized. Fourth, in a similar case where transferred layers were fine-tuned instead of frozen, this codependency among layers was learned again. Moreover, transfer learning always led to a performance decrease in the dissimilar datasets condition, regardless of from which layers knowledge was transferred.

5 Experiments

> *No amount of experimentation can ever prove me right; a single experiment can prove me wrong.*
>
> — **Albert Einstein**

In this chapter, we will describe the datasets we utilized, our methodology in the experiments, and the results obtained from these experiments. We split our experiments into two groups. The first group is concerned with the differences between the LSTM and the 3DCNN in the context of ALR. These experiments are conducted to investigate the differences between these models on several criteria, such as accuracy, training time, and computational cost. The second group of experiments is concerned with the transfer learning capabilities of the models. Here, the models trained in the first experiments are used to transfer knowledge to models that operate in the same domain, and, to models that operate in different but related domains. In all conditions, we cropped the mouth area from the video. We were able to choose the bounding box coordinates of the cropped area because all videos were already centered around the face. From the original video of size 256×256, we obtained a 30×60 'mouth crop' video (Fig. 5.1). The mouth crop video is used in all experiments unless stated otherwise. Moreover, all videos were converted to grayscale.

5.1 Datasets

The dataset we used for the first experiments was the Lip Reading in the Wild (LRW) dataset [7]. This dataset consists of up to 1000 vocalizations for each of the 500

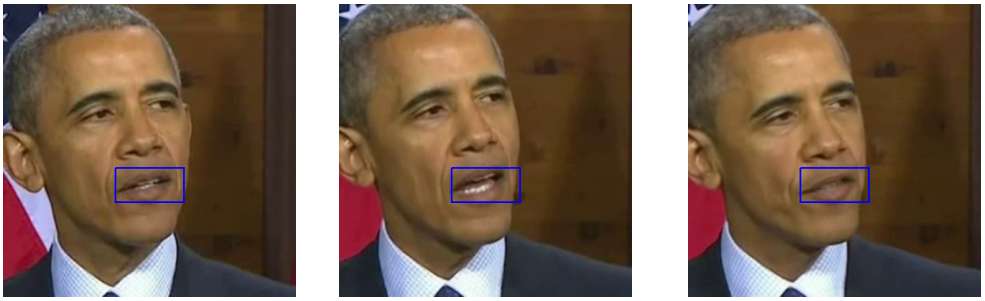

Fig. 5.1: Sample frames of a video in the LRW dataset. The blue boxes indicates the area that was used for the mouth crop video.

word classes. Each video in the dataset is 1.28s long and the word is vocalized in the middle, as part of a continuous stream of speech. We used two classes of this dataset for our experiments because the effects observed in a small dataset can most likely be extrapolated to the large dataset as well. The word classes we used were 'economic' and 'Westminster'. In the transfer learning experiment, we also use two other classes in this dataset, namely the 'politicians' and the 'temperatures' word classes. These classes were chosen because they were among the words with the highest recognition rate in [64]. For each class in this dataset; the training set consisted of 1,000 samples, and the test set and validation set consisted of 50 samples each.

For determining the suitability of the models for transfer learning to different domains, we use the UCF101 dataset, first presented in [62]. The UCF101 dataset is a human action recognition dataset, containing approximately 100,000 videos distributed over 101 human action classes. The videos contain realistic human action with a high variation of poses, camera angles, backgrounds, and object scales. We chose two classes from this dataset, namely the 'squat' and the 'hula hoop' classes. Each class in this dataset contains 25 groups of video samples. Videos within a particular group are several takes of the same scene, and thus are very similar. That is why videos from the same group should not be separated for the training and test split, for example. The video samples in this dataset are of various lengths, ranging from 48 to 267 frames. We cut every video in smaller videos of 29 frames because of this, starting from the first frame. The training, testing, and validation sets for the 'bodyweight squat' class are of size 313, 65, and 58, respectively. The training, testing, and validation sets for the 'hulahoop' class are of size 278, 68, and 53, respectively.

Fig. 5.2: Sample frames of the two classes from the UCF-101 dataset we used. The top row shows three frames from a video in the 'bodyweight squat' class, and the bottom rows shows three frames from a video in the 'hulahoop' class.

5.2 Methods

Experiment 1

In the first set of experiments, we have three experiments in which we compare the accuracy, total training time, and time per epoch. We designed a small RNN and a small 3DCNN and trained them on a small artificial dataset. In the second experiment, we reduced the 30×60 resolution to a 15×15 resolution video, thereby reducing the y dimension more strongly than the x dimension. Thus, the video will appear stretched or, from a different perspective, compressed. In the third experiment, we used the full mouth crop video. The summary of the differences between the models used in these experiments is shown in Table 5.1.

Experiment 1.1: Dummy Models

In this experiment, we designed a small artificial dataset which we used to train an RNN and a 3DCNN on. The artificial dataset consisted of eight samples of which one sample belonged to one class and the seven others belonged to the second class. Each sample was a $2 \times 2 \times 2$ matrix with each value being 0 or 1. The single sample belonging to one class was the following:

$$x_1 = \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right]$$

The other samples were of the same structure, but with a different binary pattern.

In all experiments but this one, we designed an LSTM. We decided to use an RNN here because we aimed to keep the number of parameters of both models relatively equal, while also keeping the number of parameters to a minimum. The RNN used

	Models	Input Size	# Parameters
Exp. 1.1	1-layer (1) 3DCNN	$2 \times 2 \times 2$	13
	1-layer (1) RNN	$2 \times 2 \times 2$	11
Exp. 1.2	2-layer (8/16) 3DCNN	$15 \times 15 \times 15$	10,658
	2-layer (16/16) LSTM	$15 \times 15 \times 15$	17,634
Exp. 1.3	3-layer (16/24/32) 3DCNN	$30 \times 60 \times 29$	96,218
	2-layer (16/32) LSTM	$30 \times 60 \times 29$	122,626

Tab. 5.1: The models used for the three experiments in Experiment 1. The numbers in parentheses in the 'Model' column indicate the number of feature maps every layer in the CNN extracts, or the output vector size in each layer in the RNN/LSTM.

here consists of one recurrent layer and one fully connected layer, adding up to a total of 11 parameters. The 3DCNN used consists of one convolutional layer with a single filter and one fully connected layer, adding up to a total of 13 parameters.

