KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Design, implementation and analysis of a deep convolutional-recurrent neural network for speech recognition through audiovisual sensor fusion

Matthijs Van keirsbilck

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor:
Prof. dr. ir. Marian Verhelst

Assessoren:
Prof. dr. ir. W. Dehaene
Prof. dr. ir. H. Van hamme

Begeleider:
Ir. B. Moons

Academiejaar 2016 – 2017

Preface

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text. My gratitude goes to my family and friends, who helped and supported me throughout the year.

Matthijs Van keirsbilck

Contents

Abstract

Automatic speech recognition systems often only rely on audio and don't perform well under noisy conditions. Adding visual information could improve performance. Research on audio-visual speech recognition is now possible thanks to large-scale audio-visual datasets like TCDTIMIT that have become publicly available. The recent successes of Convolutional Neural Networks on standard image classification benchmarks suggest that they can be used for lipreading.

Three Convolutional Neural Network architectures are compared for lipreading with the performance-complexity trade-off in mind. For video processing, adding a recurrent network on top of the CNN can improve performance significantly compared to CNN-only networks.

Different networks are evaluated for audio-only speech recognition. Deep Bidirectional LSTM networks of 2 to 3 layers with 256 to 512 LSTM units per layer shows the best performance-complexity trade-off.

Several methods to combine the output features from the audio and lipreading networks are analyzed. The noise performance of the audio-visual network is significantly higher than for an audio-only network, both for high and low audio SNR. Adding an attention network further improves performance for noisy audio.

The trained neural networks perform better than the TCDTIMIT baseline for audio-only, visual-only and audio-visual classification. Previous audio-visual baseline results were 59% and 44% for clean and noisy audio (white noise, 0dB SNR) respectively. The audio-visual neural networks achieve 75.70% and 58.55% correctness.

List of Abbreviations

Abbreviations

(A)SR	(Automatic) Speech Recognition
ANN	Artificial Neural Network
AN	Attention network
NN	Neural Network
FCNN	Fully Connected Neural Network
DNN	Dense Neural Network (=FCNN)
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
CNN-LSTM	a sequence of CNNs connected to a LSTM network
RF	Raw features (eg directly connecting CNN to LSTM)
PF	pre-classified features (eg FC classification layer between CNN and LSTM)
E2E	End-to-end training
MFCC	Mel-Frequency Cepstral Coefficients
CE	Cross- Entropy (loss)
MSE	Mean Square error

Networks are indicated by their type, the number of units per layer N and the number of layers k as follows: 'type: N / k'. Networks that are composed of two other networks indicate this with at '+'. Their name also indicates whether or not there was a pre-classification layer after the first of the two networks. A lipreading network composed of a CNN and a multilayer LSTM network, where pre-classification is used after the CNN is indicated with 'Lip: CNN (PF) + 256 / 2 LSTM'.

When several such networks are combined (for example an audio and a lipreading network), the same principles are applied in a hierarchical fashion. An audiovisual network composed of an '256 / 2' audio network and an 'WLAS (RF) + 64 / 1' lipreading network where the audio network is pre-classified but the lipreading network is not, is indicated with 'Multimodal: Audio (PF) 256 / 2 + Lip (RF): WLAS (RF) + 64 / 1'.

Chapter 1

Introduction

1.1 Motivation and goal

Over the past years, there has been a lot of interest in the domain of speech recognition, thanks to increased processing power, big data and advances in machine learning. Speech provides for a much more natural interaction of users with technology than other human-computer interaction methods like typing. This makes products more attractive, enhances the user experience and can increase productivity. It also allows for hands-free operation of devices, something that can be valuable in certain areas. Many companies are currently introducing these systems in their products, most notably Siri from Apple, Cortana from Microsoft, the Amazon Echo and many more.

Speech recognition systems have existed for some time, first using heuristic rules, later using statistical approaches. Recently there has been a boost to performance and ease of use thanks to better machine learning algorithms that made use of neural networks. This was possible thanks to advances in the hardware that runs these algorithms. Progress of GPU's especially has provided a large boost to interest in the area by making training and developing neural network models much faster and easier. Lack of computational resources is a main reason why neural networks haven't been widely used after their introduction for speech in the '80s.

Recurrent neural networks (RNNs) especially have made a large impact in the field of speech recognition, largely because they require much less manual work than previous models. Recently the introduction of LSTM cells instead of conventional neurons has further improved RNN performance (see Chapter 3). Most of the research and industry has focused on using audio data as input to the speech recognition system, as audio data is tightly linked to the spoken sentences.

Simultaneously, much progress has been made in the field of image recognition systems. Convolutional networks are networks optimized to extract useful information from images. Deep networks are networks containing many layers of neurons stacked on top of each other. For an extensive discussion of these networks, refer to Chapter 4. Combining these two innovations resulted in 'deep convolutional neural networks'. These networks have improved recognition performances on the standard ImageNet dataset, up to the point that classifiers are now achieving human performance and

even surpassing it.

In many practical situations, it is not easy or even feasible to obtain high-quality audio that can be used to achieve high accuracies in speech recognition. Therefore, recognition accuracy suffers in noisy circumstances and due to unwanted audio interference. Examples include traffic situations, when loud music is playing or other people are talking close by the speaker.

A solution to this problem is to use another source of input data apart from the audio, for example visual information. Humans are also known to use visual information to learn what another person has said. This is called 'lipreading'. Especially hearing-impaired people make use of this, as they mostly or only rely on visual information. Using these new convolutional neural networks to perform automatic lipreading can add more information to the recognition system than audio only can provide, thus allowing it to reach higher performances. This is especially the case in noisy environments.

The long-term goal is to be able to implement all of these algorithms on low-power, low-cost embedded hardware so the recognition systems can be mass-produced and for example implemented in smartphones, smart glasses or other wearables. There they can assist hearing impaired people in particular, but also other people in their daily lives.

1.2 This work

This thesis discusses the design, implementation and analysis of a network architecture performing automatic speech recognition by combining visual and auditive information. This technique is called 'audio-visual sensor fusion'. The goal is to improve audio recognition performance, especially in case of low audio quality.

For audio-visual recognition, the dataset TCDTIMIT [1] will be used. Information about the database and preprocessing of the data can be found in Appendix C.

The basis for a speech recognition system is the recognition of phonemes, the building blocks of any language. Phonemes are the most basic sounds a human vocal system can produce, and by combining these phonemes words are formed. When these phonemes are predicted, higher-level models of language can be used to convert the phonemes to existing words, and sentence-level models can be used to further improve recognition performance by exploiting knowledge of grammar, sentence structures and other information.

A system that recognizes phonemes is essentially language-independent, depending for its application to multiple languages only on the data provided to train it. As phoneme recognition is the most essential part of any SR system, this work focuses on that topic.

Chapter 2 gives an introduction to neural networks. It explains why they are used, how they work, and how they are trained. For different applications, specialized network architectures have been invented, and these are discussed as well.

Chapter 3 will give an overview of the state of research in audio recognizers, and performances of these systems will be analyzed on the industry-standard database

TIMIT and on a new audio-visual database, TCDTIMIT.

In chapter 4, an overview of lipreading systems will be given, followed by the implementation using several convolutional neural networks, as well as a further improvement by combining convolutional networks and recurrent networks.

After both audio and lipreading have been discussed separately, chapter 5 discusses how to combine both types of networks for audio-visual speech recognition. The multimodal network is then optimized for good performance for noisy audio by performing end-to-end training and adding an attention network.

In order for these systems to be useful in practice, some optimizations and compromises will need to be made. These are discussed throughout this text in the appropriate chapters.

The best multimodal network with attention mechanism and end-to-end training achieve a phoneme recognition correctness score on the TCDTIMIT lipspeakers of 74.60%, compared to 59% for the baseline result [1]. For an audio SNR of 0dB, the score is 58.55% compared to 44% for the baseline results. This makes these results the current state-of-the-art for the TCDTIMIT dataset.

Chapter 2

Deep Neural Networks

2.1 Introduction

Artificial Neural Networks are based on the structure of neurons in our own brains. They take some input values, perform a very simple operation on it, and pass their output to other neurons. Several of these neurons combined can form very complex input-output mappings, defined by the operation that is performed by each neuron on its inputs.

These types of networks have gained a lot of popularity in recent years, and increased state-of-the-art performance in many pattern recognition systems.

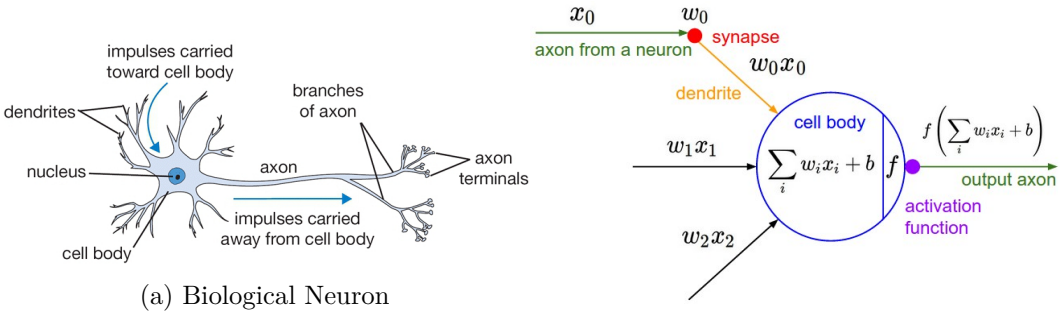
This chapter gives an overview of what Neural Networks are, as well as some important innovations in neural network architectures. Details of the architectures used in this work for audio recognition can be found in Chapter 3 and for lipreading in Chapter 4.

- What is an artificial neuron?
- Network architecture: number and type of layers, number of neurons per layer, possibly combining different network architectures.
- Training: libraries, optimization method and cost function, overfitting, learning rate, dataset
- Dataset: data size, train/test/validation split, preprocessing.

2.2 Artificial Neurons

An artificial neuron is a simplified version of the biological neuron. As mentioned above, the goal is to allow the neuron to approximate any continuous function. The simplest way to approximate the functionality of biological neurons is to multiply each input value with a certain weight and accumulating the results (each neuron performs a MACC operation on its inputs and connection weights). Different values of the weights alter the shape of the neuron's output function (see Figure 2.2a).

A bias value is added for each input in order to be able to shift the output curve of the neuron left and right. This gives the neuron more freedom to approximate



(a) Biological Neuron

(b) Artificial Neuron

Figure 2.1: Comparison between biological and artificial neurons

functions (see Figure 2.2b).

Before passing the output to the next neuron, it is acted upon by an 'activation function'. The reason to do this is to make the neuron nonlinear. Adding nonlinearities is essential in order for neural networks to learn nonlinear input-output relationships. Without it, only nonlinear functions can be represented, which is very limited. The most common activation functions are:

- Sigmoid $sigmoid(x) = 1/(1 + e^{-x})$
- Tanh $tanh(x) = 2\sigma(2x) - 1$
- ReLu (rectifier linear unit): $relu(x) = \max(0, x)$

In practice, the ReLu activation function is the simplest and also performs the best for large networks, so it is generally recommended to use that function [2].

The mathematical formulation of a neuron is shown in Equation 2.1. First, a linear transformation by the weight matrix W is applied. The results are translated by adding the bias vector, and then the activation function is applied point-wise.

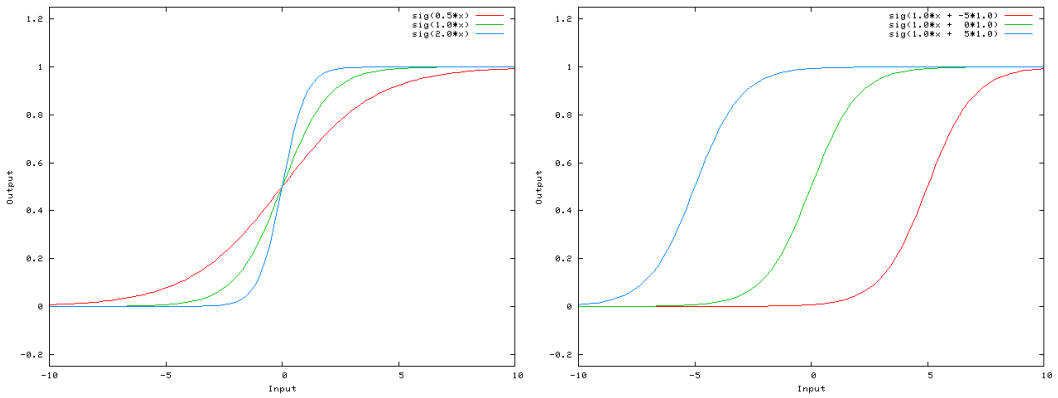
$$f(x, W, b) = activation(W * x + b) \tag{2.1}$$

x = N-dimensional input vector
W_i = N-dimensional weight vector
b = bias value
$activation$ = the activation function

2.3 Fully Connected Networks

A single neuron is of little value, but many neurons combined in certain ways can perform very complex data processing tasks. Several architectures have been proposed that have been designed with a certain function in mind. This section gives an overview of different possible architectures.

In Fully Connected feed-forward Neural Networks (FCNNs) each neuron is connected with each neuron in the layer after it. Each neuron's output is propagated



(a) Changing a weight changes the output curve's shape

(b) Changing the bias changes the output curve's position [3]

Figure 2.2: Output of an Artificial Neuron with sigmoid activation function when network parameters are changed

to a neuron that is closer to the network output. The neurons all 'point' towards the network output, hence the name 'feed-forward'.

This type of networks has a large amounts of weights and can learn complex relationships between inputs and outputs. These networks are hard to use for end-to-end classification as the network size explodes when the number of layers and number of neurons per layer increases. Therefore they are often used after a feature extraction network. They then combine the features in some nonlinear way and classify them.

Neural networks always have some input neurons and some output neurons. Neurons in between input and output layers are said to belong to 'hidden' layers, as they don't interact directly with the environment.
It can be proven that a feed-forward network with one hidden layer can approximate any continuous input-to-output mapping function under mild assumptions, making this very attractive for complex classification tasks like speech recognition. [4]

For performance measurement classification tasks, top-k accuracy is used as a metric. For each input sample, the k predictions of the network with the highest probability score are compared with the real label. If one of those k predictions is correct, the evaluation is counted as correct. If not, it is counted as wrong. The final top-k accuracy score is then the fraction of samples that were predicted correctly compared to the total number of input samples.

An example network is shown in Figure 2.3.

2.3.1 Deep Neural Networks

Although networks with just one hidden layer can theoretically represent any continuous function, learning the correct weights and biases to do so is a big problem in practice. An idea that has helped alleviate this issue is to stack several layers

of neurons on top of each other, creating more hidden layers. This enables the network to learn more complex input-output mapping functions, thus improving performance.[2] FCNNs that have many layers are often called 'Deep Neural Networks' (DNN) (Figure 2.3). Another term is 'stacked' networks.

This technique has improved FCNNs, and is also very useful for other networks like convolutional (see Section 2.4) as well as recurrent networks (see Section 2.4) . For an overview on deep learning, see [5] and [6]. Almost all image processing network are very deep [7] [8], and many speech recognition networks are also showing improved performance using deep networks [9][10].

As mentioned above, feed-forward networks with one layer can theoretically represent any continuous input-to-output mapping but have trouble learning this in practice. Stacking several layers on top of each other can significantly improve performance. The networks can learn hierarchical representations of the input, crude in the first layers, but ever more recognizable in the higher layers.

As networks get deeper, they become harder to train as well. A general technique to remedy this is to use 'batch normalization' [11]. For CNNs special 'residual' networks have been introduced to deal with this issue (see Section 2.5) [7].

2.4 Incorporating Time information: Recurrent Neural Networks

An issue with standard feed-forward networks is that they can't take into account time information. This is however extremely important for speech recognition. When a person has pronounced 'c' and 'a', the next letter might very well be an 'r'. The probability of it being 'x' is quite low.

Recurrent Neural Networks (RNNs) can use this time information by adding a kind of memory to the network in the form of internal states. These states are created by adding connections from neurons to themselves or to other neurons within the same layer. Apart from these connections, RNNs are very similar to standard feedforward neural networks (see Figure 2.3 for a comparison). Each layer of an RNN consists of a certain number of units ('neurons'). The first layer of an RNN is fully connected to the features of the input, like a normal feedforward network. Subsequent layers take the output features from previous layers as their inputs. The last layer is fully connected to output neurons for classification. See Figure 2.3.

An RNN processes a sequence of inputs (for example a sequence of words (a sentence) instead of isolated inputs. The elements of the sequence are processed one-by-one, but after processing one element the network stores information about that element in its internal state variables. Predictions for following elements can then be improved by using this stored information. This is described in Equation 2.2The network gathers more information about the sequence while it is processing it. The internal state of the network can be seen as an additional input variable next to the inputs from the sequence. Using this idea, the network can be 'unfolded' in time to get rid of the recurrent connections (see Figure 2.4). The input frames x_t at timestep t are for example the frames in a video or the MFCC frames in a

preprocessed audio file. An illustration of the processing of an input sequence is shown in Figure 2.5. In the figures 2.5 to 2.8, each gray block denotes one layer of neurons.

After they are 'unrolled in time' according to Equation 2.2, RNNs can be trained very similiarly to feedforward networks. The backpropagation algorithm can be used, although some modifications have to be made to incorporate the unrolling step. This is called 'Back propagation through time (BTT) [12].

Like feedforward networks, RNNs can use several hidden layers stacked on top of each other (see Figure 2.6) to form a 'deep RNN'. The input sequence is provided as input to the first layer; the inputs of subsequent layers are the output features of the previous layer. The number of output features of a layer is equal to the number of neurons in that layer.

$$h_i, p_i = f(h_{i-1}, x_i) \tag{2.2}$$

x_i = i'th element of the input sequence
h_i = internal state of the network after processing x_i
p_i = network output (prediction) for input x_i

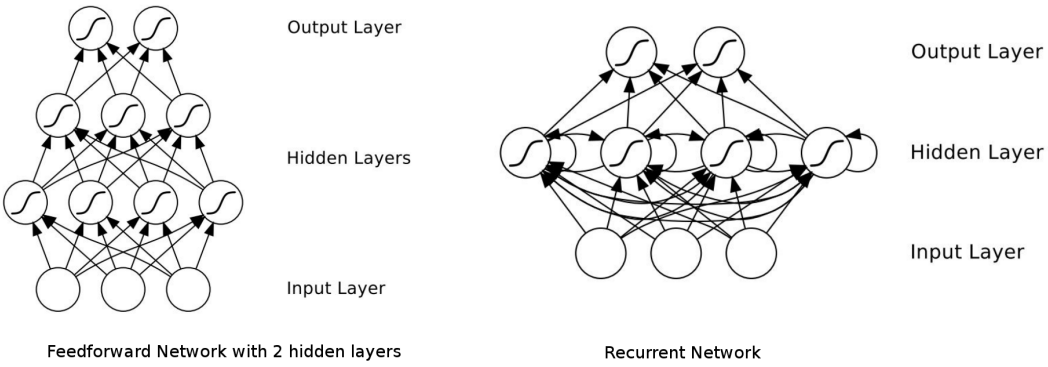


Figure 2.3: Feedforward and recurrent networks. Both are fully connected, but the feedforward neurons only have connections to neurons in the next layer. The RNN has connections within one layer as well. In figures 2.4 to 2.8, one (hidden) layer of neurons of an RNN is denoted by a gray box.

2.4.1 Bidirectional Networks

Bidirectional RNNs use the fact that the output at a certain time does not only depend on previous inputs, but also future elements. They are essentially just two identical RNNs stacked on top of each other, but traversing the input sequence in opposite directions (see Figure 2.7). One 'bidirectional layer' is then composed of forward and a backward layer, whose output features are added together. This happens simultaneously, and the network classification depends on both layers. In

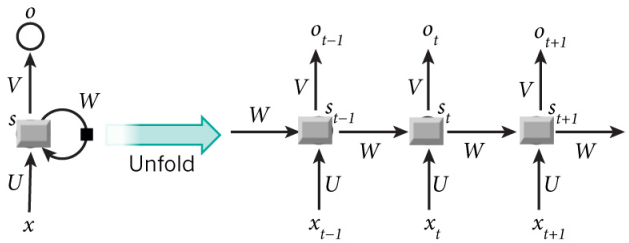


Figure 2.4: RNN unrolling in time: the left figure denotes that network has recurrent connections within its hidden layers (see Figure 2.3). The network can be rewritten by 'unfolding' the network in time. The network processes the elements in the input sequence one-by-one. After processing one element, the internal state h_i gets updated. For the next element, this hidden state will influence the network behavior. The hidden state can be seen as an 'input' from the network to itself at the next step in time (see Equation 2.2)

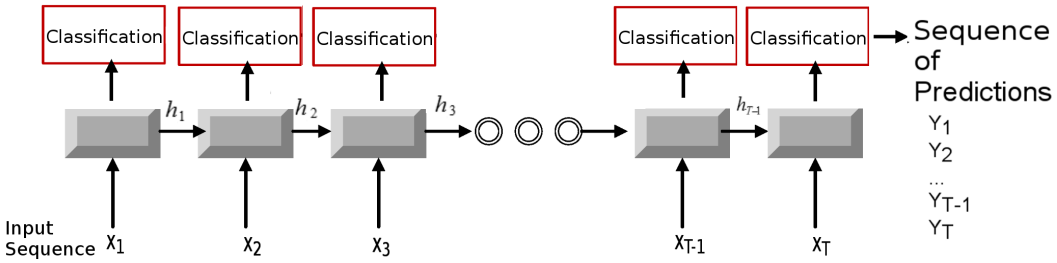


Figure 2.5: Processing of an input sequence by a RNN with one layer, unrolled in time. Each of the blocks represents the network at a different time step (see Figure 2.4) [13].