Moreover, both models were tested using three different optimizers, and, three different learning rates. Thus, Each model was trained in 9 different ways. The optimizers used are SGD, RMSprop, and Adam. The learning rates used are: 0.025, 0.05, and 0.1

Experiment 1.2: Reduced Size Data

For this experiment, the 30 $\times$ 60 video frames were resized to 15 $\times$ 15 frames, and each second frame was skipped. This resulted in each video being of size 15 $\times$ 15 $\times$ 15. Here, we do use an LSTM. The LSTM used for this experiment consists of two LSTM layers and one fully connected layer. Each LSTM cell in the network has an output of length 16. This LSTM contains 17,634 parameters. The 3DCNN used for this experiment consists of two layers of convolution and pooling, and two fully connected layers. The first layer of convolution extracts 8 feature maps, the second layer of convolution extracts 16 feature maps, and the first fully connected layer contains 16 nodes. In total, the 3DCNN contains 10,658 parameters. In this experiment, we chose to use the same single learning rate. However, we are still interested in the difference when using different optimizers, and thus we will train each model using three different optimizers: SGD, RMSprop, and Adam.

Experiment 1.3: Full-size Data

Here, we use the mouth crop video of the original size 30 $\times$ 60 and designed a new 3DCNN and LSTM. The LSTM used here is very similar to the LSTM used in the previous experiment. However, this LSTM's second layer of LSTM cells has an output size of 32. This results in this LSTM having a total of 122,626 parameters. The 3DCNN used here is similar to the previously used model as well. It is extended with another layer of convolution and pooling. The successive layers extract 16, 24, and 32 feature maps, respectively. These layers are followed by a fully connected layer of width 32 and a classifying layer. This 3DCNN contains 96,218 parameters.

Experiment 2

In the second group of experiments, the main aim was to compare both models' fitness for transfer learning. Another aim was to observe what the effect is of a smaller training set on the test set performance, and the contribution of transfer learning on smaller training sets. First, we test the models' fitness for parameter value transfer to words in the same dataset. This experiment is called the 'same domain' transfer learning experiment. With this experiment, we test the models' fitness for transfer learning within the same domain. Second, we test the models' fitness for parameter value transfer to videos in a different, but relatively related dataset. This experiment is called the 'different domain' transfer learning experiment, and with this experiment, we test the models' fitness for transfer learning to a different domain. In both the same domain transfer learning experiment and the different domain transfer learning experiment, we train a new model in three different conditions. First, we take the parameter values of the previously trained model as a starting point and fine-tune all parameters on the new words. Second, we take the parameters of the previously trained model as a starting point again, but this time retrain the final two layers from scratch while keeping the parameters of the rest of the network fixed. The third condition is used as a reference. Here, we train the model on the new word classes from scratch. Moreover, to observe the effects of smaller training sets on test set performance, and the contribution of transfer learning on smaller training sets, we conduct the same experiments with a smaller fraction of the training set. Namely, training set sizes of 10%, 5%, and 1%, of the original training set of a 1000 samples per class in the same domain condition. These smaller training set experiments will not be conducted in the different domain condition, because the dataset in this experiment is already quite small.

Experiment 2.1: Same Domain

In the same domain transfer learning experiment, we take the model used in the full-size data model of the first group of experiments. These were trained on the words 'economic' and 'Westminster'. Subsequently, we train a model under three different conditions on two different words of the LRW dataset, namely the 'politicians' and the 'temperatures' word classes. First, we train a model on the words 'economic' and 'Westminster' from scratch. Second, we take the parameters of the model optimized in Experiment 1 and fine-tune all layers on the two new words from the same dataset, 'politicians' and the 'temperatures'. The third condition is similar to the second condition. However, we only fine-tune the last layer of the network. Moreover, the same experiments are conducted with a smaller fraction of the training

set (10%, 5%, and 1%, of the original training set) to see what the contribution of transfer learning is when the training set decreases in size.

Experiment 2.2: Different Domain

In this experiment, we evaluate transfer learning in the same three conditions as in Experiment 2.1. However, here we aim to transfer knowledge from the domain of automatic lip reading to a different domain. The domain we chose was human body movement in natural scenes. The description of the dataset and classes used was already discussed in Section 5.1.

5.3 Results

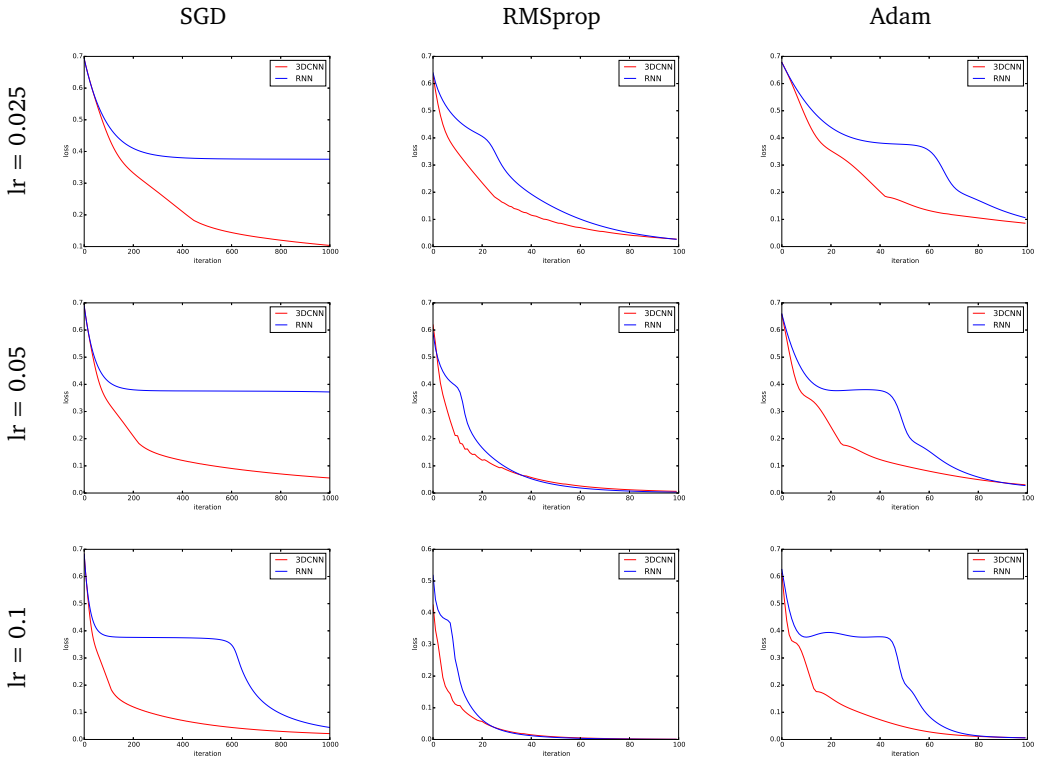

Fig. 5.3: The pair-wise comparison between the models in all nine conditions of Experiment 1.1. Note that the y-axis for SGD ends at 1000, while the others end at 100.

Experiments 1

Experiment 1.1: Dummy Models

In the first experiments, we compared the small versions of both models in nine conditions. The results of this experiment are visualized in Fig. 5.3. All of the optimizers benefited from a higher learning rate, given the tested learning rates of 0.025, 0.05, and 0.1. Moreover, RMSprop reached a near-zero loss value the fastest of all the optimizers, while SGD did this the slowest. The models reached near-zero loss values in each condition.

Experiment 1.2: Reduced Size Data

The results of this experiment are visualized in Fig. 5.4 and Fig. 5.5. In the pair-wise comparisons in Fig. 5.4, the decrease in loss of the models using different optimizers are compared. The 3DCNN seems to be faster when taking only into account the number of epochs until minimum loss. In Fig. 5.5, The differences in optimizer choice are plotted together per model. It can be seen that SGD is the slowest for both models. For the 3DCNN, Adam and RMSprop are more or less equally fast. For the LSTM, however, Adam is faster. The results regarding test set performance and training time can be found in Table 5.2. It can be read that the LSTM is faster in all cases, even though this cannot be inferred from Fig. 5.4.

	Test Accuracy	Training Time	Time per Epoch
SGD			
3DCNN	100%	1h 10m 30m	1m 38.4s
LSTM	98%	12m 23s	7.9s
RMSprop			
3DCNN	100%	28m 09s	1m 24.2s
LSTM	98%	9m 55s	6.3s
Adam			
3DCNN	99%	33m 49s	1m 46.8s
LSTM	99%	6m 10s	6.3s

Tab. 5.2: Test set accuracy and training time results of the reduced size data experiments of experiment 1.2.

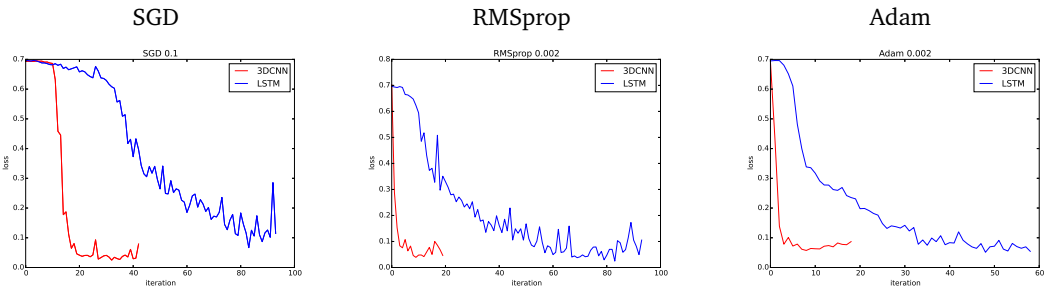

Fig. 5.4: The pair-wise comparison of the decrease in loss between the models per optimizer in experiment 1.2.

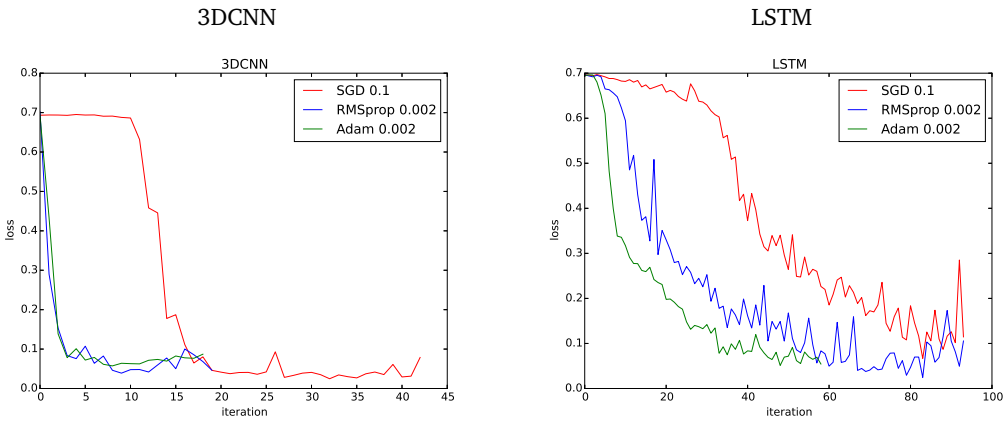

Fig. 5.5: The comparison of decrease in loss between the optimizers per model.

Experiment 1.3: Full-size Data

In this experiment, we used the full size 30 $\times$ 60 mouth crop video. Moreover, we compared the 3DCNN and LSTM in this experiment only once. In this comparison, the ideal learning rate was determined by educated guesses. The best learning rate found for the 3DCNN was 0.0005 and for the LSTM it was 0.00025. Both models were trained using the Adam optimizer. In Fig. 5.6, the decrease in loss for both models is visualized. In Table. 5.3, the test set performance, training time, and time per epoch are depicted.

	Test Accuracy	Training Time	Time per Epoch
3DCNN	99%	7h 59m 51s	36m 54.7s
LSTM	95%	39m 15s	27.4s

Tab. 5.3: The test set performance, training time, and time per epoch in the full size mouth crop video experiment of experiment 1.3.

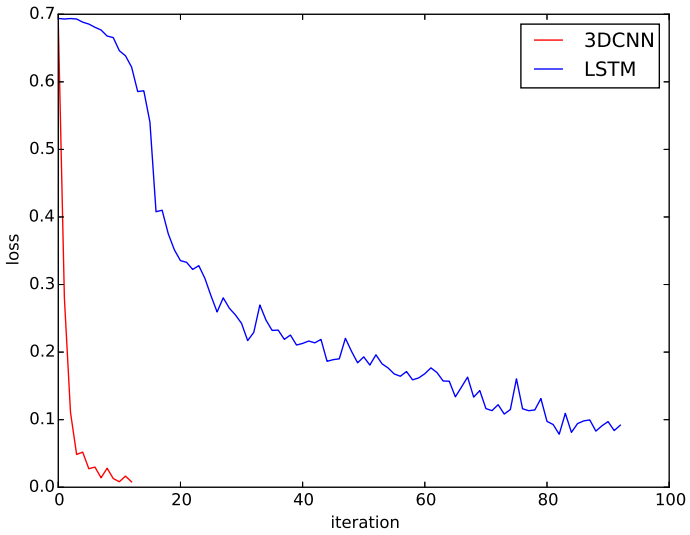

Fig. 5.6: The decrease in loss for the 3DCNN and the LSTM in Experiment 1.3. The 3DCNN needs only approximately 10 epochs to converge, while the LSTM takes approximately 90. It should be noted that the 3DCNN needs more time per epoch than the LSTM, as can be seen in Table. 5.3.