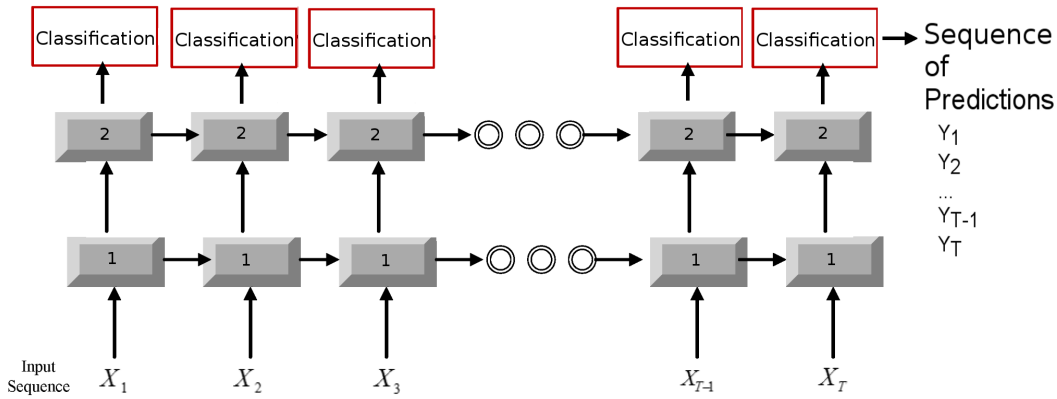


Figure 2.6: Processing of an input sequence by a RNN with 2 layers, shown unrolled in time. The input sequence is the input of the first layer. The output features of the first layer are the inputs of the second layer. [13]

this way, the classification can take both past and future information into account. The classification of an input frame will then incorporate information from both past and future frames.

For truly real-time processing bidirectional RNNs cannot be used as the future information is not yet available at processing time. They require the whole sequence to be known before they can start generating predictions, as the network traverses the sequence both forwards and backwards. For applications that are off-line or where a small delay is acceptable they can increase performance by a significant margin [14].

Stacked ('deep') Bidirectional RNNs can be built by stacking 'bidirectional layers' (see Figure 2.8). As for feedforward networks, deep networks can learn to discover hierarchical structures in the data, and often perform better.

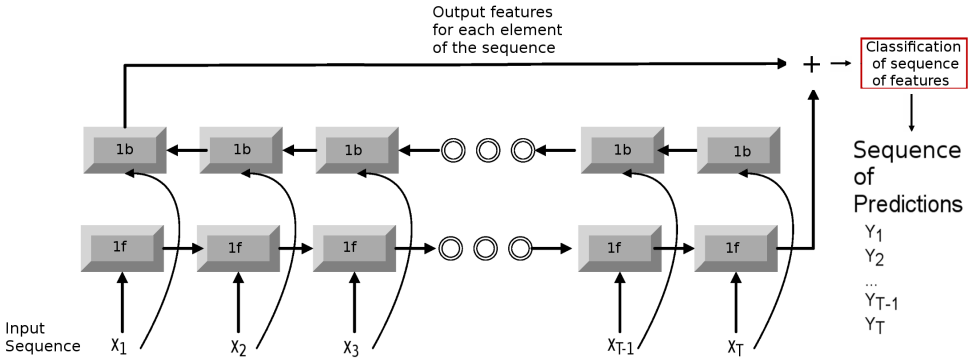


Figure 2.7: Processing of an input sequence by a bidirectional RNN. A bidirectional layer consists of a forward0looking sublayer and a backward-looking sublayer. These sublayers have an equal number of neurons. Each sublayer produces a sequence of output features. For each element of the input sequence, the output features from the forward and backward-looking sublayers are combined to produce a sequence of output features. For the last bidirectional layer of the network, each this sequence of features is classified to produce the final output sequence of predictions [13].

2.4.2 LSTM networks

Standard RNNs have some limitations because of the 'vanishing gradient' problem [15]. This causes them to quickly 'forget' about earlier inputs when the network is presented with new inputs. This is undesirable as many problems require longer-term dependencies to be taken into account.

Standard neurons can be replaced by 'LSTM units' (Long Short Term Memory). These units add a few internal connections ('gates') to the standard neuron (see Figure 2.9). The gates allow the neuron to store information that is not directly related to the current inputs. It can use this to store information long-term, and make long-term connections between inputs. Normal recurrent neurons tend to 'forget' old information as it gets overwritten by new inputs [15][16]. The standard LSTM unit

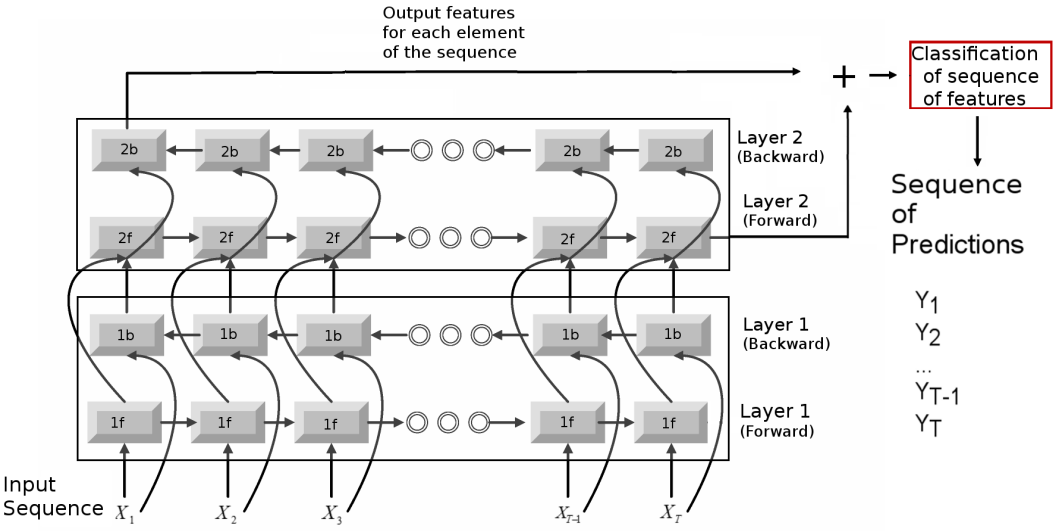


Figure 2.8: Processing of an input sequence by a deep bidirectional RNN. The input sequence is presented to only the first bidirectional layer (both forward and backward sublayers). These sublayers process the inputs as shown in Figure 2.7. For each element of the input sequence, their output features are combined to produce the outputs and then passed to the next bidirectional layer as input. The outputs of the last bidirectional layer are classified to produce the network predictions [13]

uses three gates: a 'forget gate' which controls how much of the internal state remains in 'memory', an 'input gate' controls the influence of the new information, and an 'output gate' controls the influence of the internal state on the output. RNNs with LSTM units are more computationally intensive, but perform better, especially for long sequences. Therefore LSTM units are used almost universally in state-of-the-art RNNs instead of normal neurons [10]. A layer of LSTM units is often denoted shorthand as an 'LSTM layer'.

An LSTM unit (see Figure 2.9) can be described using the equations 2.3 - 2.8 [17]. It has a 'forget gate' F, an input gate I, an output gate O, and internal state C. The output is called H.

Because of more external and internal connections there are more parameters than for the normal neuron. Each of the features of the input data has three connections to each LSTM cell (instead of just one for a standard neuron), with corresponding weights and biases. The input features are also fully connected to the first layer of LSTM units, as for normal RNNs. This means that the number of connection weights increases very quickly when the number of input features is increased, as will be shown in Section 3.3.

$$i_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{2.3}$$

$$f_t = g(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.4}$$

$$o_t = g(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{2.5}$$

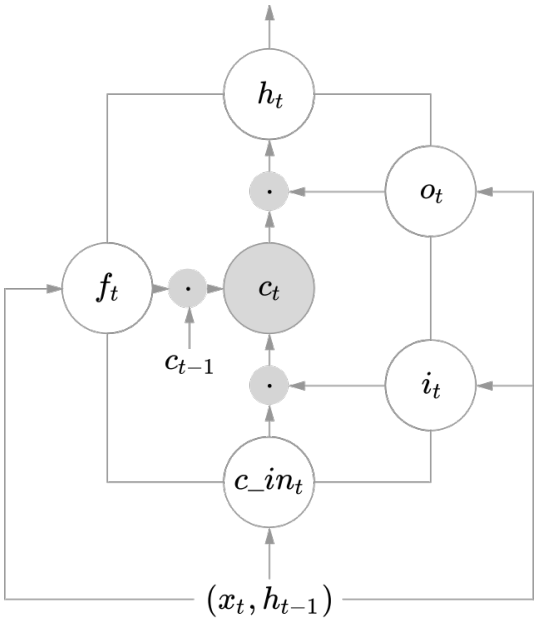


Figure 2.9: The LSTM unit as an alternative for the standard neuron in RNNs. [17]

Input transformation:

$$c\_in_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c\_in}) \tag{2.6}$$

Internal state update:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c\_in_t \tag{2.7}$$

$$h_t = o_t \cdot tanh(c_t) \tag{2.8}$$

2.5 Feature extraction and compression: Convolutional Neural Networks

Convolutional neural networks (CNNs) have revolutionized image processing for many applications, especially for recognition and classification tasks. They are quite similar to normal FCNNs, but are organized in a different way. Neurons connect only to a small part of the neurons in the previous layer, not to all of them. This reduces the amount of network parameters enormously.

A CNN generally follows the structure shown in Equation 2.9. It consists of convolutional (CONV) layers, nonlinearity layers (NonLin), pooling layers (POOL), and fully connected layers (FC).

$$Input -> [CONV -> NonLin -> POOL] * N -> [FC] * M -> FC(softmax) \tag{2.9}$$

A summary of important properties of a CNN:

- CNNs are always multi-layered. One layer processes the input image, and the next layer takes the outputs of that first layer as its inputs.
- Neurons in a layer are not connected to all the neurons of the previous layer, but only to a small region.
- Each layer is a 3D structure. This allows each 2D slice of that layer to specialize in recognizing specific features.
- CONV layers that convolve a number of filters across the height and width of volume of neurons in the previous layer.
- POOL layers to reduce the size of the data by downsampling along width and height.
- FC layers at the end perform classification.

2.5.1 Convolutional layers

The neurons in the **Convolutional Layer (CONV)** layer are arranged in a 3D shape (width, height, depth) (see Figure 2.12). The different slices in the depth dimension of the CONV layer operate independently from each other, and can be seen as filters that each perform a different operation on the input region (see Figure 2.11).

One of the most important advantages of CNNs compared to normal FC networks is that it's possible to reduce the number of network parameters tremendously. The filters of each CONV layer look for a certain feature in a small input region, but the same feature can be useful in another part of the input image. It is therefore possible to use just one set of filters, and slide it over the whole input image instead of creating a set of filters for each input region. This is called 'parameter sharing'. Sliding the set of filters over the whole image corresponds to performing a convolution, hence the name of these networks [8].

Neurons along the width and height dimensions of a CONV layer are connected to a small 2D (width,height) region of the input layer, but extend through the whole depth of that input region. Each filter is therefore connected to an 3D input volume. The 2D region that the filter is connected to is called the 'receptive field' of the neuron.

During network evaluation, dot products between the input volume and each of the filters are computed. This is done for each subregion of the input image. When that dot product is completed, the filters slide again, until the whole image has been covered. Sliding the filters over the input volume will produce the 2-dimensional responses of each filter at every region of the image. The outputs of all the filters are stacked on top of each other, creating a 3D output volume. They are then passed through an activation function, usually the ReLu activation function (see Section 2.2).

During training, each of the filters will automatically learn to recognize a different visual feature (eg horizontal edges, vertical edges, certain colors or other more abstract patterns) [8].

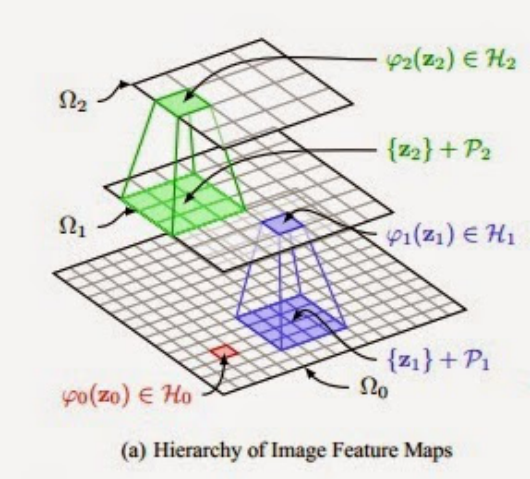


Figure 2.10: Convolutional layer hierarchy [18]

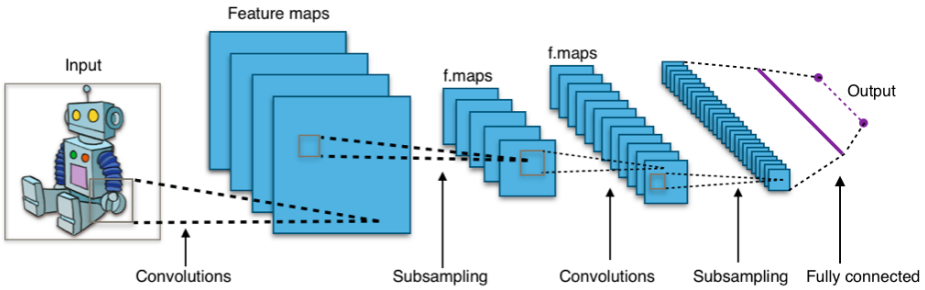


Figure 2.11: CNN structure
Source: Wikipedia - Aphex34

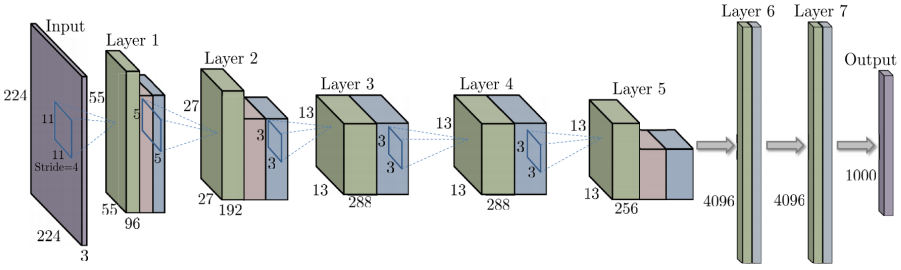


Figure 2.12: Convolutional Network structure from Babenko et al. for ImageNet classification [19]. Purple nodes correspond to the input and output. Green nodes are convolutions, red nodes are max pooling, blue nodes are ReLu transformations.

CONV layers deeper in the network that have other CONV layers as inputs will learn to recognize more high-level patterns. The first CONV layer might detect edges, while the second detects circular shapes, the third recognizes an eye shape, and the fourth a face.

In order not to lose much information, it is recommended to increase the number of filters per layer as the layer 2D dimensions get smaller through convolutions and pooling. In Figure 2.12 the number of filters starts at 96, but goes up to 288 in deeper layers [2].

An excellent explanation of these networks is given in [2]. An example network architecture for ImageNet classification is shown in Figure 2.12.

2.5.2 Pooling layers

The **Pooling Layer (POOL)** layer reduces the input volume in the width and height dimensions. This can be useful for two reasons: it reduces the size of the network and forces the network to generate abstract features out of the input, reducing overfitting. There are different types of POOL layers, but the most common is 'max pooling', where each output corresponds to the maximum of a FxF input region, with F the spatial extent of the filter. Commonly F is set to 2, and the pooling operation is executed with a stride of 2. This reduces the size of the volume by 4 (by factor 2 in width and by factor 2 in the height dimension).

2.5.3 Putting it together

A CNN stacks CNN, POOL and FC layers on top of each other in order for the network to learn progressively more abstract features in the input image. An often-used general CNN architecture is summarized in Equation 2.9. The CONV and POOL layers generate progressively higher-level abstractions of the input, and the FC networks at the end classify these abstract features into the desired categories.

When designing these networks there a lot of hyperparameters to determine:

- the number of layers
- the number of filters per layer (depth of each layer)
- the receptive field for each layer
- the stride and padding for each convolutional layer, which control the output volume shape
- the activation function to use
- whether and what kind of pooling layer to add

This makes it hard to determine what kind of architecture works best for a certain problem. For many of these parameters there's no good way to determine their optimal value, and empiric rules are all we have. One approach to tackle this issue is to simply use networks that other people designed and that have been proven to work well on certain benchmarks like CIFAR10 or ImageNet. If desired, some

customizations can be applied as well: extra layers can be added, the number of filters changed, the activation function can be modified.

Performance of several different CNNs for lipreading are compared in Chapter 4.

2.6 Making Neural Networks more efficient

Neural networks may exhibit very good performance on many classification tasks, but require lots of memory and computation power in comparison to other methods. Apart from inventing and training a better network, there are five main ways to make more efficient usage of available resources:

1. Compressing data when providing it to the network.
2. Reducing network size: depending on the application, the lower performance of a smaller network can still be acceptable.
3. Attention mechanisms: the network focuses on specific 'interesting' regions of interest. It uses its resources where the most information is available.
4. Removing redundant parts of the network by removing weights with very low values and/or by limiting precision of the weights. This is called 'deep compression'. This technique is usually not possible during training as it would hurt performance, but during inference impact on performance can be minimal [20] [21].
5. Custom hardware implementations that combine precision scaling with usage of sparsity of the weights and a data path optimized for convolutional or LSTM networks [22] [23].

2.7 Training

2.7.1 Training algorithm

To train a neural network, lots of labeled data is needed. The network is presented with an input, generates certain output probabilities, and the loss of these outputs is calculated, using the known correct class labels. The error is then fed back into the network to update weights so the network will correct itself and perform better on the next input presented.

The algorithm for these updates is called backpropagation [24]. The algorithm works in two steps: first it calculates the network outputs of for a training example, and the errors of the network outputs compared to real labels. Then the gradients of the loss function (see Section 2.7.3 with respect to each weight in the network are calculated. During training, each weight in the network is updated in proportion to this gradient. Intuitively, this means that weights that contribute a lot to the error will be changed a lot, so the error in the next iteration is improved. The backpropagation algorithm tries to find the minimum in the objective space of the

loss function ('gradient descent'). [25] The Theano [26] framework provides symbolic gradients for each type of activation function of the neurons, making the execution of backpropagation faster and more efficient.

There are many different variations on the standard back propagation algorithms, each having different advantages and disadvantages. Some of the most-used ones are listed below. For an in-depth comparison, see [27] and [28]).

- Stochastic Gradient Descent (SGD)
- SGD with Nesterov momentum
- Adagrad
- RMSProp
- Adadelta
- Adam
- Adamax

2.7.2 Learning rate

The learning rate determines the degree to which the weight parameters are allowed to change during sweep over the training data. Starting with a large learning rate (for example 0.01) is usually recommended as it allows the network to quickly converge to a minimum in the objective space. To obtain optimal performance however, it is required to reduce this learning rate during training so the network can fully descend the objective curve and reach the optimal point.

This is usually implemented by fixing the learning rate and training the network until performance no longer increases, then decreasing the learning rate and continuing training. If performance does not increase even after several decreases of the learning rate, a minimum in the objective space is reached and training is stopped.

2.7.3 Classification and loss functions

The output neurons have a different activation function than the hidden layer neurons. For classification, the number of output neurons will correspond to the number of classes the network has to distinguish between. Each output neuron's output value then represents the probability that the network input belongs to the neurons corresponding class. To achieve this, the softmax classifier can be used.

Assessing performance of the network is done by comparing predicted classes (network outputs) with the actual classes (called *targets*). To evaluate how well the predictions correspond to the targets, the **Cross Entropy Loss (CE)** can used (see Equation 2.11), although other loss functions are possible as well (eg hinge loss). The Cross-Entropy loss function requires its inputs to be probabilities (between 0 and 1).

The network outputs can be any real values, so before applying CE the network outputs need to be squashed to the $[0, 1]$ interval. This is done by the softmax function, which interprets the network outputs z as unnormalized log probabilities

for each class (see Equation 2.10). Using the log probabilities makes the network outputs more distinct for different classes, improving performance.

In summary, using the softmax classifier with the CE loss will minimize the negative log likelihood of the correct class during training. This softmax function is used as activation function in the last layer of the classification network. That layer (the 'softmax layer') has as many neurons as there are possible classes, so the output of each neuron corresponds to the probability that the network inputs belonged to the class that that neuron represents.