Experiments 2

Experiment 2.1: Same Domain

In this experiment, we tested the suitability of the 3DCNN and LSTM for parameter transfer to a target dataset within the same domain as the source dataset, and the effect of a smaller training set size on transfer learning in terms of performance. To do this, we used the models trained in experiment 1.3 and used the parameters to train a model on two different words in the LRW dataset. In Fig. 5.8, a figure is shown for each model's performance on the three conditions tested, and for all of the training set size conditions. In Table 5.4, the test set performance for each of the conditions is shown.

In the condition where the complete training set of 1000 samples per class was used, the LSTM's loss seems to decrease quite a lot in the condition where all layers were fine-tuned, in comparison to the other conditions. The 3DCNN's loss did not seem to decrease much when only the last layer was fine-tuned in transfer learning. However, when all layers were fine-tuned in transfer learning, the 3DCNN's loss decreased slightly faster than when it was trained from scratch.

In the conditions where smaller training sets were used, the 3DCNN's loss did not decrease at all when only the last layer could be fine-tuned. However, its loss

decreased at a similar rate when the model was trained from scratch and when transfer learning was applied where all parameters could be fine-tuned. In the LSTM's case, it did not benefit when the training set was of size 10 or 50, but training set performance did improve in the smaller training set of size 100.

	3DCNN	LSTM
1000		
Scratch	97%	75%
Layer	53%	82%
All	96%	97%
100		
Scratch	93%	63%
Layer	50%	82%
All	95%	82%
50		
Scratch	91%	62%
Layer	50%	48%
All	91%	49%
10		
Scratch	76%	50%
Layer	44%	32%
All	84%	54%

Tab. 5.4: Test set accuracy for all conditions of experiment 2.1. The number on the left indicates the training set size per class.

Experiment 2.2: Different Domain

In this experiment, we tested the suitability of the 3DCNN and LSTM for parameter transfer to a target dataset in a different domain as the source dataset. To do this, we used the model trained in experiment 1.3 and used the parameters to train a model on two classes in the UCF101 dataset. In Fig. 5.7, a figure is shown for each model's performance on the three conditions tested. In Table 5.5, the test set performance for each of the conditions in this experiment is shown.

Fine-tuning the last layer does not seem to benefit the LSTM in this experiment, compared to when it was trained from scratch. However, fine-tuning all layers in the LSTM does seem to make the loss decrease faster, albeit to approximately the same loss value as when it was trained from scratch.

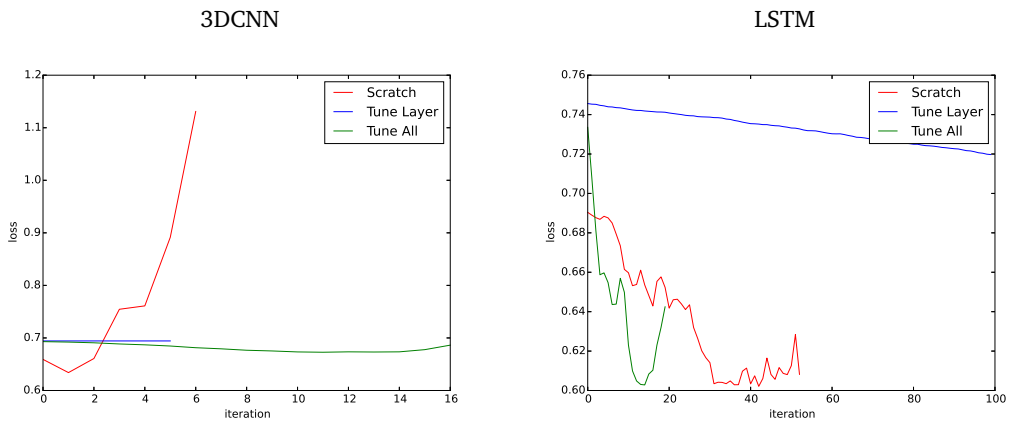

Fig. 5.7: The decrease in loss value for the 3DCNN and LSTM in experiment 2.2.

	3DCNN	LSTM
Scratch	65%	72%
Layer	49%	67%
All	49%	53%

Tab. 5.5: Test set accuracy for all conditions of experiment 2.2.

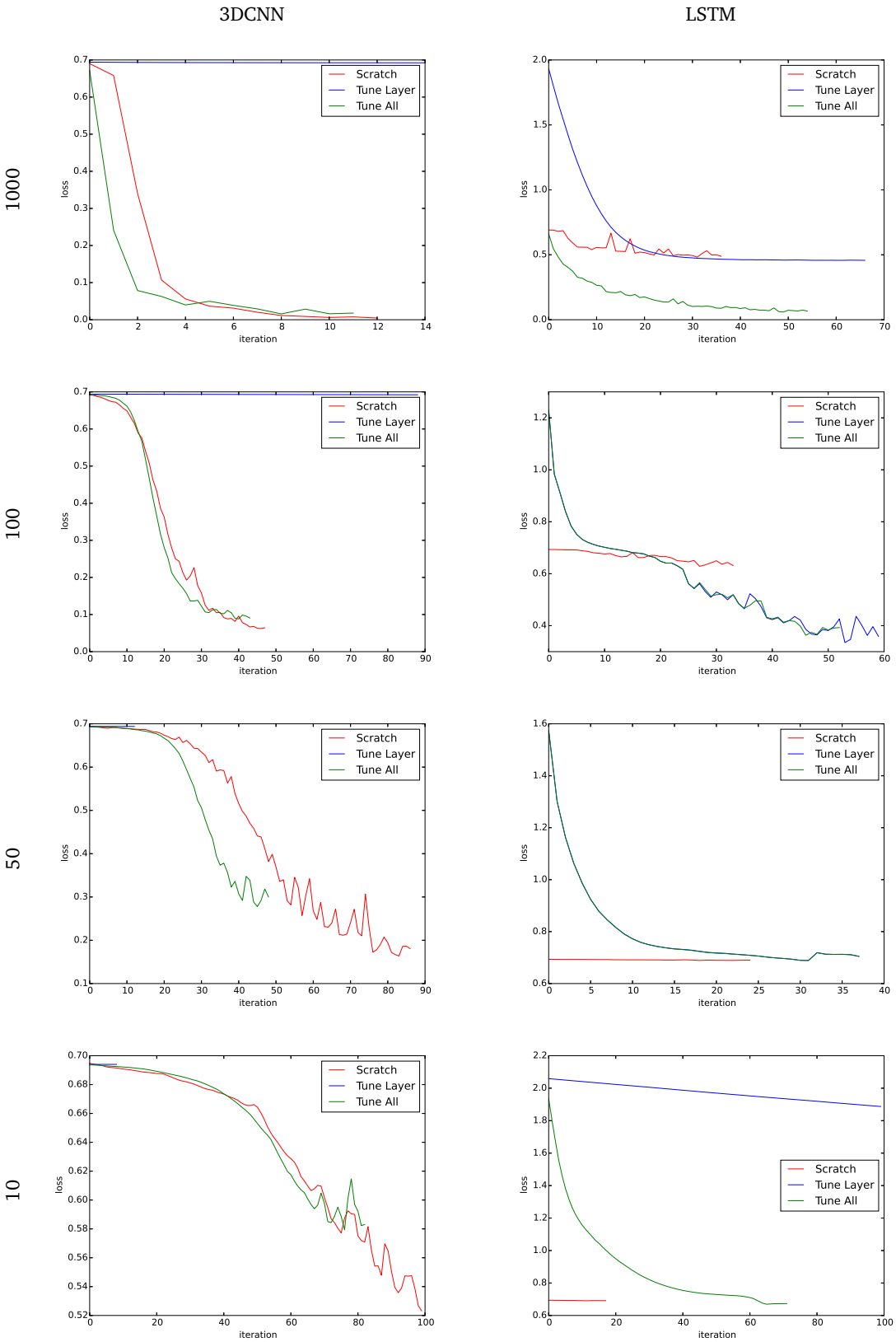

Fig. 5.8: The the decrease in loss for all of the conditions in Experiment 2.1. On the left the training set size per class is indicated, and at the top it is indicated which model is used. Each graph contains the three conditions: Scratch is when a model was trained from scratch, Tune Layer means transfer learning was applied but only the parameters in the last layer were fine-tuned, and Tune All means transfer learning was applied and all parameters in the model were fine-tuned.

6 Discussion

6.1 Findings

The focus of this thesis was to compare the 3DCNN and the LSTM in the context of ALR on several criteria. We conducted two groups of experiments in which the models were compared in several conditions in which they were either trained from scratch or where transfer learning was applied.

First, we compared several 3DCNN and RNN/LSTM models using several optimizers and learning rates. What we found was that all the models seemed to benefit from a relatively higher learning rate. Moreover, both models seemed to benefit from a more complex optimizer. Between SGD, RMSprop, and Adam, the Adam optimizer seemed to work best when comparing the models' performance and training time. Another finding was that between an LSTM and a 3DCNN with a comparable number of parameters, the LSTM is always much faster to train than the 3DCNN. However, the 3DCNN takes fewer epochs to reach its minimum loss value. Thus, in our case, an epoch in an LSTM is a lot faster than an epoch in a 3DCNN with a comparable number of parameters.

Second, we compared the LSTM and the 3DCNN in their capability for transfer learning. We used a 3DCNN and an LSTM trained in the domain of lipreading for transfer learning to a dataset in the same domain, and for transfer learning to a dataset in a different domain, namely the domain of human body movement recognition. In the same domain condition, we also tested what the effect is of transfer learning in the 3DCNN and LSTM when we use a smaller training set. In the case where we performed transfer learning within the same domain, and using the full-size training set, we found that both models training improved when they could make use of the parameters of a pre-trained model as a starting point, but only when the new model was able to fine-tune all parameters and not only the parameters in the last layer. When the parameters of the source model were frozen until the last layer, the target model seemed to be unable to learn anything from the target dataset. When we decreased the training set size, the 3DCNN only benefited from transfer learning when the training set was reduced to 10 samples per class. The LSTM benefited a lot from transfer learning when the training set was of size 100 per class, but not when the training set was decreased further to 50 samples per class.

In the condition where 10 samples per class were used, the LSTM's performance improved slightly. Moreover, when comparing the 3DCNN and LSTM when they were trained from scratch, the 3DCNN performed better in all training set sizes. In the different domain condition, transfer learning did not seem to improve the performance. However, when training a model from scratch, the LSTM performed better than the 3DCNN.

In the first group of experiments, we can see that using SGD as the optimizer caused the models to converge the slowest. Adam and RMSprop were in all cases faster than SGD. This is most likely because both Adam and RMSprop keep track of the average of the past gradients squared, and take this into account in the computation of the weight update. This caused the weight updates to be more drastic than when SGD would have been used, and thus, it caused the loss function to decrease quicker in this case. Between Adam and RMSprop, it is difficult to say which optimizer was faster. In experiment 1.1, RMSprop was faster than Adam at all learning rates. In experiment 1.2, Adam and RMSprop were roughly equally fast when used in the 3DCNN, but Adam was faster when used in the LSTM. There are two main differences between experiment 1.1 and experiment 1.2; the data and models in experiment 1.2 were much larger than the data and models in experiment 1.1, and, the data in experiment 1.2 is data recorded in the natural world, while the data in experiment 1.1 is artificially generated. Both of these differences could contribute to the fact that RMSprop performed better in 1.1 and Adam performed better in 1.2.

Experiment 1.3 was the only experiment of the first group of experiments that used the true size video samples. Comparing the decrease in loss value in the Adam figure in Fig. 5.4 and Fig. 5.6 (in which only the Adam optimizer was used), it can be observed that the patterns are similar. However, the LSTM's loss value does not reach the same minimum value the 3DCNN does. This is reflected in the test set performance in Table 5.3. In Table 5.2, it can already be seen that, overall, the LSTM performs slightly worse on the test set, while the 3DCNN takes longer to train. In experiment 1.3, these effects seem to be magnified.

Going by the results in the first experiment, it is possible that there exists a trade-off between performance and computational cost. The LSTM trains faster but performs worse on the test set, while the 3DCNN performs better but takes longer to train. Our experiments were conducted on small datasets, but the effect is visible. We expect this trade-off effect will most likely be magnified in the case where larger datasets are processed. The answer to the first research question, however, can be found in experiment 1.2 most clearly. And specifically, in Table 5.2 and Fig. 5.4. For the sake of answering the first research question, we assume that the range of test set accuracies indicated in Table 5.2 (98-100%) counts as a 'similar accuracy', the constraint imposed by the first research question. In experiment 1.2, the LSTM is

between 2.8 and 5.8 times faster than the 3DCNN. The LSTM is even 12.2 times faster than the 3DCNN in experiment 1.3, but the difference in test set accuracy is slightly larger (95% and 99%).