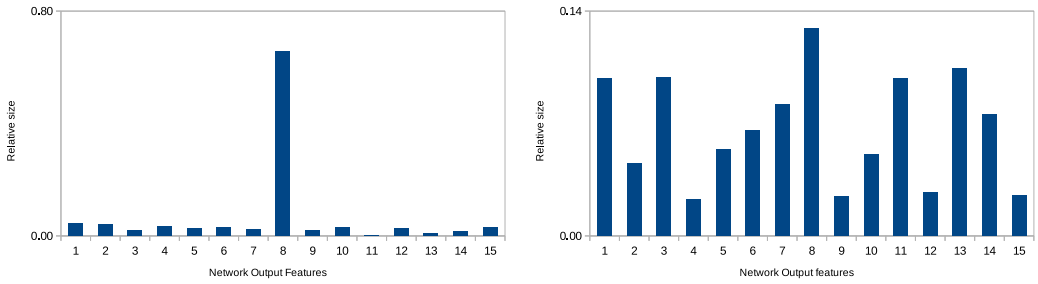
The Softmax function (the probability of prediction p, between 0 and 1):

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.10}$$

Cross Entropy loss of the $i-th$ class:

$$L_i = -f_i + \log \sum_j e^{f_j} \tag{2.11}$$

When the network classifies well, the output features of the softmax layer will have much larger values for the predicted classes than for the others (see Figure 2.13a). If the network is unsure, the output feature values will not be as distinct (see Figure 2.13b).



(a) Good classification

(b) Bad classification

Figure 2.13: Relative sizes of network outputs (predictions) for good and bad classification

2.7.4 Dataset

In order to perform well, neural networks need to train on a good dataset that contains lots of varied data. The choice of a dataset very important. If not enough data is provided, the network will start overfitting and it will perform badly on unseen data. Many datasets can be found on deeplearning.net.

Train/Val/Test split: The data is split in a two main parts: a training set and a test set. The test set will not be used during training, and only after training will it be used to evaluate the network performance on unseen data. An often used

training/test split is 80/20. Some datasets suggest a training/test split to make comparison between results easier.
The training set is then split into a train set and a validation set. The train set is used to actually train the network. The performance on the validation set can be used to determine when to stop training to prevent the network from overfitting. If the train and validation scores diverge too much, the network is overfitting and will lose its predictive capabilities.

Preprocessing: Preprocessing can be necessary for many reasons. The most common ones are summed up here. A good overview of dataset preprocessing is given in [2].

- 'Data augmentation' can be used to prevent overfitting. Data is transformed in some way to create a larger dataset from limited amounts of data. Images can be mirrored, rotated or cropped for example. This is not as good as simply using a larger dataset, but can still be beneficial.
- Most neural networks don't directly take raw data as input, but expect the data to be in a certain shape and format. This depends on the library used and the network architecture.
- Reduce the size of the input data (compression) can make training networks much faster. Using raw data would make networks too large and difficult to train. Throwing useless data away is also a valid strategy. For lipreading from an image of a speaker, the mouth area contains far more information than other parts of the image, and can be cropped out. The other parts of the image are discarded.
- Normalization by mean subtraction and/or scaling can improve performance and is almost always used. For example, image pixel values are usually rescaled from [0, 255] to [-1,1] [2].

For details on how TCDTIMIT is preprocessed for the different tasks (audio recognition, lipreading, multimodal recognition), see Appendix C.

2.7.5 Other issues

- **Deep learning framework:** Different deep learning libraries are available at the moment of writing. Most use the Python programming language, and almost all require a CUDA- capable GPU for GPU acceleration. Some libraries provide a very high-level interface which is easy to use but doesn't provide a lot of control. Others take the opposite approach. An important factor in the choice of a library is the amount of available code that can be reused. The most notable ones are:
  - Theano [26]: A low-level library that provides for a lot of control. Higher-level wrappers are available (Keras and Lasagne [29] being the most notable). This work was developed using Theano in combination with Lasagne.

  - TensorFlow: originally developed by Google and providing both high-level and low-level functions.
  - Torch: a library using Lua language.

- **Batch size:** For practical training, a batch size needs to be chosen. A larger batch size means faster training, but requires a more powerful GPU with more memory.
- **Activation function:** for convolutional neural networks the *ReLu* function is standard; for LSTM networks the *tanh* function is generally used.

2.8 Conclusion

A general overview of artificial neural networks, different architectures and aspects of training these networks was given. Specifics relating to the networks used for audio speech recognition, lipreading and multimodal speech recognition can be found in the next chapters.

Chapter 3

Acoustic Speech Recognition

3.1 Introduction

Speech is one of the most important human inventions that distinguishes us from all other animals. It is a major reason why our species has been so successful as it allows for easy communication of complex ideas. This is essential for a social creature as the human, and helps tremendously when cooperating in group. Just like many other parts of our society have been (and are still being) automated, speech is no exception. Fully automated speech recognition (speech-to-text) and speech generation (text-to-speech) systems can be useful for a wide range of applications, not in the least for automated translation.

Speech recognition has an extensive history, dating back to the mid-20th century (see Figure 3.1). Over time, many innovations and improvements were made, every time improving performance, increasing the recognized vocabulary and increasing robustness. The most important phases in the development of this technology are described below. Afterward, the systems used in this work are described, as well as the datasets. Performance is then analyzed and compared between different network architectures to determine which performs optimally.

3.1.1 Linguistical Models

An analysis of human language was a first step to build recognition systems. The field of phonetics is concerned with analyzing human speech and classifying these sounds based on wavelength, amplitude and harmonics in the spectrum. Human speech can be decomposed in so-called phones or phonemes, which are the basic building blocks of any sound a human vocal apparatus can produce. They can be divided into different classes, each with specific characteristics (Figure 3.2). It is easier to determine the phoneme class than a specific phoneme, and many early systems used these manually created categories.

The first working system was created by Bell Labs in 1952 [32]. AT&T wanted to use speech recognition to provide automated telecommunication services and improve efficiency in its call centers. The team detected resonance peaks in the power spectra

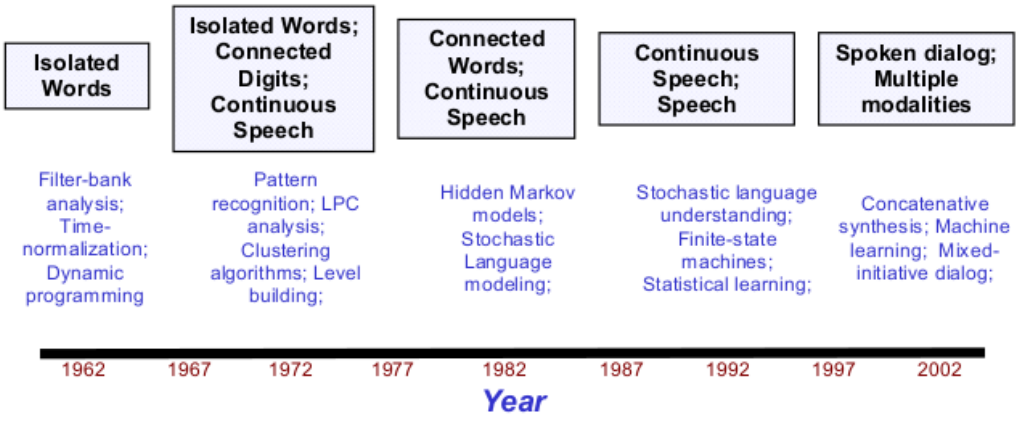


Figure 3.1: Speech Recognition milestones in the 20th century.
In the 21st century, big data and deep learning have sparked a new interest in speech recognition [30].

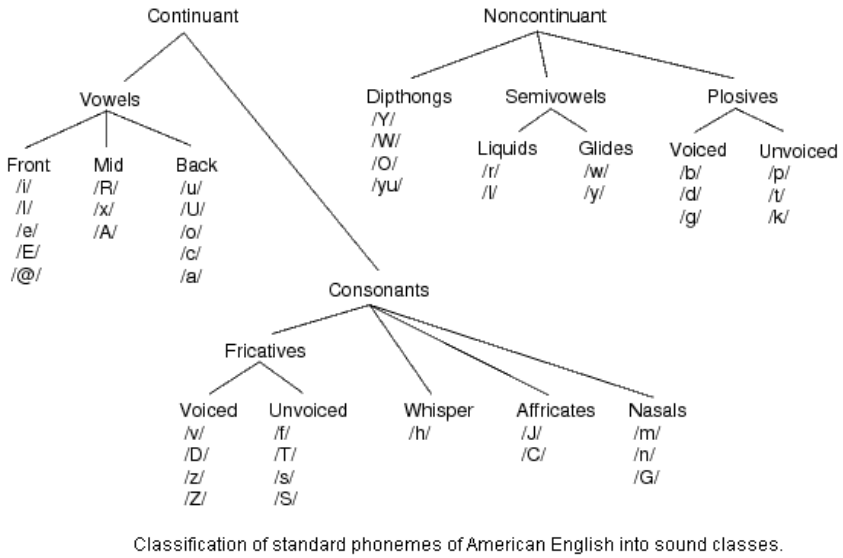


Figure 3.2: Phonemes can be divided into different classes [31]

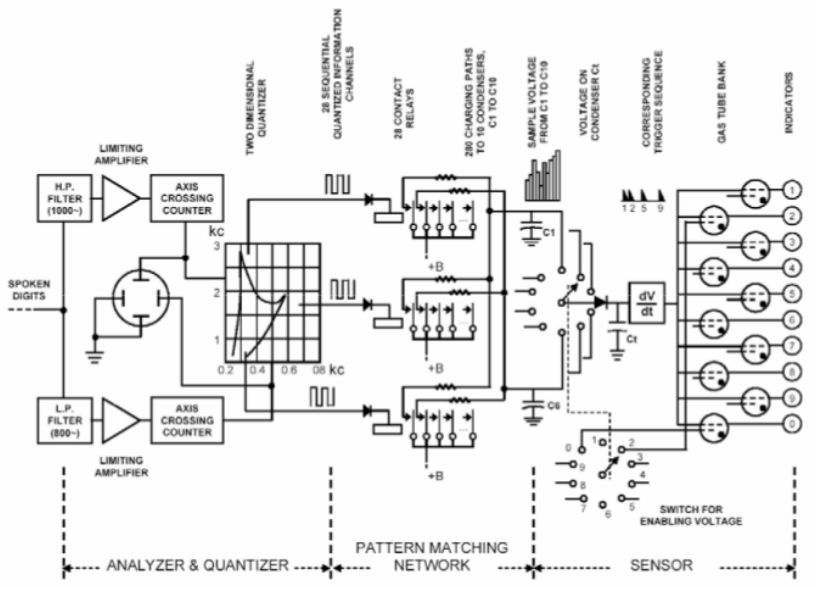


Figure 3.3: The First Speech Recognizer, Bell Labs 1952. [32]

of voice recordings (so-called formants), combined with the knowledge of phonetics to perform recognition.

The second innovation was to separate sound data into segments, and process each segment separately. Research on this was done in several places, most notably in Japan and Russia. Using Dynamic Time Warping and other time alignment techniques, it was easier to find the distinct location of phonemes in the speech spectrum, and different sounds could more easily be compared and thus recognized.

The US Department of Defense Advanced Research Projects Agency (DARPA) helped funding research in SR in the '70s which boosted interest in the field and produced the first usable systems [30].

3.1.2 Statistical Models

After Leonard Baum's paper on Hidden Markov Models in 1967[33], interest in statistical methods grew slowly. The n-gram model, developed by IBM, was one of the first such systems. It is a language model that takes into account the probability of occurrence of certain words in a sequence. In essence, it is a statistical model of the grammar of a language. Another technique used was 'keyword spotting'. Not all words in a sentence have the same importance, and especially in known situations like callcenter support desks, specific words occur often and contain most of the information.

The success of these methods showed that mathematical models were superior to previous manually created models that only used knowledge of the phonetics. During the '80s and '90s, Hidden Markov Models were further developed.

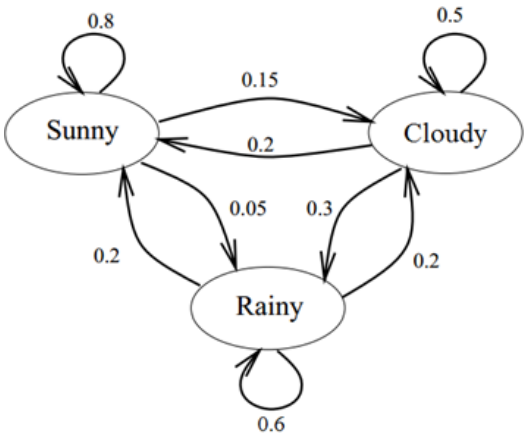


Figure 3.4: Hidden Markov Models

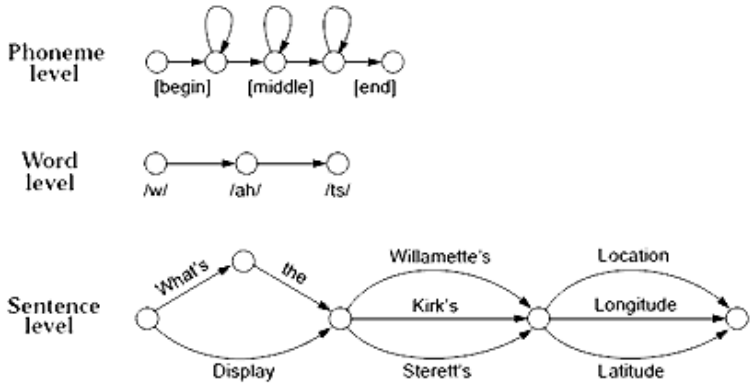


Figure 3.5: HMMs on multiple levels of the language

The Hidden Markov Model models both the speech features and the language model. It uses a Markov Chain to account for variability in terms of speaker and tonality. An example is shown in Figure 3.4. It uses labeled data to learn the dependencies between speech features and between words, and increases its own performance during a training phase. After training, it can be used on new, previously unseen data [34]. Hidden Markov Models have long been a standard method in Speech Recognition, and are still widely used in many systems. The most widely used toolkit is the Hidden Markov Model Toolkit (HTK) [35].

3.1.3 Neural Networks

Neural networks have already extensively been discussed in Chapter 2. For audio processing, time dependency between the input data is very important. As such, RNNs and LSTMs are mostly used and are now omnipresent for speech recognition. It is also possible to use CNNs for speech recognition [36] [37]. There has also

been interest in combining the different network types into one architecture [38] [39]. Some research has been done into feeding raw audio data into the neural network [37], though many systems preprocess the raw audio data by converting it to Mel-Frequency Cepstral Coefficients (see Section C.1.1. Another method to further improve performance is by using attention mechanisms [40]. Since neural networks generalize and perform better when they are trained on more data, while big data has made large amounts of data available via the internet, industry is also increasing performance by scaling up the systems [41].

3.1.4 Language models

In addition to detecting specific phonemes in an audio fragment, these phonemes can only be combined in specific ways to form valid words of a language. Use of a dictionary or a HMM or neural network trained on the dictionary can make use of this external knowledge of the language and even grammar to achieve high word-level and sentence-level accuracies. Much of current speech recognition research focuses not on phoneme-level but on word-level or sentence-level classification. Many combine the phoneme-level system with a language model [42].

3.1.5 Determining valid predictions

For practical systems, there are no phonetic labels provided for the test data files, and the network will have to decide when and whether a phoneme was pronounced. This work does not implement that step and uses the timestamps of the provided label files. The focus is on the classification task, not the time alignment and phoneme selection.

When it is not known at what exact time frames the network needs to generate a prediction, the network itself has to determine that. It can simply generate a prediction for each input frame, but as phonemes overlap a large part of the time, many predictions will be based on audio frames 'in between' two phonemes. The predictions will often be wrong and there is still some selection mechanism needed to choose which ones to compare with the target labels in order to properly evaluate network performance. See Figure 3.6.

In research there has emerged a dominant solution to this problem, called 'Connectionist Temporal Classification (CTC)' [43][44][45]. As the name implies, it is a technique to perform classification while making use of temporal information, resulting in a certain sequence of valid predictions.

The goal is to optimize the probability of a sequence of predictions (network outputs) for a sequence of input frames, accounting for all possible alignments of predictions with those input frames. On objective function takes this into account and automatically removes superfluous predictions (for example when the same phoneme is predicted for 10 successive frames, this results in only one prediction output). It also allows the network to simply produce no output (blank class) at a timeframe. Because it selects the time frames at which predictions are considered

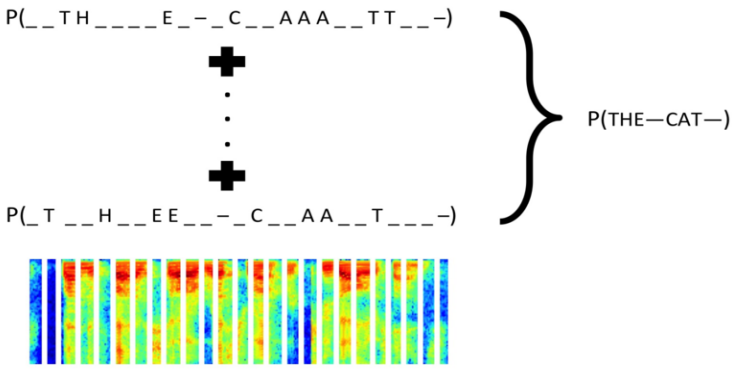


Figure 3.6: Selecting valid predictions using CTC [41]

valid, this can be seen as an alignment between the network predictions and the phonemes that were actually spoken.

The performance of systems that make use of CTC is lower than for systems without it as there's an extra level of uncertainty. It's possible that CTC doesn't eliminate all superfluous predictions. These extra predictions that don't correspond to a valid time frame are called 'insertions'. To compensate for this, the metric of 'correctness' was introduced in addition to the 'accuracy' metric. Correctness takes into account the inserted phonemes. Accuracy ignores these 'inserted' phonemes [46].

$$correctness = \frac{C}{N}\% \qquad (3.1) \qquad\qquad accuracy = \frac{C-I}{N}\% \qquad (3.2)$$

C = The number of correct predictions at valid time frames
N = The total number of valid predictions
I = The number of inserted predictions at invalid time frames

3.1.6 Comparison with previous work

The dataset TIMIT was chosen to compare performance of audio SR networks. TIMIT has been a standard dataset for speech recognition for a long time, and some performance comparisons are available. Usage of different datasets can result in large performance differences, due to misalignment or errors in the labels, low recording quality, the distribution of speakers, the dataset size or the phonetic richness and variety of the dataset.

As industry has gained a large interest in this field in recent years and big data has made it possible to use much larger datasets, many papers use these large datasets and no longer report performance scores on TIMIT [41][47]. Sadly, most of these new datasets are also not made publicly available. The availability of TIMIT was another reason to choose that dataset.

Much industry research uses word-level prediction, as word-level labels are much easier and cheaper to obtain than phoneme-level labels. Even when phoneme-level classification is targeted, there is often a language model on top of the phoneme classification. Phoneme-level accuracies are often not reported as word-level perfor-

Dataset	Correctness (%)
TIMIT (CTC + DBLSTM)	82.3
TCDTIMIT (HMM)	65.47

Table 3.1: Performance comparison: previous work
The best performing TIMIT network used Deep Bidirectional recurrent networks with LSTM units (DBLSTM networks), with CTC for alignment [10]. The only known score on TCDTIMIT is achieved with HMMs [1].

mances are more directly useful for a practical system. There are also several different phoneme sets that can be used for classification and also impact performance scores [48].

The factors described above make direct performance comparisons between different papers quite difficult, but not impossible. Some performance figures are given in Table 3.1. The correctness scores are compared with the performance of networks from this work in Section 3.6.

State-of-the-art frame-level phoneme recognition performance on TIMIT with CTC for segmentation is 82.3% correctness using deep bidirectional LSTM networks, and language model (an 'RNN transducer') on top of the phoneme prediction network [10].

There is only one known result on TCDTIMIT, using Hidden Markov Models [1]: 65.47% correctness and 47.63% accuracy. The authors attribute the relatively low accuracy to errors and misalignment in the dataset labels as TCDTIMIT is semi-automatically labeled in contrast to the manually-labeled TIMIT dataset. Another factor is that the number of speakers in TCDTIMIT is much smaller and have less variety in accents, which makes it hard for the trained networks to generalize.

3.2 Network Architectures and Training method

State-of-the-art performance is currently reached by recurrent networks, specifically multilayer bidirectional LSTM networks [10]. LSTM networks are RNNs with LSTM units instead of standard neurons. The LSTM 'neurons' are called 'units' to avoid confusion as they are more complex and not really comparable to a simple neuron.

The performance impact of different network architectures is discussed in Section 3.3. These networks differ in number of layers and number of units per layer.

The performance impact of variations to hyperparameters other than the network architecture are discussed in Section 3.4.

The tested networks are first trained and evaluated on TIMIT, since there are more comparable results than for TCDTIMIT. The best networks are then trained and evaluated on the TCDTIMIT dataset.

3.2.1 Network Architectures

For a discussion of the possible architectures of an LSTM network, see Section 2.4. The general network structure of a deep (bidirectional) LSTM network can be summarized as follows:

- K layers, with N_i LSTM units in each layer
- The LSTM units on the first layer are fully connected to the features of one frame of the input sequence.
- For bidirectional networks, each layer contains two 'sublayers' of LSTM units, traversing the input sequence in opposite directions.
- The output features of the last LSTM layer are fed through a softmax (FCNN) layer for classification.

A visualization of the network architecture is shown in Figure 3.7. The workings of an LSTM are hard to visualize because of the time dimension. It processes a sequence of inputs one by one, but this is more complex because it's a bidirectional network. Then some parts of the network process the sequence in reverse order. This means the network is influenced by both the current input frame, the previous input frames, and the future input frames simultaneously.

To calculate predictions for an audio fragment the preprocessed audio frames (MFCCs) are used. The frames are shown one by one to the network , in chronological order to the forward LSTM layers, and for bidirectional networks they are shown in reverse chronological order to the reverse LSTM layers. An illustration is shown for unidirectional RNNs is shown in Figure 2.5 and for bidirectional RNNs in Figure 2.7.

Multilayer LSTM networks take the MFCCs as inputs only for the first layer. Subsequent layers process the output features of the previous layers. This is illustrated in Figure 2.6.

As the number of LSTM units increases, the number of connection weights between the input features and the first LSTM layer will also increase a lot (see Section 2.4.2). This will have a large impact on the number of parameters of the network (see 3.3).

3.2.2 Training

For the data set used and required preprocessing, see Appendix C.

When evaluating the network, the LSTM layers produce an output for each time frame (generally 10ms for MFCCs, Section C.1.1). To filter these outputs and only retain distinct phonemes, techniques like CTC can be used (see Section 3.1.6). In this work, the provided labels are used to calculate the timestamp of each phoneme in the labelfiles, and only network outputs at the timeframes corresponding to these timestamps are considered. This is achieved by using a mask which contains 0's for invalid timeframes and ones for valid timeframes. This mask is used as input to the LSTM layers, and is also used in the Theano function for calculation of the loss.

As the videos are not of equal length, the number of timeframes varies between videos as well. The number of frames is required to be equal for all sequences in a

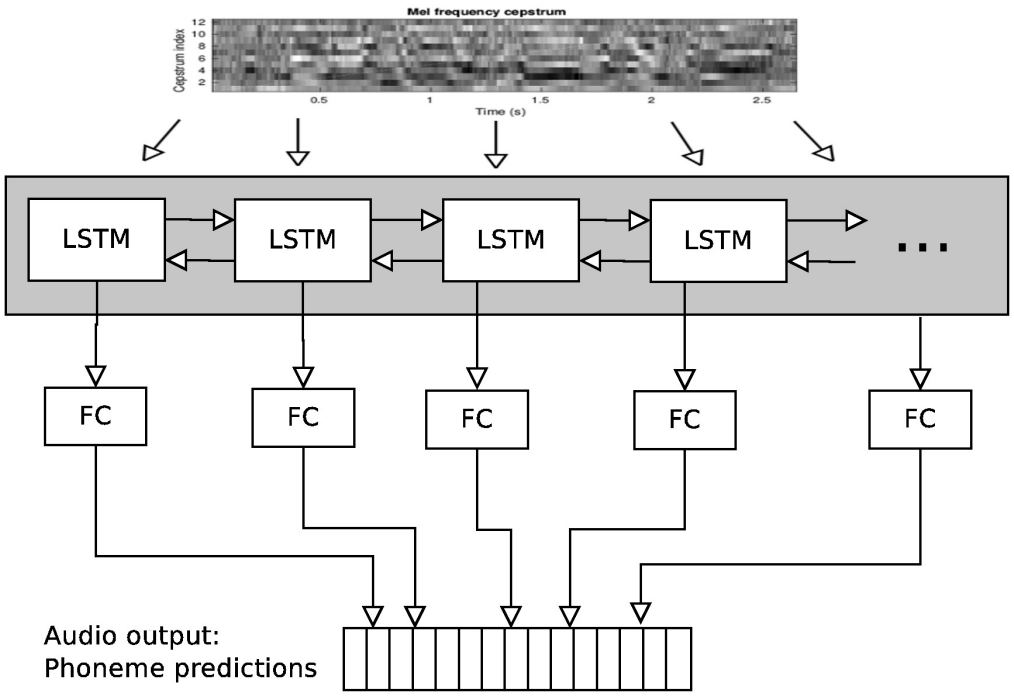


Figure 3.7: Architecture of audio network. The 'LSTM' blocks represent the LSTM network at different timesteps. The LSTM network processes the input sequence one frame at a time, but in two directions simultaneously (as it's bidirectional, as the arrows show). The forward LSTM layers of the network process the sequence front to back and the reverse LSTM layers process the sequence back to front (see Section 2.4). The FC classification layer processes the output at each timestep to generate phoneme predictions for each frame of the sequence.

batch. One solution is to simply use a batch size of 1. This makes training very slow, so is not an option for large datasets. The solution is to pad the shorter videos with zeros until all videos in a batch have the same number of timeframes. A mask is used to indicate which frames are valid (0) or invalid (1). This mask can be multiplied with the mask for determining the valid frames, so the final mask determines which frames are both within the video length and have valid timestamp.

To correctly calculate the categorical cross-entropy for the loss function, the Lasagne library requires the prediction and true class label arrays to be 1D. As the label arrays were first arranged in a 2D shape (Section C.1.1), the label array needs to be flattened to 1D. This corresponds to concatenating the labels from each video in the batch to each other. The same is done for the valid predictions.

Fully Connected layers in Lasagne require a 2D-shaped input, in contrast to the 3D shape that LSTM layers work with. To connect the two, the LSTM outputs are reshaped using the Lasagne *ReshapeLayer*.

All of the networks are trained on the TIMIT training set and evaluated with the TIMIT test set, using the recommended train/test split [49]. This is done to be

better able to compare results with results from literature. It is assumed that the same networks will perform similarly when trained and evaluated on another dataset, like TCDTIMIT.

The learning rate for training is set to 0.01 at the beginning, and decreased when performance does not improve in an epoch. When performance didn't improve for several epochs, training is stopped. Glorot weight initialization is used (the default in Lasagne) [27].

As a measure of performance, top-1 accuracy is used (see Section 2.3). If it doesn't obscure the graphs, the top-3 accuracy is given as well.

3.3 Performance impact of Network Architecture

Larger and more complex networks usually deliver better performance, but require a lot more memory and computational power. It is important to achieve good recognition performance, but not at any cost. For practical applications, especially in embedded systems, the resource requirements are very important. Therefore it is useful to determine what the trade-off is between performance and computational complexity.

A reasonable measure of computational complexity is the number of weights in the network. During evaluation, each weight will need to be multiplied with some value, the result summed up and passed along. This is the most expensive operation in the neural network and a therefore the number of weights can be used as a reasonable approximation of the true computational cost. It is also a good indicator of the memory required to store the network.

Several different network architectures have been evaluated, as shown in Table 3.2 and Figure 3.9. For this comparison, hyperparameters are set as follows:

- Input features: 13 MFCC coefficients, their first and second derivatives, for a total of 39 input features.
- MFCC frame length of 10 ms, with overlapping window size of 25ms.
- Bidirectional LSTM layers
- 39 phoneme output classes

3.3.1 Single layer, different number of LSTM units

Figure 3.8 shows that as the number of units per layer is increased, performance generally goes up as well. There are diminishing returns, and when the number of network weights is added in, the 'sweet spot' for a best compromise between performance and network size seems to be around 256 units per layer. This is consistent with results in state-of-the-art networks [10].

3.3.2 Multiple layers, different number of LSTM units

The same trend from Figure 3.8 can be seen here: more LSTM units increase performance. It is also clear that going from single-layer to multi-layer networks

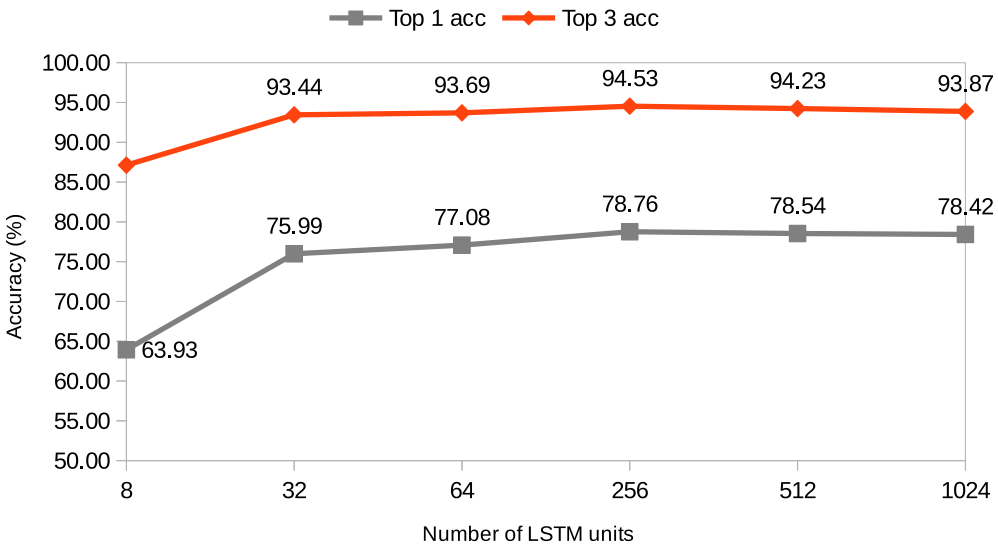


Figure 3.8: Single layer bidirectional LSTM networks

# Units/ Layer	1 layer	2 layers	3 layers	4 layers
8	63.93	64.22	65.02	64.67
32	75.99	76.70	76.74	77.67
64	77.08	79.48	79.90	78.66
256	78.76	80.98	80.93	79.42
512	78.54	81.61	82.04	79.34
1024	78.42	79.56	81.42	81.64

Table 3.2: Multilayer bidirectional LSTM networks

improves performance by quite a large margin (about 4%). Going to more than 2 layers does not seem to impact performance very much though.

Figure 3.10 and Table 3.3 show the trade-off between accuracy and network size. When the number of units per layer is increased, network size goes up exponentially because of the internal connections within an LSTM layer (see Figure 2.3).

The number of weights in a network consists of 3 parts:

1. Input connections: the weights to connect to the input features (this is fixed for a given number of units in the first layer).
2. Classification weights: the weights in the FC classification network, which is also fixed for a given number of units in the last LSTM layer.
3. Internal connections: the weights connecting LSTM layers internally. LSTM layers contain lots of internal connections. The number of these weights increases exponentially with the layer size as shown in Table 3.3.

The network with one layer only is connected to the input and contains a

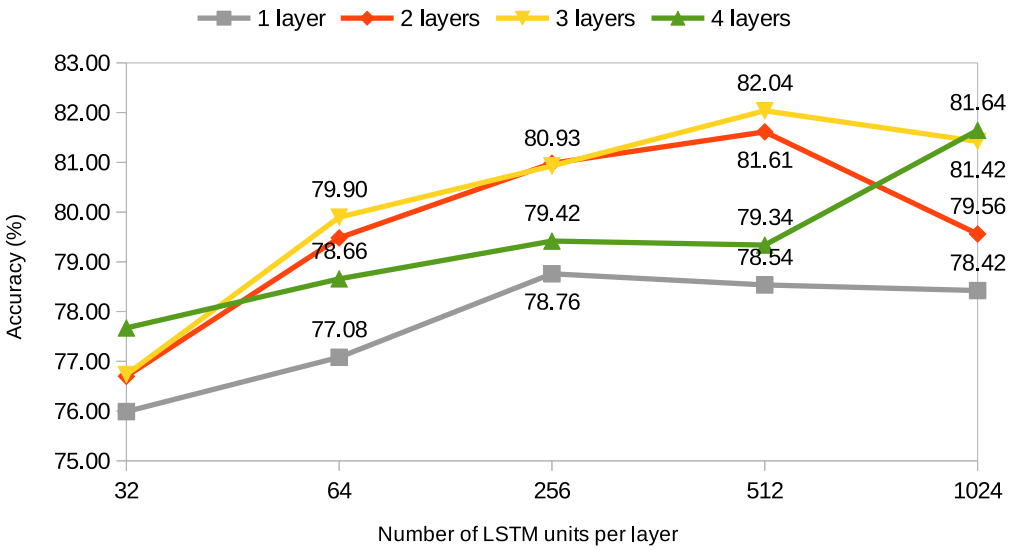


Figure 3.9: Performance for all trained networks
(network with 8 units per layer not shown)

classification network. It has a different number of weights than subsequent layers. This is because subsequent layers only connect to the previous LSTM layer, not to the input data. Adding additional layers increases the number of weights in the network with a fixed amount, given the number of units per layer. This information is used to determine the number of FC classification weights for every type of network. All of this is shown in Table 3.3.

It is generally better to add more layers than to increase the number of units. Adding more layers has a modest impact on the computational complexity, while using larger layers has a much larger impact on the cost for a smaller relative gain.

The network with just 8 units per layer has a very small network size of just a few thousand weights, but still achieves more than 60 % accuracy even with just one layer. On the other hand, going from 256 to 512 units per layer quadruples the number of weights for only a small increase in accuracy. It will depend on the application which network is the best choice.

As a baseline network for comparing variations in other hyperparameters, the two-layer network with 64 LSTM units per layer was chosen as it trained quickly and still delivered good performance. The performance results below use that network unless specified otherwise.

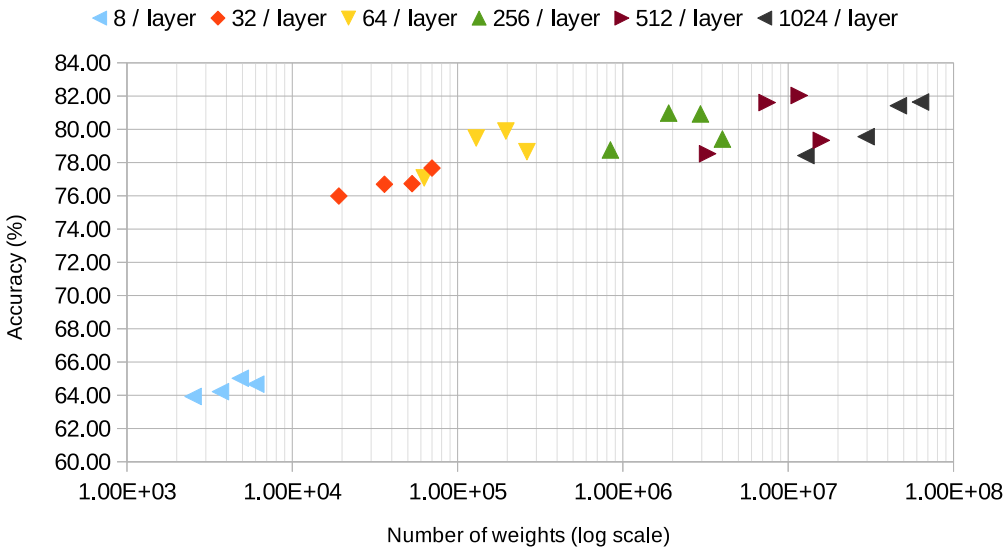


Figure 3.10: Accuracy - Number of weights trade-off. The four dots for each layer size are (from left to right) the networks with 1, 2, 3 and 4 layers. The number of units per layer has a larger impact than the number of layers. The performance increases the most by adding a second layer. Adding a third layer does not increase performance much and a fourth can even hurt it.

3.4 Impact of other factors

3.4.1 Bidirectional Networks

Bidirectional LSTM layers require twice as many units compared to unidirectional layers, but their performance is also significantly higher (see Figure 3.11). For truly real-time applications they cannot be used as they make use of future information, but where a slight delay is acceptable they can improve performance.

3.4.2 Input data features

Proccessing of the raw audio files and generation of MFCCs and other preprocesssing steps are explained in the appendices, Section C.1.1.

Number of MFCCs

The standard in literature is to use 13 MFCC coefficients plus their first and second derivatives, bringing the number of input features per audioframe to 39. Figure 3.12 shows how performance is impacted when the both derivatives are left out (13 coefficients), the second derivative is left out (26 coefficients), and when 40 instead of 13 MFCC coefficients are used with first and second derivatives.

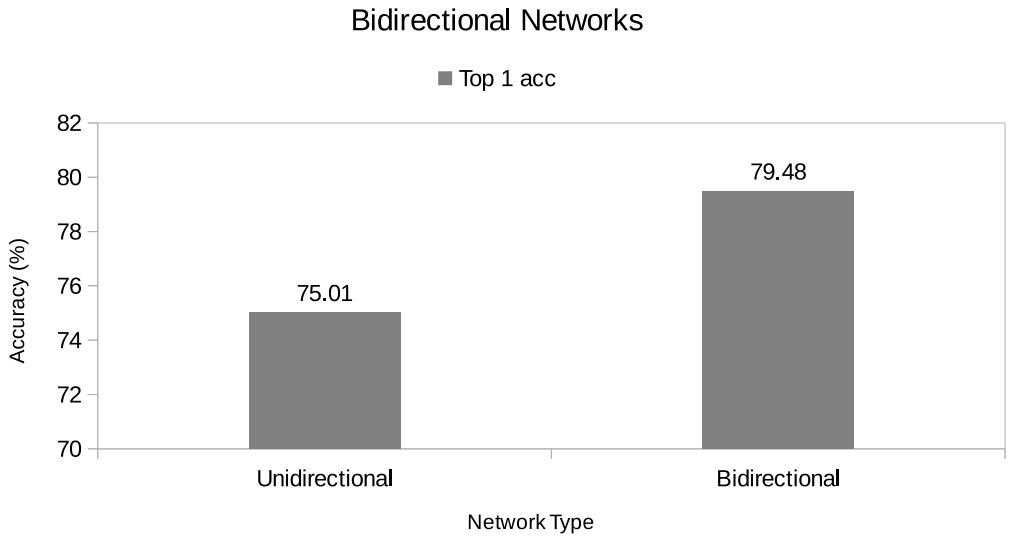


Figure 3.11: Bidirectional networks (2 layers, 64 LSTM units/layer)

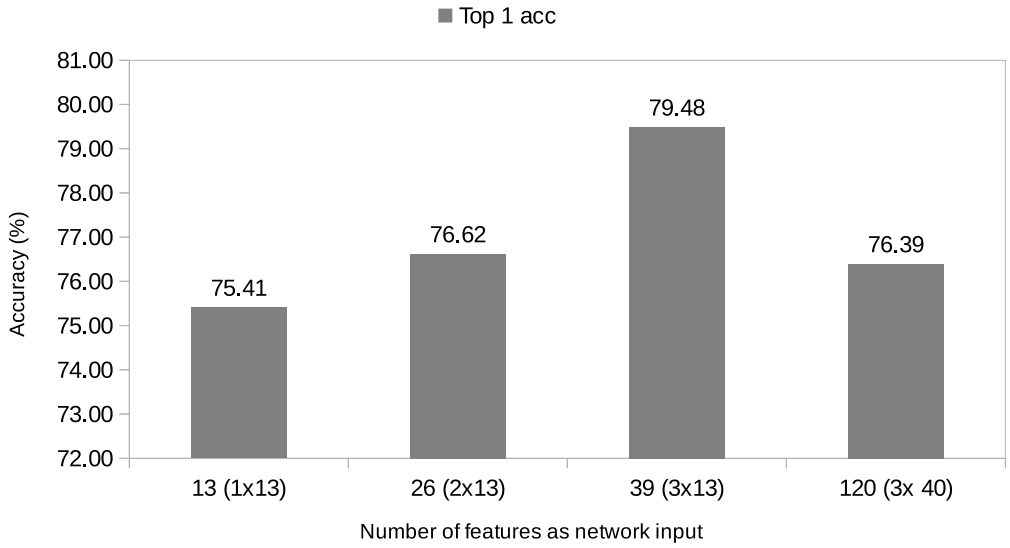


Figure 3.12: Varying the number of MFCC coefficients (2 layers, 64 LSTM units/layer)

Network Type	Total # weights	Per additional LSTM layer	Classification weights	Accuracy
8 / 1	2,511	2,160	351	63.93
8 / 2	3,679	1,168	351	64.22
8 / 3	4,847	1,168	351	65.02
8 / 4	6,015	1,168	351	64.67
32 / 1	19,143	17,856	1,287	75.99
32 / 2	36,103	16,960	1,287	76.70
32 / 4	53,063	16,960	1,287	76.74
TIMIT	70,023	16,960	1,287	77.67
64 / 1	62,823	60,288	2,535	77.08
64 / 2	129,511	66,688	2,535	79.48
64 / 3	196,199	66,688	2,535	79.90
64 / 4	262,887	66,688	2,535	78.66
256 / 1	840,999	830,976	10,023	78.76
256 / 2	1,894,183	1,053,184	10,023	80.98
256 / 3	2,947,367	1,053,184	10,023	80.93
256 / 4	2,947,367	1,053,184	10,023	79.42
512 / 1	3,254,823	3,234,816	20,007	78.54
512 / 2	7,458,343	4,203,520	20,007	81.61
512 / 3	11,661,863	4,203,520	20,007	82.04
512 / 4	15,865,383	4,203,520	20,007	79.34
1024 / 1	16,835,623	12,761,088	39,975	78.42
1024 / 2	29,596,711	16,795,648	39,975	79.56
1024 / 3	46,392,359	16,795,648	39,975	81.42
1024 / 4	63,188,007	16,795,648	39,975	81.64

Table 3.3: TIMIT phoneme classification accuracy - Number of weights tradeoff. More weights mean more MAC operations and more required memory to store the network.