The reason why this difference in training time exists between an LSTM and a 3DCNN with a comparable number of parameters is the difference in the number of operations in these models in the calculations of the gradients of the loss function in relation to the number of parameters of the models. In general, computing the gradients in a 3D convolutional layer with an arbitrary number of parameters involved more computations than computing the gradients in an LSTM layer with the same number of parameters. This is due to the nature of the mechanisms within these models. The results of this experiments provide an answer as to how large this difference can be in practice.

Looking at Experiments 2, and specifically Table 5.4, we can see that the 3DCNN performs better in all conditions where the models were trained from scratch. The dataset sizes go from 1000 to 100, to 50, to 10. And, the 3DCNN's and LSTM's test set performance goes from 97% to 93%, to 91%, to 76%; and from 75% to 63%, to 62%, to 50%; respectively. In this experiment specifically, the 3DCNN only needs 10 samples per class, where the LSTM needs 1000 samples per class, to achieve a similar performance. This suggests that the LSTM requires quite a lot more data than the 3DCNN to achieve a similar test set performance. Further experiments with larger training sets would be needed to see if the LSTM could achieve the same result as the LSTM in this case and if this finding can be generalized. However, these results support and strengthen the findings in Experiments 1, namely that the 3DCNN performs better than the LSTM in the context of ALR.

Looking at the transfer learning experiments in experiment 2.1, we can see that the 3DCCN does not benefit at all when only the last layer of a pre-trained model can be fine-tuned. The LSTM, however, does seem to benefit from this, but only when the training set was large enough. Only in the experiments where the training set size was 100 or greater did the LSTM benefit from transfer learning. This suggests that in these experiments, the features extracted by the 3DCNN were not general enough to be transferred to other classes in the same dataset, while the features extracted by the LSTM were. However, the LSTM using a pre-trained model and that was able to fine-tune the last layer still performed quite a lot worse than a 3DCNN that was trained from scratch. Again, these results support the previous findings that the 3DCNN is better suited for ALR.

When the models were able to fine-tune all parameters in the network and not just the parameters in the last layer, in all cases the models performed better or comparable than its versions trained from scratch. The LSTM using transfer learning

and fine-tuning all parameters even performed as well as the 3DCNN when the full training set of size 1000 was used. However, this was not the case where a smaller training set was used. In the small training set sizes of 50 and 10, the LSTM was not able to benefit from transfer learning very much, or not at all. Using a pre-trained 3DCNN on training set sizes of 10 even performed better than all LSTMs in the training sets sizes of 100, 50, and 10. This shows that the 3DCNN is much better suited than the LSTM in ALR, and especially when the training set is very small.

Finally, in experiment 2.2 we also tested the suitability of the models for transfer learning to other domains. We used the same methodology as in experiment 2.1, except for the training set size reductions. What we found was that transfer learning did not improve the performance of either model. The main reason for this is most likely that the features extracted by the pre-trained models were not general enough to be used in the new domain. A noticeable difference between the data in the source domain and the data in the target domain was that the data in the source domain was very 'clean' (Fig. 5.1). The way we cut out the mouths from the original video resulted in the videos only containing the mouths, and thus there was relatively little noise in the video samples. There was a lot more noise in the samples of each class in the target dataset(Fig. 5.2), compared to the source dataset. A pre-trained model trained on a dataset containing more noise could possibly have improved the baseline performance in this specific experiment.

What was noticeable about the results in experiment 2.2 was that the LSTM achieved a higher test set performance than the 3DCNN overall. This is the only experiment in this thesis in which this occurred. The only difference between this experiment and the previous experiments is the data used, which suggests the 3DCNN performs better in data with different characteristics than the LSTM. As said before, the dataset used in experiment 2.2 contained more noise than the dataset used before. It might be that the LSTM is more suited for this type of data. However, given the fact that this was not one of the main focuses of this thesis, this conclusion can not be drawn solely from this experiment.

6.2 Conclusions

From the experiments conducted in this thesis, we can conclude that the 3DCNN seems to be better suited for video classification in the context of ALR than the LSTM. Generally, the 3DCNN achieves a lower loss value and a higher test set accuracy. There were a couple of experiments where the LSTM performed as well as the 3DCNN, but the LSTM did not perform better than the 3DCNN on any subset of the LRW dataset. These findings are likely to be extrapolated to larger datasets

with more data per class and/or more classes, but further research would provide a more elaborate answer to this. Because the 3DCNN did not outperform the LSTM in experiment 2.2, in which a model was trained on a different domain than ALR, we will only conclude that the 3DCNN is better suited than the LSTM for video classification in the domain of ALR, and not in video classification in general.

Aside from the findings coming forth from our main focus, we also draw several other conclusions. It seems that the Adam optimizer performed better than SGD and RMSprop. RMSprop and Adam were comparable in general, and RMSprop was faster than Adam in some conditions. However, in experiment 1.2, in which downsized LRW data samples were used, Adam and RMSprop had comparable performance, but the LSTM trained slightly faster using the Adam optimizer. Another result we found rather unexpected was the fact that parameter value transfer learning in which only the last layer could be retrained was counterproductive in nearly all conditions. Even in the situation where the difference between the source and target data was that they were different classes from the same larger dataset, the features learned were not general enough to be transferred to the different classes.

A limitation of this work is that the datasets used were relatively small. As a suggestion for further research, using bigger datasets for comparison of the 3DCNN and LSTM would provide a better answer to which model is best suited for video classification in the domain of ALR and other domains. Moreover, the models' capability for transfer learning has been tested in this paper, but more elaborate experiments would provide a more generalizable answer as well for this point. Moreover, it would be useful to use pre-trained models that were trained on larger datasets, and perform parameter value transfer to models trained on larger datasets as well, to be able to answer which of the models is better capable of transfer learning. This is because the pre-trained models were trained in experiment 1.3, and used as a source model for parameter value transfer in experiment 2.1 and experiment 2.2, were relatively small. It is likely that transfer learning did not produce many noticeable results because the source models were not extracting very general features to begin with.

Bibliography

[1] Ibrahim Almajai, Stephen Cox, Richard Harvey, and Yuxuan Lan. „Improved speaker independent lip reading using speaker adaptive training and deep neural networks“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 2722–2726 (cit. on p. 5).