MFCC window length

Results here are shown for the 2-layer network with 64 LSTM units per layer, trained on both TIMIT and TCDTIMIT and evaluated on the TCDTIMIT test set. They are not directly comparable with the other results given, but do give an indication of the relative performance differences for other test or train sets. The standard window overlap for the MFCC calculation is 25ms, with a frame length of 10ms.

As LSTM networks incorporate time information through their architecture, it was speculated that smaller overlapping windows would result in more distinct frames, while the network could learn the time dependencies between the frames by itself. As shown in Figure 3.13, this proved to be false in practice.

3.4.3 Reduced weight precision

An experiment where network weights were converted to 16-bit floating point numbers (from 32-bit floats) showed that network performance was not affected at all (see Table 3.4). This is a simple way to save computational resources during inference

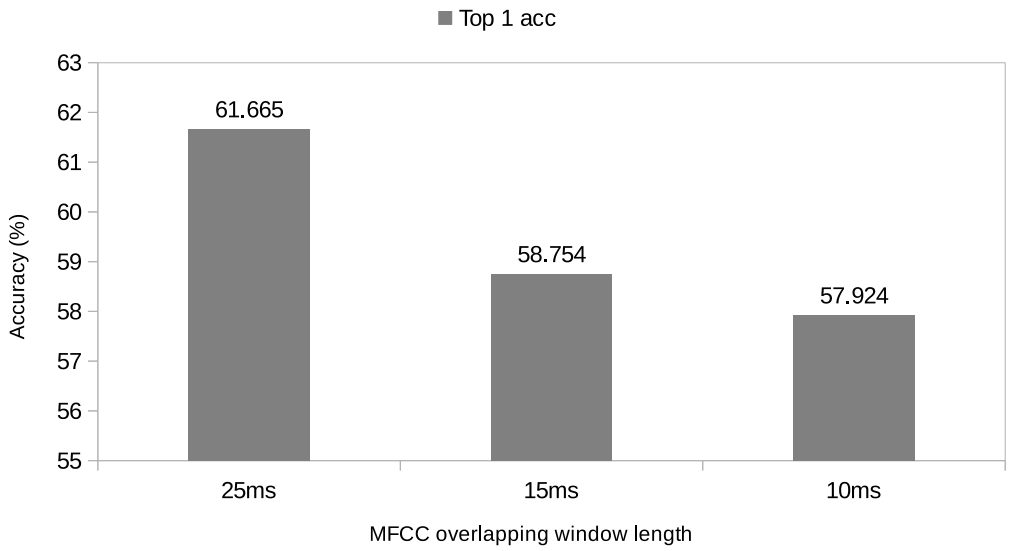


Figure 3.13: Varying MFCC overlapping window length (2 layers, 64 LSTM units/layer). These scores are on TCDTIMIT instead of TIMIT, with a small network and incomplete training. The relative performance differences are clear though.

Network type	float32 top 1	float32 CE cost	float16 top 1	float16 CE cost
Audio (256 / 2)	67.0630	0.08044	67.0578	0.08044

Table 3.4: TIMIT: phoneme classification performance with 16 bit weights

of these LSTM networks. Another technique would be to use binarized networks as ie [50], similar to the convolutional binary networks from [51], but that was not explored here.

3.5 Performance impact of Noise

The networks were trained and evaluated on clean, noise-free audio. In a real-world situation the audio will not be noise-free. In order to test the performance of the trained networks on noisy data, two situations were examined. The first uses audio with added white noise at different SNR levels (0dB, 3dB, 5dB, 10dB). In the second situation, the noise comes from other speakers. The average power of the other speakers is added at 0dB, -3dB, -5dB and -10dB relative to average power of the real speaker. See Section C.1.1 for further details.

The results can be seen in Figure 3.14 and Figure 3.15. For each noise level, the results are the scores of the best performing network out of all the networks in Figure 3.9.

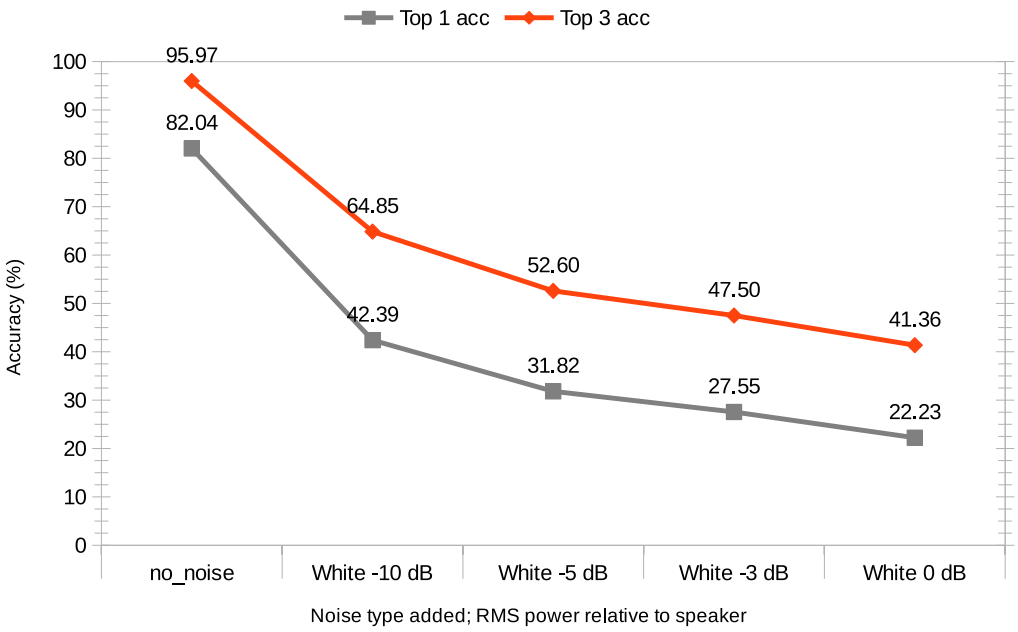


Figure 3.14: TIMIT: Performance with white noise

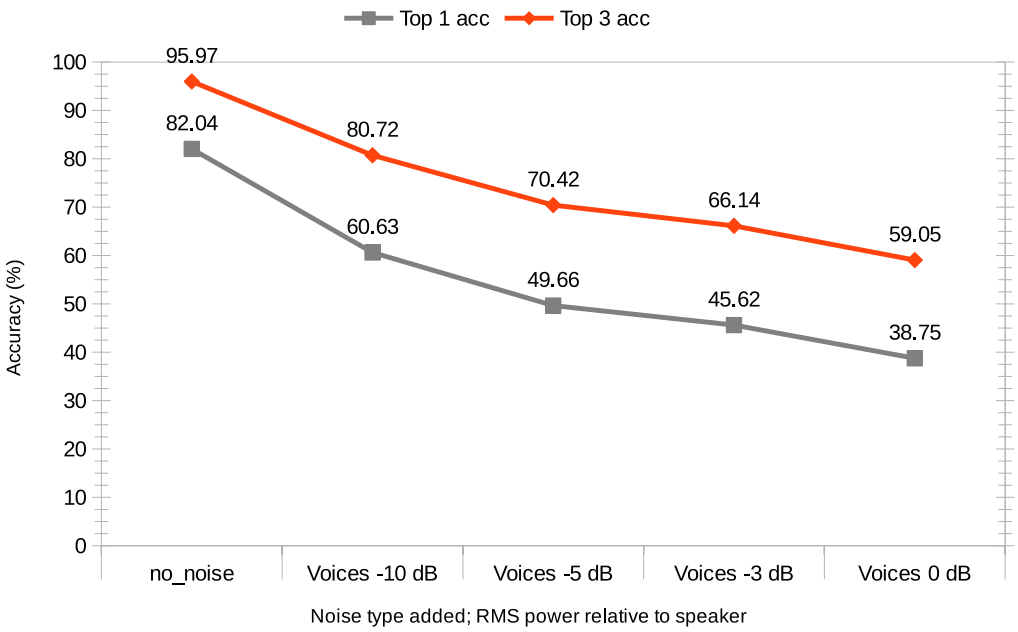


Figure 3.15: TIMIT: Performance with noise from other speakers

The results show that performance decreases significantly when the audio is not clean. The loss in performance is lower when other speakers are speaking, possibly due to the fact that the power spectrum for human speech is quite uneven, and as the exact timestamps of the speakers‘ phonemes are known, the network always evaluates good frames. If these labels are not known and for example CTC is used, the system could get confused as to who was speaking, and insert many undesired phonemes from the second speaker.

3.6 Performance on TCDTIMIT

Performance on TCDTIMIT is generally significantly lower than on TIMIT (Figure 3.16), as also noted by the authors of the TCDTIMIT database [1]. Overall performance on the TCDTIMIT test set (containing only non-lipspeakers) was 67.03% correctness. This is comparable to the baseline reported in the authors' paper (65.47% correctness).

One of the reasons cited for the lower performance was the lack of variety in the TCDTIMIT speakers, both in number of speakers and in dialects. Another factor is that the TCDTIMIT labels are not manually created, but automatically. This causes alignment errors. Listening to TCDTIMIT audio and comparing it with the TIMIT audio, the quality seemed similar or even better for TCDTIMIT, so the issue is likely due to inconsistencies the labels.

It was attempted to increase performance on TCDTIMIT by using both the TIMIT and the TCDTIMIT train sets for training. This provides about double the number of training examples than only using TIMIT or TCDTIMIT. Performance results are shown in Figure 3.19. Accuracy decreased instead of improving. This may be due to differences in labeling (TCDTIMIT used a force-alignment technique to automatically label the phonemes, while TIMIT was manually labeled) [1].

In order to bring performance on TCDTIMIT to similar levels as achieved on TIMIT, the labeling issue should be resolved. If the issue is due to misalignments, using CTC for automatic selection of valid predictions could help a lot because it does not make use of the timestamps in the labels. Only which phonemes are spoken and their order is relevant.

The network trained on TCDTIMIT experiences strong performance degradation when noise is introduced in the audio, similar to the TIMIT-trained networks (Figure 3.18 and Figure 3.17). The baseline performance curve for audio with white noise is shown in Figure 3.20. Only the audio-only curve for the correctness is relevant for comparison.

The TCDTIMIT authors also trained only on the lipspeakers and evaluated the performance on test data from the lipspeakers. They reported about 6% higher correctness for clean audio. This was not done here as it is not relevant for the TCDTIMIT test set (which contains only non-lipspeakers). Performance of the volunteer-trained network was tested on the lipspeaker test set however, and reached around the same correctness as for the volunteer test set (67%).

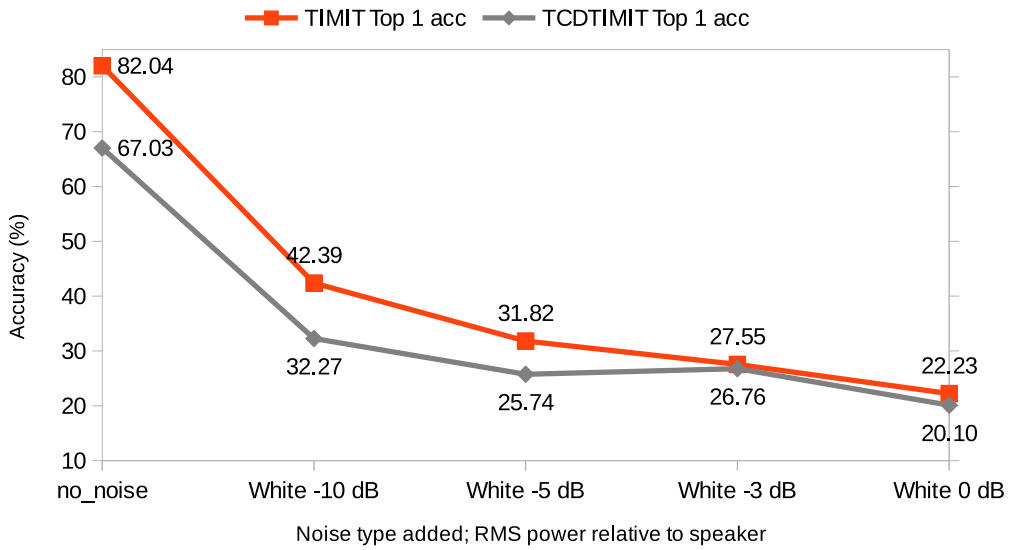


Figure 3.16: Performance difference between TIMIT and TCDTIMIT (trained and evaluated on their respective train and test sets)

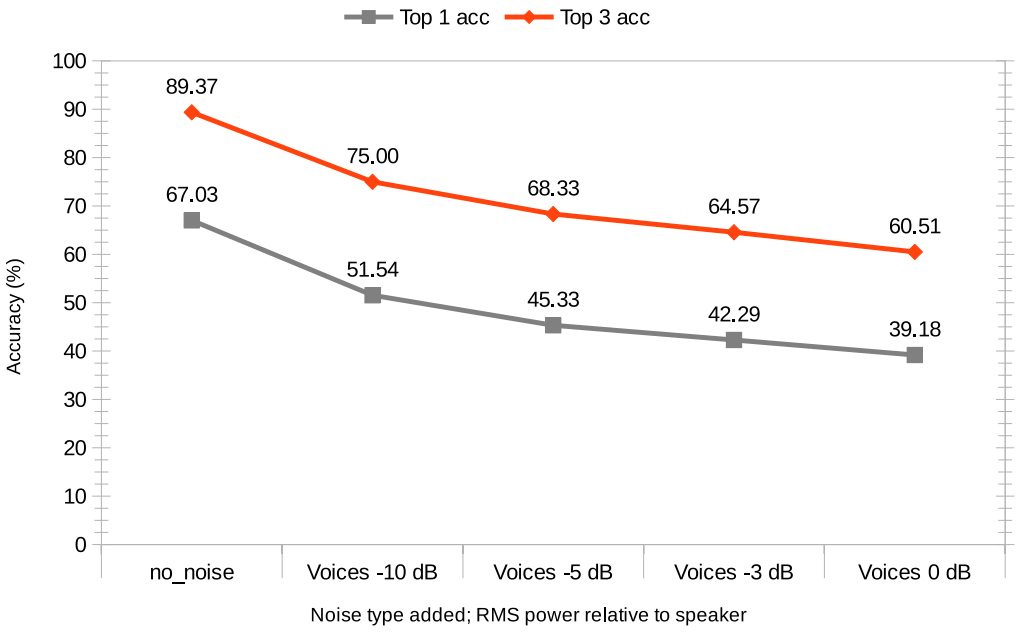


Figure 3.17: TCDTIMIT test set: Performance with noise from other speakers (2 layer, 256 units/ layer network)

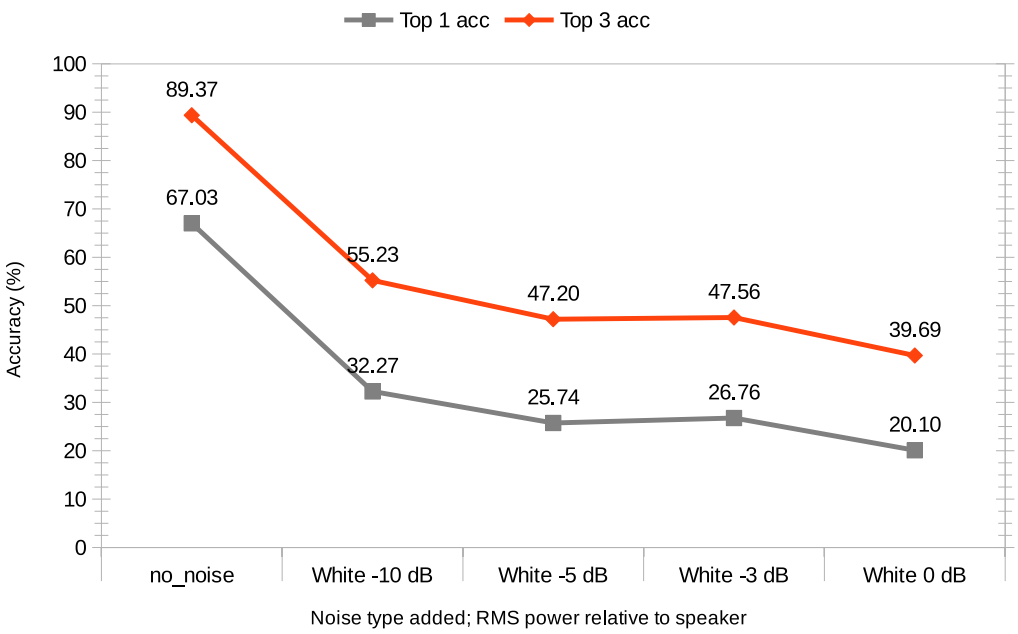


Figure 3.18: TCDTIMIT test set: Performance with white noise
(2 layers, 256 units/ layer network)

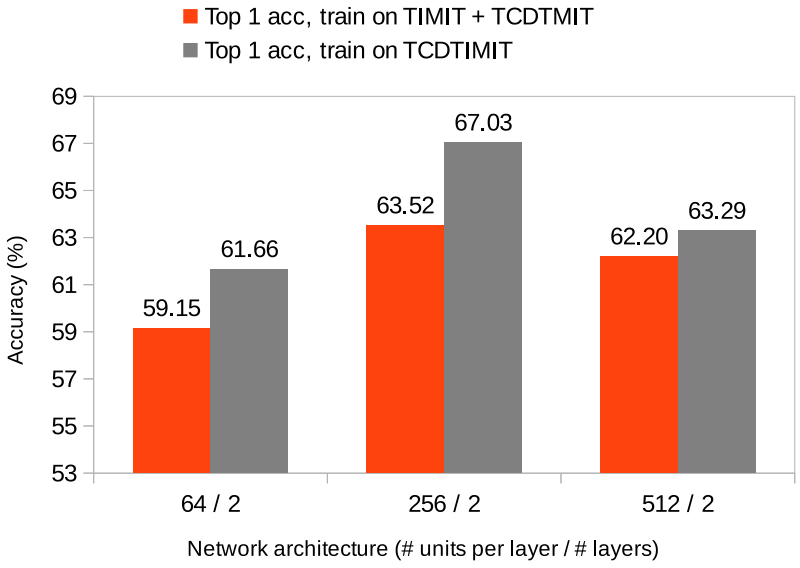


Figure 3.19: Performance difference on TCDTIMIT test set between
a network trained only on TCDTIMIT train data and
a network trained on both TIMIT and TCDTIMIT train data
(2 layer, 256 units/ layer network)

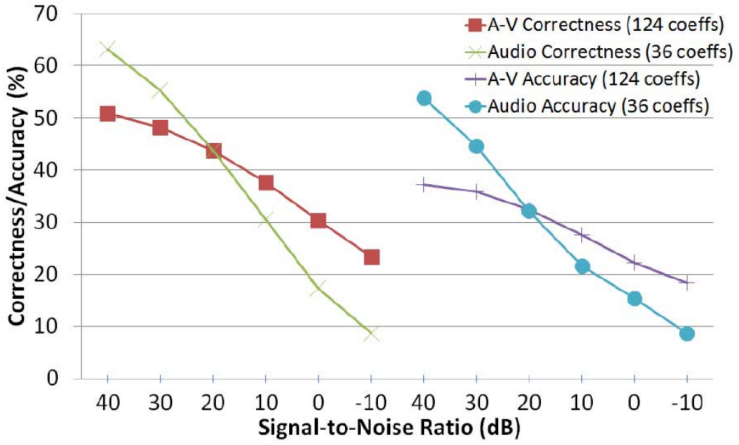


Figure 3.20: Baseline performance on TCDTIMIT under noisy audio conditions. Only the audio-only curve for the correctness is relevant for comparison with the other results from this section.

3.7 Conclusion

Many different network architectures were explored. Single-layer LSTM networks were compared with multilayer LSTM networks, with multilayer networks taking the upper hand. Networks with different numbers of units per layer were compared. There seems to be an optimal combination of number of units and number of layers. The 2-layer and 3-layer networks with 256 and 512 LSTM units per layer performed best overall (Section 3.3).

The influence of the number of MFCC features as inputs to the network was examined, and the standard in the field of using 13 MFCC coefficients plus first and second derivatives was shown to deliver the best results. Bidirectional networks were compared with unidirectional networks, and performed significantly better. They should be used wherever possible, if the application and available resources allow it (Section 3.4).

Finally, the performance of the networks was analyzed under noisy circumstances, and it was shown that performance is degraded significantly even with relatively low noise levels. (Section 3.5). A technique to tackle this problem will be discussed in Chapter 5.

The state-of-the-art network uses a pre-trained language model in addition to a phoneme-prediction network [10]. It achieved 82.3 % correctness on TIMIT (see Table 3.1). However, it didn‘t make use of predefined phoneme time labels, instead selecting the valid network predictions using CTC.

The networks trained in this work use a only phoneme-prediction network and use the time labels to select valid predictions. The best performing network achieved 82.04 % correctness on TIMIT. The TCDTIMIT baseline network [1] achieved 65.47%; the best performing network from this work achieved 67.03 % correctness. The results on both datasets are in line with expectations.

Chapter 4

Lipreading

4.1 Introduction

Humans have a very well developed system for producing all kinds of sounds voluntarily. The auditory system, vocal chords, shape and volume of mouth, stance of tonge and lips all have an important influence on the produced sounds. The idea of lipreading is making use of visual information of the stance of the mouth in addition to the produced audio. Lipreading is something that most people do unconsciously. It is very much something that works 'behind the scenes' as we normally don't pay any attention to it, but in specific situations we do make use of lipreading. When a movie is playing with dubbed audio it's immediately clear that something is wrong, and the movie is harder to follow than a non-dubbed one. When conversing, we tend to look at the other persons face not just to see their expressions, but also to gather extra information about what they are saying apart from what we hear. This is especially clear in noisy environments like parties.

Learning to read lips by a human requires a lot of training and knowledge. It is not something everyone can do easily, and this is very inconvenient for hearing-impaired people. Sign language was developed, but an alternative communication approach could be to simply form the shapes of the words with the mouth. The partner could then lipread what the first person said. Because lipreading is so hard to learn, this method is not often used. If automated lipreading could be developed and possibly surpass human performance it would substantially simplify the lives of hearing-impaired people.

In the past, designing such a system required much effort and extensive preprocessing of the visual data. The mouth image was analyzed, contours were searched and features extracted using tools like the Karhunen-Loève Transform [52]. These features were then processed by a Hidden Markov Model.

Systems fully based on neural networks on the other hand require no manual feature recognition efforts. Recent results in image recognition competitions show that they can perform better than HMMs and even better than humans in for example the benchmark ImageNet [53] (Figure 4.1). This work will analyze performance of neural networks on the lipreading task, which is essentially also image classification.

Figure 4.1: Performance evolution on ImageNet benchmark [54]

Most work done on automated lipreading constrains the data to specific situations with a very limited vocabulary. High recognition accuracies can already be achieved there [55]. Chung et al. [56] expanded this to a much broader field of application. They use almost 5000 hours of video from BBC programs (118.000 sentences) to train on, and combine a CNN-LSTM network for lipreading with LSTM networks for audio processing. The output features are then fused by another network that uses attention mechanisms, after which classification is performed to produce word-level predictions.

Automated lip reading has not received nearly as much attention as audio recognition as it doesn't lend itself to easy classification in categories, like phonemes can classify human speech. There have been some attempts to do the same for lipreading, and the concept of 'visemes' has emerged as an way to classify different shapes of the mouth. Each viseme can be linked to corresponding phonemes. Several phonemes are mapped to one viseme as many phonemes are visually indistinguishable. A commonly used mapping is the Neti map [57], shown in Figure 4.2. In [48] different mappings were analyzed and this mapping proved to be best suited to classification. However, some information is lost when using a mapping such as this one. Especially for multimodal networks it is best to preserve as much information as possible in all parts of the network. Therefore the networks are always trained for phoneme classification, unless specified otherwise.

The used dataset and the required preprocessing are be discussed in Appendix C, Section C.2.2. This work focuses on frame-level classification, so words will not be considered. Performance for phoneme classification is discussed in Section 4.4.2. In order to compare with the TCDTIMIT baselines, performance for viseme classification will also be discussed (Section 4.4.7).

Professional human lipreaders use lots of contextual information to determine what the speaker has said. They use information about the sequence in which these

Viseme	TIMIT Phonemes	Description
/V1	/ao/ /ah/ /aa/ /er/ /oy/ /aw/ /hh/	Lip-rounding based vowels
/V2	/uw/ /uh/ /ow/	”
/V3	/ae/ /eh/ /ey/ /ay/	”
/V4	/ih/ /iy/ /ax/	”
/A	/l/ /el/ /r/ /y/	Alveolar-semivowels
/B	/s/ /z/	Alveolar-fricatives
/C	/t/ /d/ /n/ /en/	Alveolar
/D	/sh/ /zh/ /ch/ /jh/	Palato-alveolar
/E	/p/ /b/ /m/	Bilabial
/F	/th/ /dh/	Dental
/G	/f/ /v/	Labio-dental
/H	/ng/ /g/ /k/ /w/	Velar
/S	/sil/ /sp/	Silence

Figure 4.2: The phoneme-viseme mapping from [48]. The phonemes are mapped to 13 viseme classes, of which one represents silence.

visemes occur, and this helps to narrow down the possible word combinations. .If subsequent images from a video are available, it is possible to improve recognition by utilizing the time information using RNNs as is done in audio speech recognition (Section 4.3.4).

4.2 Baseline Performance

The dataset used for visual (and audio-visual) SR is TCDTIMIT as discussed in Chapter C. An overview of other audio-visual datasets is given in [58]. The preprocessing of the dataset for lipreading is described in Appendix C, Section C.2.2.

Lipreading performance on TCDTIMIT varies a lot between lipspeakers and volunteers. This does make sense as it is professional lipspeakers articulate more clearly, which makes recognizing them easier. This can even be seen manually. On TCDTIMIT, viseme classification performance was around 57% for the lipspeakers and 42% for the volunteers.

There is not much literature on frame-level lipreading. Most results focus on word-level predictions as obtaining phoneme-level labels is quite hard and costly when done manually.

The results from [56] are shown in Figure 4.4. These are character-level and-word-level predictions on a very different dataset, with attention mechanism and several other techniques employed. Results are not directly comparable but they can give an indication of expected performance. Character-level predictions are around 40-55% depending on the network. The characters used are the letters of the alphabet, plus

LIPSPEAKER RESULTS ON LIPSPEAKER-ONLY TRAINED HMMS VERSUS VOLUNTEER RESULTS ON VOLUNTEER-ONLY TRAINED HMMS

	Lipspeakers		Volunteers	
	Train set	Test set	Train set	Test set
%correct	60.38	57.85	42.69	41.98
%accuracy	56.45	52.74	36.05	34.54

Figure 4.3: Baseline performance on TCDTIMIT (viseme classification)

some punctuation marks and padding characters. This is somewhat close to phoneme classification, and can serve as a reference.

Method	SNR	CER	WER
Lips only			
Professional‡	-	58.7%	73.8%
WAS	-	59.9%	76.5%
WAS+CL	-	47.1%	61.1%
WAS+CL+SS	-	44.2%	59.2%
WAS+CL+SS+BS	-	42.1%	53.2%

Figure 4.4: Performance metrics in [56]. CER = character error rate (a-z + 1-10 + ',!?:''). WER = word error rate.
CL: Curriculum Learning; SS: Scheduled Sampling; BS: Beam Search

4.3 Network Architecture

As the lipreading tasks requires extensive image processing and Convolutional Neural Networks are currently the best networks for these tasks, they are used here. For background on Convolutional Networks, see Chapter 2. Several CNN architectures were explored to perform lipreading. They are first described here. Their performance for lipreading is analyzed in Section 4.4.

1. A network designed by Chung et al. as a component in the WLAS (Watch-Listen-Attend-Spell) architecture [56]
2. A network developed by Courbiaux et al. based on VGG that also showed good performance on the CIFAR10 benchmark [51]
3. A deep residual network with 50 layers, developed by He et al [59]

4.3.1 WLAS network

This CNN network used for lipreading in the paper [56] by the Oxford and Deepmind researchers is proven to work quite well, and is therefore used in this work. It uses five CONV layers with ReLu activations, three POOL layers performing max-pooling,

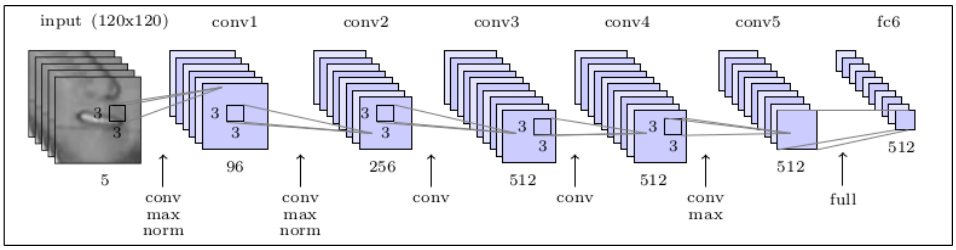


Figure 4.5: The CNN network used in WLAS

Layer	Support	Filt dim.	# filts.	Stride	Data size
conv1	3×3	5	96	1×1	109×109
pool1	3×3	-	-	2×2	54×54
conv2	3×3	96	256	2×2	27×27
pool2	3×3	-	-	2×2	13×13
conv3	3×3	256	512	1×1	13×13
conv4	3×3	512	512	1×1	13×13
conv5	3×3	512	512	1×1	13×13
pool5	3×3	-	-	2×2	6×6
fc6	6×6	512	512	-	1×1

Figure 4.6: The CNN network used in WLAS

and a batch normalization layer after each POOL layer. Other parameters of the network architecture are shown in Figure 4.6.

4.3.2 VGG based network

This network was designed by Courbiaux et al. performed well on the CIFAR10 benchmark[51]. It was used there as a binary network, but here the network uses floating point weights and biases. Its structure is as follows:

$$[2xCONV1]->POOL1->[2xCONV2]->POOL2->[2xCONV3] \\ ->POOL3->[FC1]->FC_{softmax} \tag{4.1}$$

The CONV layers use ReLu activations. CONV1 uses 128 filters, CONV2 256 filters, CONV3 512. The POOL layers perform max-pooling with a pool-size of [2,2]. The FC1 layer has 256 neurons. A batch normalization layer is added in between each of the double CONV layers, and after each POOL layer.

4.3.3 Residual Neural Networks

Deep multilayer Neural Networks have been shown to have advantages over single-level networks, as deeper networks can work with more layers of abstraction. This makes them better at forming hierarchies of feature recognition, which helps with classification.

However, deeper networks are harder to train as the weight gradients have to propagate much further through the network during backpropagation and are faced

with the ‘vanishing gradients‘ problem. ResNets have been designed to work around this issue and as a result can be very deep. They currently achieve some of the highest scores on ImageNet. ResNets work by adding extra ’shortcut’ connections in the network, skipping several layers (Figure 4.7). [7]

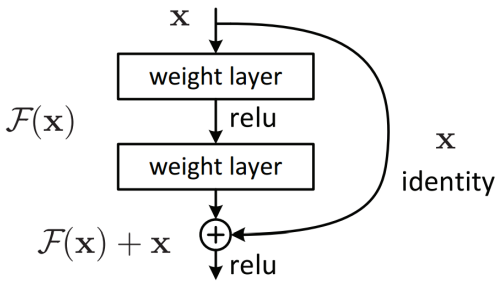


Figure 4.7: A ResNet building block

4.3.4 CNN-LSTM networks

As any practical lipreading system will not be based on single images but on a sequence of images (a video), it makes sense to try to use this information. This can be done with Recurrent Neural Networks. The RNNs in Chapter 3 used the MFCCs of the processed audio data as inputs, which contained a sequence of timeframes, with several features per timeframe. The image frames from a video can be converted to a sequence of features that a RNN can use as inputs: a CNN processes each of the images sequentially, and the output features are stored in a sequence that the RNN can use.

Just like in Chapter3 the RNN architecture is a bidirectional deep LSTM network. Several architectures will be explored and analyzed in Section 4.4.

There are two possibilities to connect the CNN with the LSTM layers: the first one is to use the output features of the convolutional layers (called RF for ’raw features’); the second possibility is to use the phoneme predictions of the CNN (called PF or pre-classified features) as inputs to the LSTM network. A visualization of the first architecture is shown in Figure 4.8, the structure of the second architecture is shown in Figure 4.9.

4.3.5 Fully Connected Layers

Just like RNNs for audio recognition, CNNs don’t produce predictions of the classification labels, but certain features that they ’discovered’ in the data. These features are then fed through a Fully Connected layer which performs the classification.

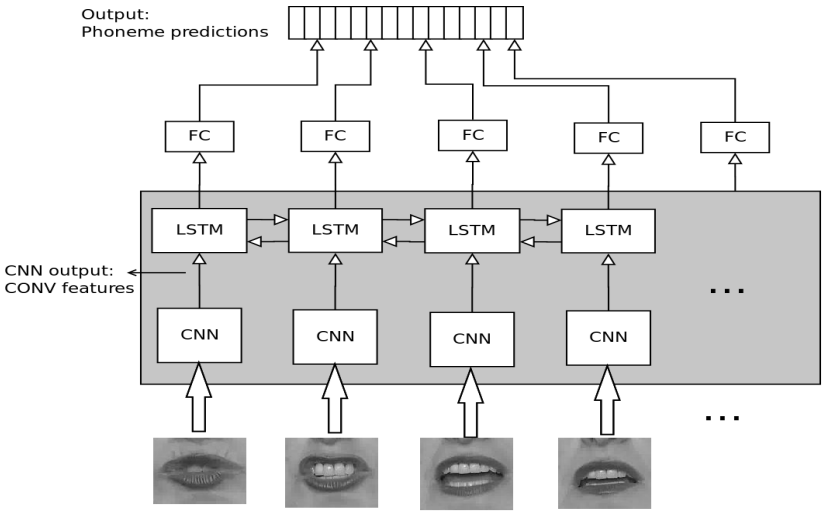


Figure 4.8: Architecture of CNN-LSTMs using Raw CNN features (RF). The LSTM network is shown unrolled. It processes the sequence formed by the outputs of the CNN for all of the images, and does so in two directions (it's a bidirectional network). See Section 2.4. The FC classification layer processes the output at each timestep to generate phoneme predictions for each frame of the sequence.

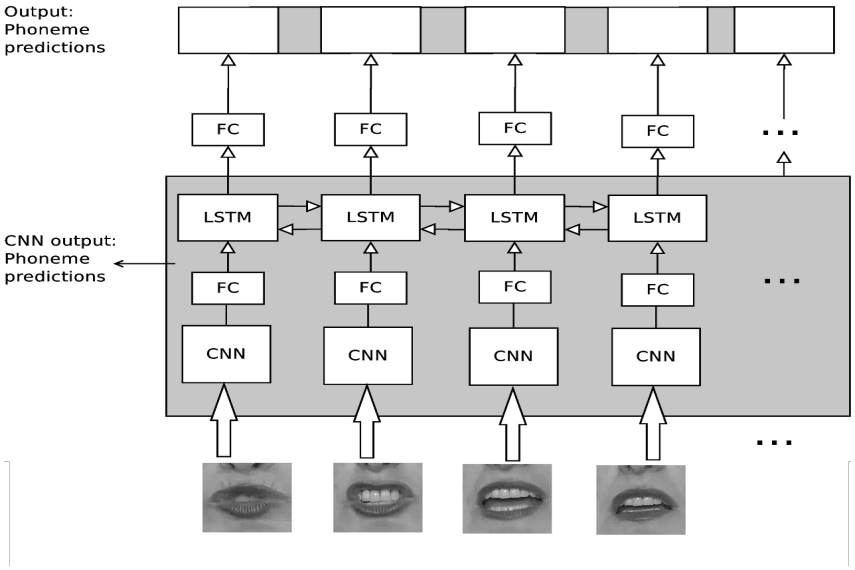


Figure 4.9: Architecture of CNN-LSTMs using Pre-classified CNN features (PF). The LSTM network is shown unrolled. It processes the sequence formed by the outputs of the CNN for all of the images, and does so in two directions (it's a bidirectional network). See Section 2.4. The FC classification layer processes the output at each timestep to generate phoneme predictions for each frame of the sequence.

4.4 Performance Analysis

4.4.1 Overview

The data sizes for images are much larger than for audio files. This is an issue when evaluating different network architectures, as more memory, storage and computational power is needed.

In order to evaluate networks relatively quickly, the 'lipspeakers' in the TCD-TIMIT dataset are used. The dataset contains data from three professional lipspeakers who each speak 397 sentences. This is already a fairly large dataset in itself, and can be used. Networks that perform well on this subset of the data will most probably also perform well on the whole dataset. Results reported below are from networks trained on the lipspeaker dataset, unless mentioned otherwise.

In [1] it is mentioned that lipreading performance on the lipspeakers was much higher than on the volunteers Figure 4.3. This will be compared in Section 4.4.8.

The learning rate for training is set to 0.01 at the beginning, and decreased when performance does not improve in an epoch. When performance didn't improve for several epochs, training is stopped. Glorot weight initialization is used (the default in Lasagne) [27].

Results reported include both the top-1 classification accuracy and top-3 accuracy (see Section 2.3. Many reports in literature do not do this and only use the top-1 accuracy, but when the network output features are used in further networks, a lot of information is lost if only the class with the highest score is passed on.

Especially when the lipreading networks are used for phoneme classification instead of viseme classification, multiple phonemes will map to (almost) indistinguishable visual features (=visemes). The network output probabilities for an image will then be distributed among the different phonemes that map to the same viseme, and much of the information is lost if only the highest probability score is taken into account.

4.4.2 Phoneme classification

As phoneme classification can be seen as a generalization of viseme classification (phonemes can be mapped to visemes), this task is analyzed first. The best performing networks are then also used for viseme classification. First performance of different CNN networks is discussed, then a LSTM network is added on top of the CNN.

4.4.3 Convolutional Neural Networks

A summary of performance results is shown in Figure 4.10. Performance figures are reasonably close to each other, with the ResNet and the VGG network outperforming the WLAS network. This comes at a cost however. The networks VGG network is much larger and deeper, making training harder and increasing complexity of the network. On a NVIDIA GTX 1060 GPU, the WLAS network takes about 350s per epoch; the VGG network around 1000s, and the ResNet around 310s. Apart from training duration, the network architecture determines the number of weights in the

convolutional part of the network, the number of weights in the Fully Connected part of the network, as well as the number of output features of the CNN (hereafter called 'CONV output features'). The number of output features has important consequences for the network complexity if the network outputs are to be used in subsequent networks (see Section 4.4.4). A comparison between the number of parameters is shown in Table 4.2.

For convolutional networks, the computational complexity is much more dependent on the number of weights in the convolutional layers than in the Fully Connected Layers. This is because each convolutional weight is reused many times for one input image ('parameter sharing', see Section 2.5), while each weight in the FC part is just used once for each input. Large numbers of weights in the FC part does increase required memory size as each weight needs to be stored.

The VGG-based network doesn't use as many CONV and POOL layers as the WLAS network, but this results in much larger output features (see Table 4.1). Because of the huge number of features classification gets harder, which is several FC layers are needed. These layers have a very large number of neurons because the feature sizes are so large (see Table 4.2). Classification performance is quite good, but not much better than for the other networks.

The ResNet50 network has a very small number of output features, resulting in very little FC neurons. On the other hand, it has many convolutional layers which are computationally very intensive.

The WLAS network strikes a good balance between number of CONV output features and limiting the complexity of the convolutional part of the network. The features can be used by subsequent networks (eg a LSTM network or to combine with an audio recognition network) without requiring huge amounts of FC neurons. The performance is also quite good. For these reasons, this WLAS network is chosen as the lipreading network for multimodal speech recognition (see Chapter 5).

Network	# CONV output features
WLAS	25,088
VGG based	115,200
ResNet50	2048

Table 4.1: Comparison of number of CONV output features

Network	# CONV weights	# FC weights
WLAS	6,197,760	978,471
VGG based	4,581,248	59,649,280
ResNet50	23,554,944	79,911

Table 4.2: Comparison of number of weights for CNNs

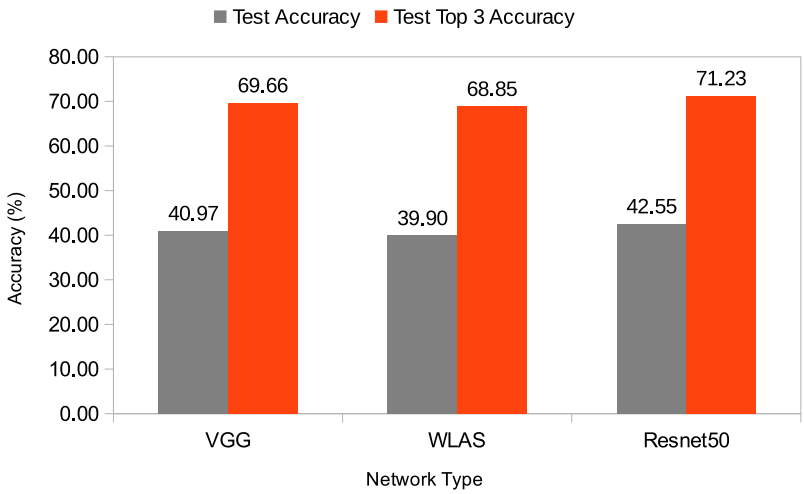


Figure 4.10: Phoneme classification performance of CNNs

4.4.4 Convolutional Neural Networks and LSTM networks

As mentioned in Section 4.3.4, it is expected that lipreading networks can benefit a lot from incorporating time information through the use of LSTM layers on top of the CNN networks. There are some implementation issues when combining these networks due to limitations in the library; these are discussed in Chapter 5.