[2] Yannis M Assael, Brendan Shillingford, Shimon Whiteson, and Nando de Freitas. „LipNet: end-to-end sentence-level lipreading“. In: (2016) (cit. on p. 11).

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on p. 33).

[4] Xudong Cao, David Wipf, Fang Wen, Genquan Duan, and Jian Sun. „A practical transfer learning algorithm for face verification“. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3208–3215 (cit. on p. 2).

[5] François Chollet. *Keras*. `https://github.com/fchollet/keras`. 2015 (cit. on p. 20).

[6] Joon Son Chung and Andrew Zisserman. „Lip reading in profile“. In: BMVC. 2017 (cit. on p. 11).

[7] Joon Son Chung and Andrew Zisserman. „Lip reading in the wild“. In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 87–103 (cit. on pp. 11, 39).

[8] Joon Son Chung, Andrew Senior, Oriol Vinyals, and Andrew Zisserman. „Lip reading sentences in the wild“. In: *arXiv preprint arXiv:1611.05358* 2 (2016) (cit. on pp. 11, 12).

[9] Martin Cooke, Jon Barker, Stuart Cunningham, and Xu Shao. „An audio-visual corpus for speech perception and automatic speech recognition“. In: *The Journal of the Acoustical Society of America* 120.5 (2006), pp. 2421–2424 (cit. on pp. 11, 13).

[10] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. „Boosting for transfer learning“. In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 193–200 (cit. on p. 35).

[11] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. „Self-taught clustering“. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 200–207 (cit. on p. 35).

[12] Hal Daumé III. „Frustratingly easy domain adaptation“. In: *arXiv preprint arXiv:0907.1815* (2009) (cit. on p. 35).

[13]Jia Deng, Wei Dong, Richard Socher, et al. „Imagenet: A large-scale hierarchical image database“. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255 (cit. on p. 27).

[14]Ezequiel A Di Paolo. „Organismically-inspired robotics: homeostatic adaptation and teleology beyond the closed sensorimotor loop“. In: *Dynamical systems approach to embodiment and sociality* (2003), pp. 19–42 (cit. on p. 16).

[15]Barbara Dodd. „Lip reading in infants: Attention to speech presented in-and out-of-synchrony“. In: *Cognitive psychology* 11.4 (1979), pp. 478–484 (cit. on p. 6).

[16]Ariel Ephrat, Tavi Halperin, and Shmuel Peleg. „Improved speech reconstruction from silent video“. In: *ICCV 2017 Workshop on Computer Vision for Audio-Visual Media*. 2017 (cit. on p. 13).

[17]Ariel Ephrat, Tavi Halperin, and Shmuel Peleg. *Vid2Speech: Speech Reconstruction from Silent Video*. `http://www.vision.huji.ac.il/vid2speech/`. [Online; accessed 03-March-2018]. 2017 (cit. on p. 13).

[18]Richard J Evans. *The third Reich in history and memory*. Oxford University Press, USA, 2015 (cit. on p. 6).

[19]Kathleen E Finn and Allen A Montgomery. „Automatic optically-based recognition of speech“. In: *Pattern Recognition Letters* 8.3 (1988), pp. 159–164 (cit. on pp. 7, 8).

[20]Kunihiko Fukushima. „Neocognitron: A hierarchical neural network capable of visual pattern recognition“. In: *Neural networks* 1.2 (1988), pp. 119–130 (cit. on p. 26).

[21]Garrett B Goh, Nathan O Hodas, and Abhinav Vishnu. „Deep learning for computational chemistry“. In: *Journal of computational chemistry* (2017) (cit. on p. 27).

[22]Alex Graves, Marcus Liwicki, Santiago Fernández, et al. „A novel connectionist system for unconstrained handwriting recognition“. In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2009), pp. 855–868 (cit. on p. 12).

[23]Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. „Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks“. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 369–376 (cit. on p. 12).

[24]Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. „Speech recognition with deep recurrent neural networks“. In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE. 2013, pp. 6645–6649 (cit. on p. 12).

[25]Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Delving deep into rectifiers: Surpassing human-level performance on imagenet classification“. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034 (cit. on p. 1).

[26]Sarah Hilder, Richard W Harvey, and Barry-John Theobald. „Comparison of human and machine-based lip-reading.“ In: *AVSP*. 2009, pp. 86–89 (cit. on p. 7).

[27]Sepp Hochreiter and Jürgen Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 33).

[28]Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. „Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 7304–7308 (cit. on p. 37).

[29]David H Hubel and Torsten N Wiesel. „Receptive fields and functional architecture of monkey striate cortex“. In: *The Journal of physiology* 195.1 (1968), pp. 215–243 (cit. on pp. 25, 26).

[30]Sergey Ioffe. „Probabilistic linear discriminant analysis“. In: *European Conference on Computer Vision*. Springer. 2006, pp. 531–542 (cit. on p. 12).

[31]Koji Iwano, Satoshi Tamura, and Sadaoki Furui. „Bimodal speech recognition using lip movement measured by optical-flow analysis“. In: *International Workshop on Hands-Free Speech Communication*. 2001 (cit. on p. 10).

[32]Katarzyna Janocha and Wojciech Marian Czarnecki. „On loss functions for deep neural networks in classification“. In: *arXiv preprint arXiv:1702.05659* (2017) (cit. on p. 20).

[33]Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. „3D convolutional neural networks for human action recognition“. In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–231 (cit. on pp. 11, 27, 29).

[34]James W Kalat. *Biological psychology*. Cengage Learning, 1980 (cit. on p. 15).

[35]Andrej Karpathy, George Toderici, Sanketh Shetty, et al. „Large-scale video classification with convolutional neural networks“. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732 (cit. on pp. 11, 35, 36).

[36]Alireza Khotanzad and J-H Lu. „Classification of invariant image representations using a neural network“. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38.6 (1990), pp. 1028–1038 (cit. on p. 16).

[37]Diederik P Kingma and Jimmy Ba. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 23).

[38]Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 27).

[39]Peter Lennie. „Single units and visual cortical organization“. In: *Perception* 27.8 (1998), pp. 889–935 (cit. on p. 15).

[40]Xuejun Liao, Ya Xue, and Lawrence Carin. „Logistic regression with an auxiliary data source“. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 505–512 (cit. on p. 35).

[41]Frederick N Martin and John Greer Clark. *Introduction to audiology*. Allyn and Bacon Boston, 1997 (cit. on p. 5).

[42]Harry McGurk and John MacDonald. „Hearing lips and seeing voices“. In: *Nature* 264.5588 (1976), p. 746 (cit. on p. 6).

[43]Youssef Mroueh, Etienne Marcheret, and Vaibhava Goel. „Deep multimodal learning for audio-visual speech recognition“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE. 2015, pp. 2130–2134 (cit. on p. 6).

[44]Thien Huu Nguyen and Ralph Grishman. „Relation extraction: Perspective from convolutional neural networks“. In: *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. 2015, pp. 39–48 (cit. on p. 27).

[45]Yukio Nishimura, Hirotaka Onoe, Yosuke Morichika, et al. „Time-dependent central compensatory mechanisms of finger dexterity after spinal cord injury“. In: *Science* 318.5853 (2007), pp. 1150–1155 (cit. on p. 16).

[46]Kuniaki Noda, Yuki Yamaguchi, Kazuhiro Nakadai, Hiroshi G Okuno, and Tetsuya Ogata. „Audio-visual speech recognition using deep learning“. In: *Applied Intelligence* 42.4 (2015), pp. 722–737 (cit. on p. 6).

[47]Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. „Learning and transferring mid-level image representations using convolutional neural networks“. In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE. 2014, pp. 1717–1724 (cit. on pp. 2, 36).

[48]Sinno Jialin Pan and Qiang Yang. „A survey on transfer learning“. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359 (cit. on pp. 2, 35).

[49]Eric Petajan, Bradford Bischoff, David Bodoff, and N Michael Brooke. „An improved automatic lipreading system to enhance speech recognition“. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 1988, pp. 19–25 (cit. on pp. 8, 9).

[50]Stavros Petridis, Themos Stafylakis, Pingchuan Ma, et al. „End-to-end Audiovisual Speech Recognition“. In: *arXiv preprint arXiv:1802.06424* (2018) (cit. on p. 13).

[51]Stavros Petridis, Yujiang Wang, Zuwei Li, and Maja Pantic. „End-to-End Multi-View Lipreading“. In: *arXiv preprint arXiv:1709.00443* (2017) (cit. on p. 11).

[52]Prajit Ramachandran, Barret Zoph, and Quoc V Le. „Searching for activation functions“. In: (2018) (cit. on p. 19).

[53]Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. „Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017) (cit. on p. 6).

[54]Ahmed Rekik, Achraf Ben-Hamadou, and Walid Mahdi. „A new visual speech recognition approach for RGB-D cameras“. In: *International Conference Image Analysis and Recognition*. Springer. 2014, pp. 21–28 (cit. on p. 11).

[55]Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 16).

[56]Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *arXiv preprint arXiv:1609.04747* (2016) (cit. on p. 23).

[57]Olga Russakovsky, Jia Deng, Hao Su, et al. „ImageNet Large Scale Visual Recognition Challenge“. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252 (cit. on p. 27).

[58]Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. „300 faces in-the-wild challenge: The first facial landmark localization challenge“. In: *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE. 2013, pp. 397–403 (cit. on p. 6).

[59]Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. „A semi-automatic methodology for facial landmark annotation“. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*. IEEE. 2013, pp. 896–903 (cit. on p. 6).

[60]Cicero dos Santos and Maira Gatti. „Deep convolutional neural networks for sentiment analysis of short texts“. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. 2014, pp. 69–78 (cit. on p. 27).

[61]David Silver, Aja Huang, Chris J Maddison, et al. „Mastering the game of Go with deep neural networks and tree search“. In: *nature* 529.7587 (2016), pp. 484–489 (cit. on pp. 3, 27).

[62]Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. „UCF101: A dataset of 101 human actions classes from videos in the wild“. In: *arXiv preprint arXiv:1212.0402* (2012) (cit. on p. 40).

[63]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: A simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on p. 25).

[64]Themos Stafylakis and Georgios Tzimiropoulos. „Combining residual networks with LSTMs for lipreading“. In: *arXiv preprint arXiv:1703.04105* (2017) (cit. on pp. 12, 40).

[65]Themos Stafylakis and Georgios Tzimiropoulos. „Deep word embeddings for visual speech recognition“. In: *arXiv preprint arXiv:1710.11201* (2017) (cit. on p. 12).

[66]Ilya Sutskever, Oriol Vinyals, and Quoc V Le. „Sequence to sequence learning with neural networks“. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112 (cit. on p. 12).

[67]Kwanchiva Thangthai, Richard W Harvey, Stephen J Cox, and Barry-John Theobald. „Improving lip-reading performance for robust audiovisual speech recognition using DNNs.“ In: *AVSP*. 2015, pp. 127–131 (cit. on p. 6).

[68]Simon J Thorpe and Michel Imbert. „Biological constraints on connectionist modelling“. In: *Connectionism in perspective* (1989), pp. 63–92 (cit. on p. 16).

[69]Tijmen Tieleman and Geoffrey Hinton. „Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude“. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31 (cit. on p. 23).

[70]Michael Wand and Jürgen Schmidhuber. „Improving Speaker-Independent Lipreading with Domain-Adversarial Training“. In: *arXiv preprint arXiv:1708.01565* (2017) (cit. on p. 13).

[71]Gregory J Wolff, K Venkatesh Prasad, David G Stork, and Marcus Hennecke. „Lipreading by neural networks: Visual preprocessing, learning, and sensory integration“. In: *Advances in neural information processing systems*. 1994, pp. 1027–1034 (cit. on p. 9).

[72]Yue Wu, Tal Hassner, KangGeon Kim, Gerard Medioni, and Prem Natarajan. „Facial landmark detection with tweaked convolutional neural networks“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017) (cit. on p. 6).

[73]Kelvin Xu, Jimmy Ba, Ryan Kiros, et al. „Show, attend and tell: Neural image caption generation with visual attention“. In: *International Conference on Machine Learning*. 2015, pp. 2048–2057 (cit. on p. 27).

[74]Li Yao, Atousa Torabi, Kyunghyun Cho, et al. „Describing videos by exploiting temporal structure“. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4507–4515 (cit. on p. 27).

[75]Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. „How transferable are features in deep neural networks?“ In: *Advances in neural information processing systems*. 2014, pp. 3320–3328 (cit. on pp. 28, 36, 37).

[76]Richard Zhang, Phillip Isola, and Alexei A Efros. „Colorful image colorization“. In: *European Conference on Computer Vision*. Springer. 2016, pp. 649–666 (cit. on p. 27).

[77]Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. „Facial landmark detection by deep multi-task learning“. In: *European Conference on Computer Vision*. Springer. 2014, pp. 94–108 (cit. on p. 6).

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.