The results shown in this section all use the WLAS network as it is a relatively simple network with a manageable number of CONV output features, trains faster and still performs well.

There is are several possibilities to join the CNN and LSTM networks:

1. *Raw CNN features (RF)*: Using intermediate output features of the CNN network as inputs to the LSTM network. The intermediate features are the features just before the FC layers of the CNN. See Figure 4.8.

2. *Pre-classified CNN features (PF)*: Before connecting to the LSTM network, the CNN output features are classified to generate phoneme predictions. This means that the LSTM network will have 39 input features (a probability value for each phoneme). See Figure 4.9.

The network architecture is noted by the CNN type, then 'RF' (Raw CNN features) or 'PF' (pre-classified CNN features), followed by the number of units per LSTM layer. If the LSTM network contains multiple layers, a slash and the number of layers are added at the end. For example, a network using the WLAS CNN with a pre-classification layer and then a 2-layer LSTM of 256 unit per layer is denoted by 'WLAS (PF) + 256 / 2 LSTM'.

All of the evaluated networks use bidirectional LSTM networks, so in fact there are twice as many LSTM units on a layer as displayed here, but for simplicity only the number of forward LSTM units is shown.

The phoneme classification performance for several RF-type CNN-LSTM networks is given in Figure 4.11. For PF-type networks, the classification performances are shown in Figure 4.12.

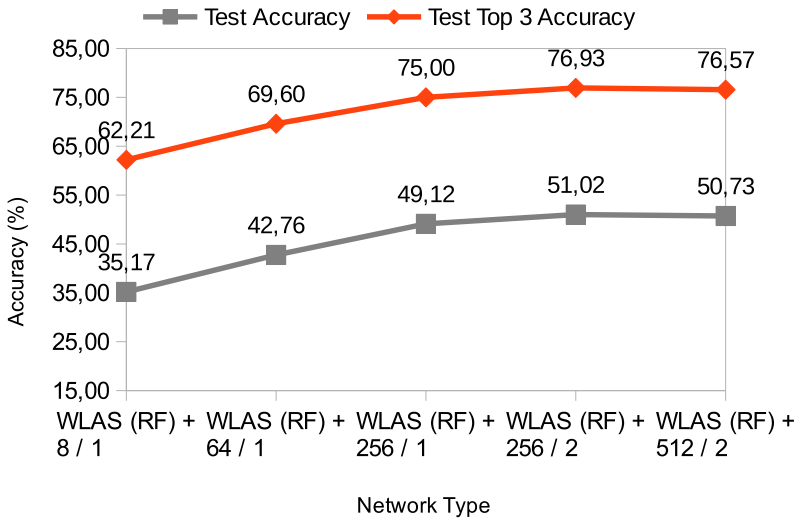


Figure 4.11: Phoneme classification performance of CNN-LSTMs (Raw CNN features)

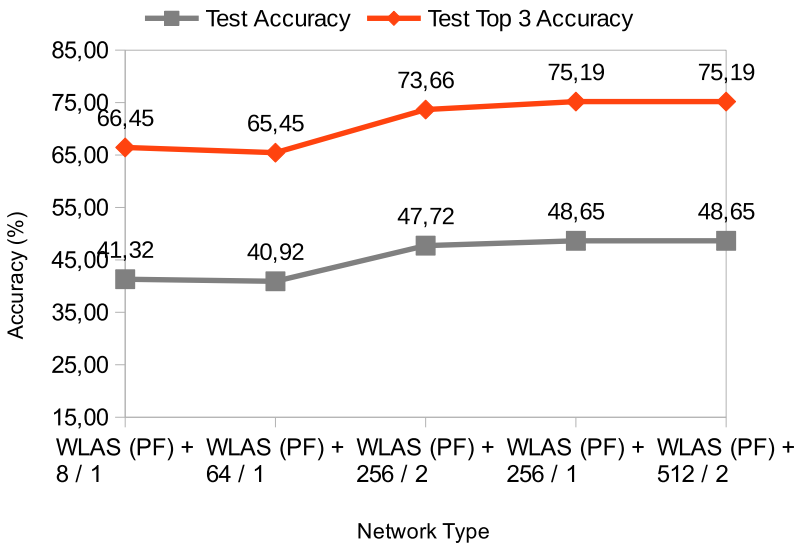


Figure 4.12: Phoneme classification performance of CNN-LSTMs (Pre-classified CNN features)

All of the CNN-LSTM networks significantly outperform the networks with only CNNs. This confirms the hypothesis that incorporating time information is very important in lipreading. The achieved performance scores are in line with the scores by Chung et al [56]. In Section 4.4.7 it is shown that when these networks are applied to viseme classification, they significantly outperform the HMM-based classifiers that were used for the TCDTIMIT baseline results (see Figure 4.3).

Contrary to the audio networks from Chapter 3, multilayer LSTM networks don't seem to improve performance here. The reason might be that the CNN already 'discovered' and separated important features out of the input image. Making networks deeper has the same goal, so is redundant in this case. The main purpose of the LSTM layers is to add time information to the CNN features, and they do that very well as classification scores rise by about 8 % (absolute).

4.4.5 Comparison Raw and Pre-classified CNN features

It is clear from figure Figure 4.13 that using raw CNN features generally yields better results. The disadvantage is that the LSTM networks get much more complex as they have to accommodate much larger input features (see Section 4.5). Depending on the application one or the other should be preferred.

Only the PC network with a very small number of LSTM units manages to perform better than the respective RC network. This might be that the RC network doesn't have the extra FC network for classification and the LSTM network is very small, so there are not enough neurons to perform proper classification. The PC network doesn't have this problem as there is already a classification early on.

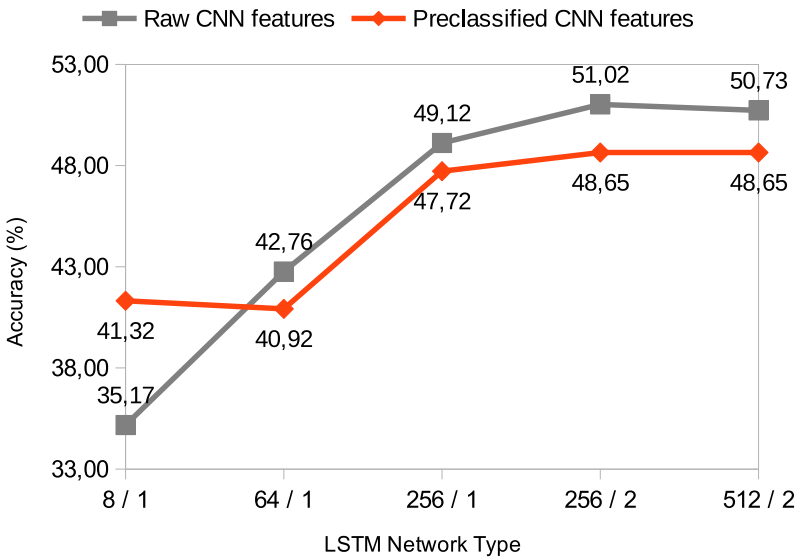


Figure 4.13: Comparison of phoneme classification performance between CNN-LSTMs using pre-classified (PF) and raw (RF) CNN features

Network Type	# CONV weights	# class. weights (after CNN)	# LSTM weights	# class. weights (after LSTM)
RF 8	6,197,764	0	804,079	351
RF 64	6,197,765	0	6,475,367	2,535
RF 256 / 2	6,197,766	0	27,017,767	10,023
RF 256	6,197,767	0	26,491,175	10,023
RF 512 / 2	6,197,768	0	56,656,935	20,007
PF 8	6,197,768	978,467	2,511	351
PF 64	6,197,768	978,467	62,823	2,535
PF 256 / 2	6,197,768	978,467	1,367,591	10,023
PF 256	6,197,768	978,467	840,999	10,023
PF 512 / 2	6,197,768	978,467	5,356,583	20,007

Table 4.3: The number of network weights of a CNN-LSTM networks, depending on the LSTM network architecture. All networks use the WLAS CNN. The table shows the number of weights in the convolutional layers, in the (fully connected) classification layer after the CNN, in the LSTM layers, and the in the fully connected classification layer (after the LSTM). There is a significant difference in the number and the distribution of weights between networks using Raw CNN features (RC) and pre-classified CNN features (PC) as inputs to the LSTM network. RF networks have much more weights in the LSTM layers while the PF networks have additional weights because of the extra FC layer after the CNN.

Network Type	Top 1 performance	Top 3 performance
WLAS + LSTM, PC 256 /2	48.65	75.19
ResNet50 + LSTM, PC 256 /2	49.05	75.16

Table 4.4: Phoneme classification performance of CNN-LSTMs with ResNet50

4.4.6 CNN-LSTM with other CNNs

The previous networks all used the WLAS CNN [56] and added LSTM layers on top. This section shows performance scores when the CNN is replaced by a ResNet. Replacing the CNN by the VGG-based network was attempted, but resulted in networks that were too large to train due to hardware constraints, so results have not been obtained. The performance for a network with pre-classified CNN features are shown in Table 4.4. Results for the ResNet are slightly better, but as shown in Table 4.2 the CNN is also much larger.

4.4.7 Viseme classification

Viseme classification networks are almost exactly equal to the phoneme classification networks; only do they have fewer categories to classify. The softmax FC layer has only 12 output neurons instead of 39 for phoneme classification (see Section 4.1).

Performances are expected to be higher as many phonemes map to single visemes, and Figure 4.14 illustrates this. There is a very similar trend in the performance scores; CNN networks only perform significantly below CNN-LSTM networks.

These performance figures are very competitive with the baseline figures given for the TCDTIMIT dataset [1], which are shown in Figure 4.3. The best trained network achieves a score of 68.46 % while the best correctness score in [1] was only 57.85%.

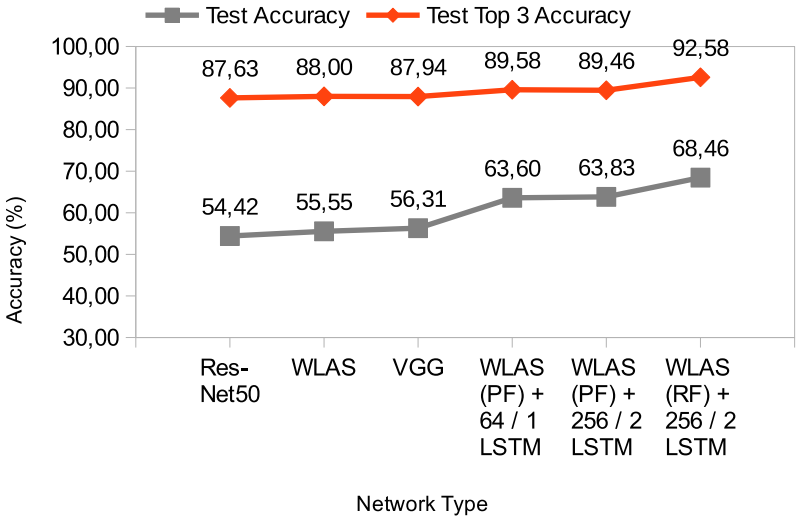


Figure 4.14: Viseme classification performance of CNNs and CNN-LSTMs

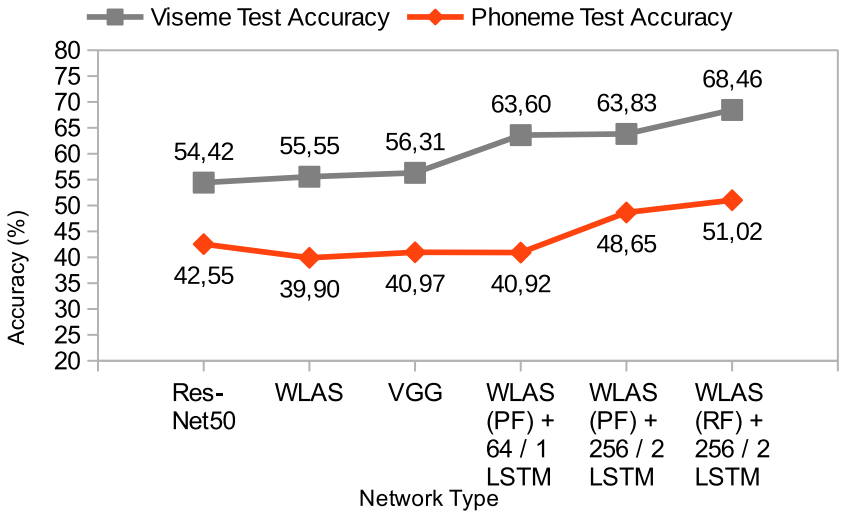


Figure 4.15: Comparison of phoneme and viseme classification performance

4.4.8 Performance on TCDTIMIT volunteers

The CNN-LSTM networks performed well on the lipspeakers dataset. This section shows the results of the same networks, but trained and evaluated on the volunteers data of TCDTIMIT. TCDTIMIT provides a train/test split for this data, and that split is also used here (Appendix C). The test dataset contains videos from other speakers than the train dataset, so performance scores evaluate speaker-independent classification.

Performance on this volunteer test dataset is shown in Figure 4.16, and a comparison with the lipspeaker performances is shown in Figure 4.17. For the last it is important to keep in mind that the volunteer scores measure speaker-independent performance, while the lipspeaker scores measure speaker-dependent performance.

This speaker dependency is a possible explanation for the discrepancies between the lipspeaker and volunteer performance, as also noted in [1]. For the lipspeakers, a speaker independent test set could not be made as there are only three lipspeakers in TCDTIMIT.

It would be interesting to evaluate the speaker-dependent performance on the volunteers, but this was not attempted here. In [1] it was noted that even for networks trained with speaker-dependency the scores on the volunteer test set were still significantly lower than for lipspeakers. This confirms the theory that lipreading is very much dependent on the way an individual pronounces the phonemes, and how distinct the visual features are. The lipspeakers' mouth movements are more distinct and therefore the visemes are easier to classify.

To achieve higher performances, a larger dataset with more speakers and a wider variety of speakers would be needed, as for example the one used in [56]. Creating such a dataset is hardly feasible with phoneme-level labeling, which is why much research has moved away from phoneme-level classification toward word-or sentence-level classification (Section 3.1.6, Section 4.2).

4.5 Computational Considerations

4.5.1 Network Architecture

Network performances were shown in Section 4.4.2 and Section 4.4.4. It was also shown that the network architecture determines the number of weights in each part of the network (convolutional, recurrent, classification). The WLAS CNN delivered the good performance while keeping the number of weights small (see Table 4.2). It was also a relatively small convolutional network, which saves resources. This is important because most of the computation time spent during evaluation of a CNN-based classification network is in the convolutional layers [60].

A rough estimate of the number of MAC (multiply-accumulate) operations needed to process one image by the WLAS CNN is shown in 4.5. This is calculated using the network layer size specifications from Figure 4.6. Each layer can be described by the size of its input features (MxN), the depth of the input features (Q), the depth of its output (R, the number of filters in that layer), the size of the convolutional

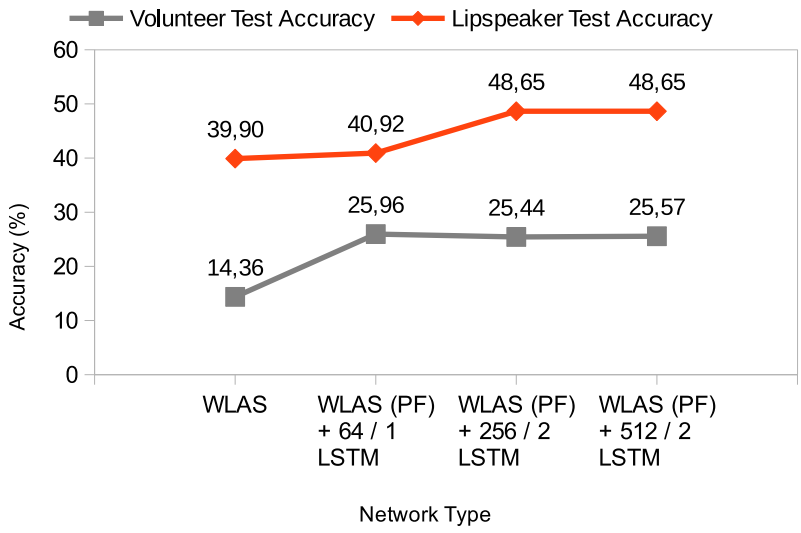


Figure 4.16: Volunteer phoneme classification performance

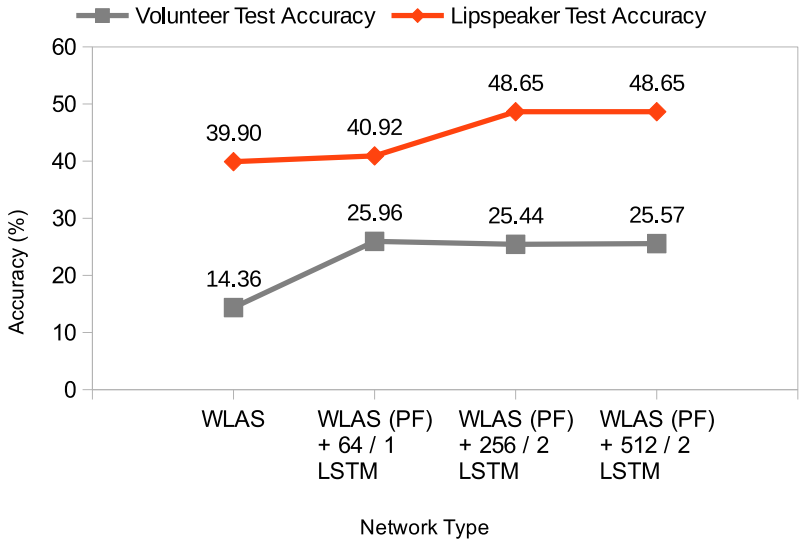


Figure 4.17: Phoneme classification performance: network trained on lipspeakers and tested on lipspeaker test set, compared to network trained on volunteers and tested on volunteer test set

kernel (KxL) and the stride (S) with which that kernel is applied. The workload is then on the order of $O(R * Q * M * N * K * L/S^2)$. If the input features and the convolutional kernel are square, this reduces to $O(R * Q * M^2 * K^2/S^2)$ [60].

For CNN-LSTM networks, the number of parameters in the network is largely

	Q	R	M (=N)	M*N	K (=L)	K*L	S	S^2	MAC	estimate
CONV 1	1	96	120	14400	3	9	1	1	12.44	e+06
POOL 1	96	96	109	11881	2	4	2	4	285.14	e+03
CONV 2	96	256	27	729	3	9	2	4	40.31	e+06
POOL 2	256	256	13	169	2	4	2	4	10.82	e+03
CONV 3	256	512	13	169	3	9	1	1	199.36	e+06
CONV 4	512	512	13	169	3	9	1	1	398.72	e+06
POOL 5	512	512	6	36	2	4	2	4	4.61	e+03
FC 6	512	39	1	1	/	/	/	/	19.97	e+03
								Total	651.15	e+06

Table 4.5: Estimate of computational complexity of WLAS for the processing of one image. Most of the computation is in the convolutional layers, especially in the deeper layers.

determined by the size of the LSTM network and the type of connection between CNN and LSTM. PF CNN-LSTMs use 978.471 more FC neurons than the RF CNN-LSTMs in the CNN part of the network because classification already takes place there (see Table 4.3). On the other hand, RF CNN-LSTMs require many more weights to deal with the larger number of CONV output features (see Table 4.3), which are fully connected to the first layer of the LSTM network. There are three connections per LSTM unit and each of these connections requires a weight, as explained in Section 2.4.2. This is the cause of the large increase in network parameters.

4.5.2 Reducing weight precision

Apart from choosing a network architecture fit for the specific application, another technique to reduce the computational complexity is to reduce the precision of the weights [23]. There are several techniques possible. The weights can simply be rounded after training so that the inference stage is much cheaper to execute, with a trade-off between the amount of rounding and the savings in computation cost. Another method is to train the networks with strict limitations on the weights. An extreme example is to use 'binary weights', where any weight can only take on the values 0 or 1 [51].

Rounding the weights By using 16-bit floating point numbers instead of 32-bit during inference it is possible to reduce the hardware requirements for the network significantly. The results in Table 4.6) show that performance is hardly impacted. It should be possible to go even lower in precision [23], but that was not attempted in this work.

Binary Networks Networks with binary weights and activations ('Binary networks') require a few small modifications to the training method [51]. Some changes to the network have to be made: batch normalization is very important for good performance with binary weights, and an extra batch normalization layer is used

Network Type	float32 top 1 acc.	float16 top 1 acc.	float 32 loss	float16 loss
CNN	39.9074	39.8714	2.7834	2.7943
CNN-LSTM 1x64	40.9252	40.9248	2.2748	2.1742
CNN-LSTM 2x256	48.6365	48.6326	2.1094	2.1394

Table 4.6: Phoneme classification performance with reduced precision network parameters, and Cross-Entropy loss

compared to the standard networks. The activation functions are changed, and the layers are modified to make it possible to train with binary weights. The loss function used is hinge loss (compared to the more common cross-entropy loss in non-binary networks). It was attempted to make some changes to this so the binary network could be incorporated with LSTM layers and used for multi-modal sensor fusion, but this was unsuccessful. It could be interesting to investigate this further in future work.

Performance of standard CNNs is compared with binary CNN networks in Table 4.7. The binary networks are implemented using the code published on GitHub from [51].

The WLAS CNN doesn't perform well in binarized form. The VGG-based network does not suffer such ar large performance decrease. Possibly this is due to the fact that that VGG-based network has more FC layers. A binarized version of the ResNet network has not been evaluated as there were no binarized versions of the residual blocks available yet in the software.

Network Type	Top 1 Acc
WLAS CNN	39.90
WLAS Binary CNN	21.26
VGG-based CNN	40.97
VGG-based binary CNN	35.74

Table 4.7: Phoneme classification performance of binary networks compared with non-binarized networks

4.6 Conclusion

Lipreading is a difficult problem, but good performance scores are obtained with several networks, using the TCDTIMIT dataset.

Many different network architectures for lipreading are explored. First, standard Convolutional Networks were used. A network used in [56] (called 'WLAS') that was successful for lipreading is compared with a ResNet with 50 layers and with a VGG-based network. The WLAS network offers a good compromise between performance, complexity and speed of training. It has a limited number of CONV

output features but not too few, so there is not much loss of information when these features are used in further network layers (see Chapter 5) (Section 4.4.3).

To incorporate time information in the lipreading networks, bidirectional LSTM layers are added. They improve performance significantly, showing that time information is very important for lipreading. The best performing network reaches a Top-1 classification score of 51.02% and a top-3 score of 76.93%.

Contrary to the audio networks in Chapter 3, multilayer LSTMs don't improve performance, possibly because the features have already been well separated by the CNN, so making LSTM networks deeper would perform a redundant function while adding complexity (Section 4.4.4).

The CNN and LSTM networks can be joined using either Raw CNN features or pre-classified CNN features. Raw CNN features delivered better performance by around 2 percentage points but requires much larger networks. Depending on the application this may require too much hardware (Section 4.4.5).

The same networks are evaluated for viseme classification, and are shown to perform much better than the baseline results on TCDTIMIT from [1] (57.85%), with the CNN-LSTM networks improving scores by over 10 % (Section 4.4.7).

Performance of these networks is then evaluated on the volunteer dataset of TCDTIMIT. Performance is significantly lower than for the lipspeakers. This is at least in part due to speaker dependency of the trained networks, but also due to the fact that the lipspeakers' mouth movements are more distinct and easier to classify [1] (Section 4.4.8).

Different techniques for reducing computational complexity are analyzed. Reducing weight precision to 16-bit floating point numbers during inference does not impact performance and is an easy way to save resources. A more radical approach constrains the network weights and activations to +1 or -1. This requires modifying the network architecture somewhat and retraining the network from scratch. Training is much slower and performance is affected, but the potential in saved resources is very significant. This technique could be very useful for resource-constrained applications (Section 4.5).

Chapter 5

Multimodal Speech Recognition

5.1 Introduction

In Chapter 1 the motivation for multimodal speech recognition is already given. It is repeated in short here. Speech recognition through audio is much easier than by using lipreading, which is why audio is most often used. In situations where the audio quality is very low however (for example in a car or airplane, at a party or in a busy street) performance decreases sharply. This is shown in Section 3.5.

Adding extra information which is not degraded because of environmental noise could improve performance in these situations. Lipreading is a straightforward way to obtain this extra information. People do it all the time and some people can understand speech only be reading another person's lips. This chapter discusses the architectural possibilities to combine audio and lipreading networks. Several different networks are evaluated and compared with the separate networks for audio and lipreading.

For detailed implementation issues and data preprocessing, see the Appendices (Section C.2.3 and below).

An 'audio frame' is one MFCC frame. There is one such frame every 10ms of a

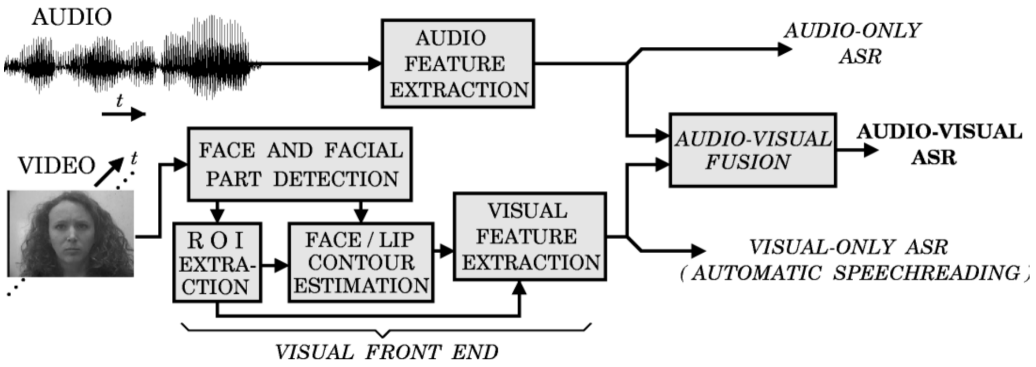


Figure 5.1: An audio-visual speech recognition system [61]

video (see Chapter 3). A 'timestep' is a moment in time when a phoneme is spoken in a video. Each video can be described as a sequence of timesteps. The term 'time frame' is used to indicate the input data to the network at a timestep. There is one image of the speaker's mouth and one corresponding audio frame that belong to each timestep. This audio frame is called a 'valid' audio frame. Other audio frames fall in between two phonemes and are invalid.

5.2 Network Architecture

When designing a network for multimodal speech recognition, there are even more hyperparameters than for lipreading-only or audio-only networks. Much like in the CNN-LSTM networks for lipreading, there is a choice to apply pre-classification before combining the lipreading and audio features, or to simply pass the raw features to the next layers. The first type of networks are indicated below with 'PF' (pre-classified features), the second type with 'RF' (raw features). The RF and PC indications only determine whether the outputs of the lipreading and audio subnetworks are pre-classified. Within the lipreading network, it is also possible to add a pre-classification network after the CNN but before the LSTM layers. This is also indicated in the network name.

For example, a multimodal network using raw audio features, and where the lipreading LSTM receives raw features from the CNN but the lipreading LSTM output is pre-classified before combination with the audio, will be indicated as 'Audio (RF) | Lipreading (PF) CNN (RF) + LSTM'.

The lipreading networks that use classification perform phoneme classification (not viseme classification) in order to conserve the maximum amount of information possible. The lipreading networks use a CNN-LSTM architecture as that performs better than a CNN-only architecture. The CNN network used is WLAS (see Chapter 4).

In summary, the different architectural choices for a multimodal network are:

1. Architecture of lipreading network: CNN or CNN-LSTM. The type of the CNN and the type of the LSTM network on top.
2. Architecture of audio network: type of LSTM network.
3. Outputs of audio and lipreading subnetworks: use raw features (RF) or pre-classified features (PF).
    a) Lipreading raw features + Audio raw features (Figure 5.3)
    b) Lipreading preclassified + Audio raw features
    c) Lipreading preclassified + Audio preclassified
    d) Lipreading preclassified + Audio preclassified (Figure 5.2)
4. Combining the output features of the subnetworks: the number of FC layers and their size (for final classification)

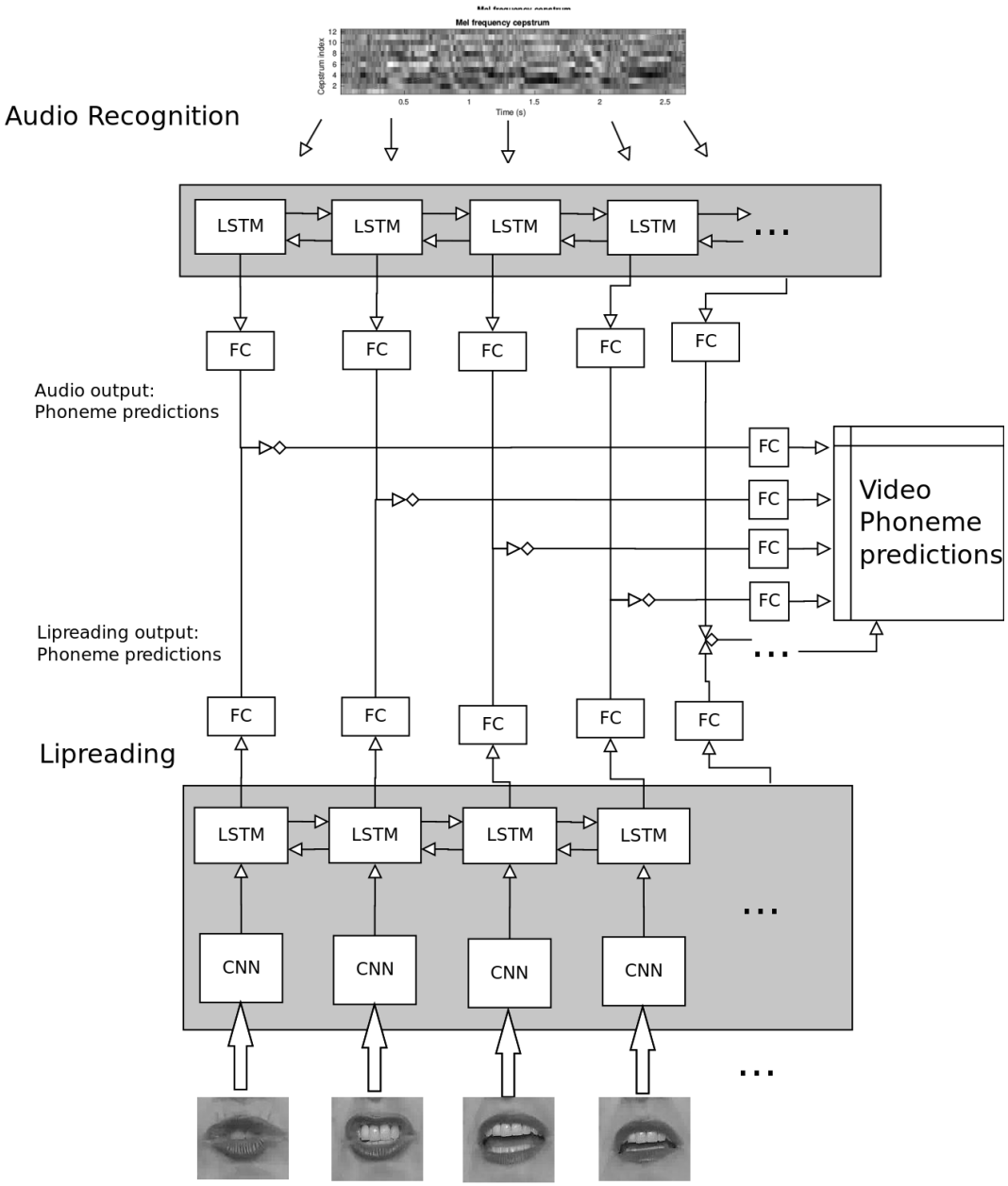


Figure 5.2: Classifying phonemes from a video by a multimodal network (Lipreading PF + Audio PF). The LSTM networks are shown unrolled. The audio LSTM network processes the sequence of the MFCCs of the audio. The lipreading LSTM network processes the sequence formed by the outputs of the CNN for all of the images. Both of the LSTM networks are bidirectional. See Section 2.4. The audio and lipreading features at each timestep are concatenated. At each timestep, the FC classification network processes the concatenated output to generate phoneme predictions for each frame of the sequence.

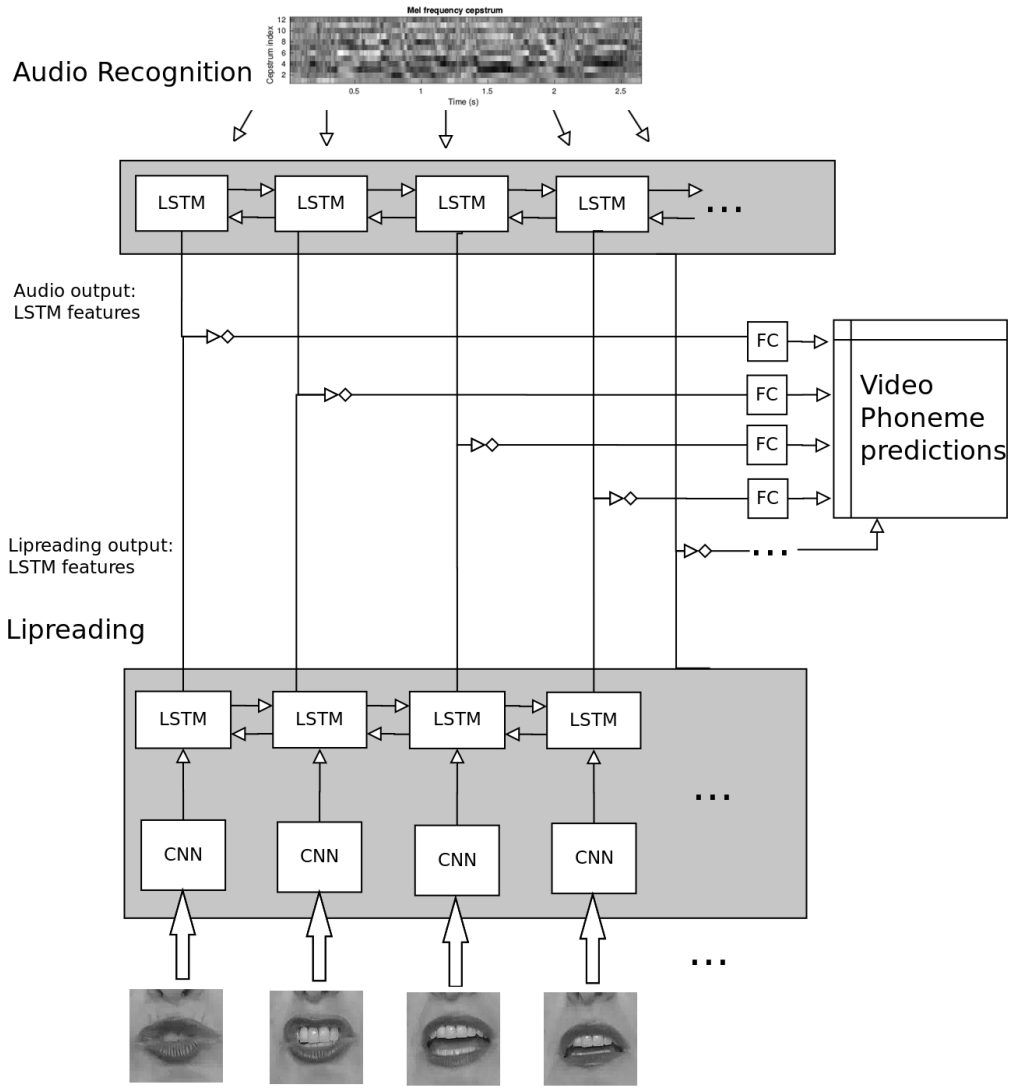


Figure 5.3: Classifying phonemes from a video by a multimodal network (Lipreading RF + audio RF). The LSTM networks are shown unrolled. The audio LSTM network processes the sequence of MFCCs of the audio. The lipreading LSTM network processes the sequence formed by the outputs of the CNN for all of the images. Both of the LSTM networks are bidirectional. See Section 2.4. The audio and lipreading features at each timestep are concatenated.At each timestep, the FC classification network processes the concatenated output to generate phoneme predictions for each frame of the sequence.

5.3 Performance Analysis

5.3.1 Training

Training is done very much like in audio-only or lipreading-only networks. The learning rate for training is set to 0.01 at the beginning, and decreased when performance does not improve in an epoch. When performance didn't improve for several epochs, training is stopped. Top-1 accuracy is used as measure of performance. Glorot weight initialization is used for the combination FC layers [27].

Before training the combined network, the lipreading and audio subnetworks are trained separately and those trained weights are loaded.

5.3.2 Performance for multimodal networks on clean audio

There are two ways to train a multimodal network: the first one is to use 'fixed subnetworks', meaning that the weights of the lipreading and audio networks are fixed during training of the combined network. Only the weights of the FC combination layers are trained as a result. The second way is to train all of the weights of the multimodal network, also the weights of the lipreading and audio networks. This can result in better performance.

An overview of different network is shown in Table 5.1 .

The non-E2E results shown in Table 5.1 shows that a large enough audio network is needed for good performance. It is better to use the output features from the audio network without pre-classification. On the other hand, it seems better to pre-classify the lipreading LSTM network features.

When the subnetwork weights are not fixed, the whole network can be trained (end-to-end training).

Table 5.1 shows the performance difference between different types of multimodal networks that were trained end-to-end and the corresponding network where the subnet weights were fixed. Performance of the end-to-end trained networks is higher by a few percentage points. It increases especially for networks using raw features, as those can truly be trained end-to-end. These types of networks trained end-to-end deliver the best performance for audio-visual speech recognition of all the networks evaluated.

5.3.3 Comparing Audio, Lipreading and Multimodal networks

A comparison of the achieved performance between the different network architectures is shown in Figure 5.4. The audio network is a two-layer bidirectional LSTM network. The lipreading network uses the WLAS CNN with 2 bidirectinoal LSTM layers on top. The multimodal network combines the output predictions from these networks and adds three layers of 512 FC neurons. As discussed in Section 5.3.2, performance can be further improved by training the multimodal network end-to-end.

The best multimodal network (with end-to-end training) outperforms both the audio-only and the lipreading-only networks by a significant margin, and improves on the highest score of the audio networks by almost 14 %.

Network architecture	normal	E2E
Lipreading PF networks		
Audio (PF):LSTM 256/2 | Lip.(PF): WLAS PF + LSTM 256/2 | FC 512/3	74.06	75.87
Audio (PF):LSTM 256/2 | Lip.(PF): WLAS RF + LSTM 256/2 | FC 512/3	72.24	76.35
Audio (RF):LSTM 256/2 | Lip.(PF): WLAS PF + LSTM 256/2 | FC 512/3	77.55	76.08
Audio (RF):LSTM 256/2 | Lip.(PF): WLAS RF + LSTM 256/2 | FC 512/3	75.23	75.96
Lipreading RF networks		
Audio (PF):LSTM 256/2 | Lip.(RF): WLAS RF + LSTM 256/2 | FC 512/3	69.91	79.22
Audio (RF):LSTM 256/2 | Lip.(RF): WLAS RF + LSTM 256/2 | FC 512/3	76.19	81.10
Attention: Audio (RF):LSTM 256/2 | Lip.(RF): WLAS RF + LSTM 256/2 | FC 512/3	72.93	74.60

Table 5.1: Performance of multimodal networks for clean audio: comparison between normal training and end-to-end (E2E) training. There is a clear difference between networks that use raw features (RF) of the lipreading LSTM network, and those that pre-classify the lipreading outputs (PF). The RF networks show a lot more improvement after E2E training, and surpass the performance of the PF networks. The PF networks perform better without end-to-end training. The RF network with attention (discussed in Section 5.4.2) also improves after end-to-end training.

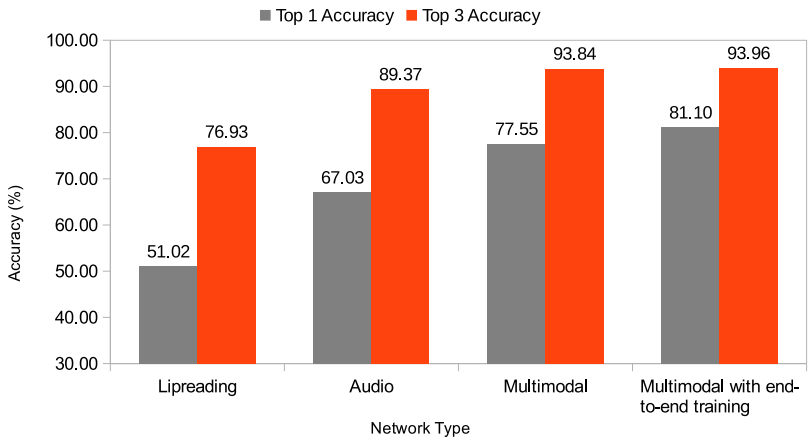
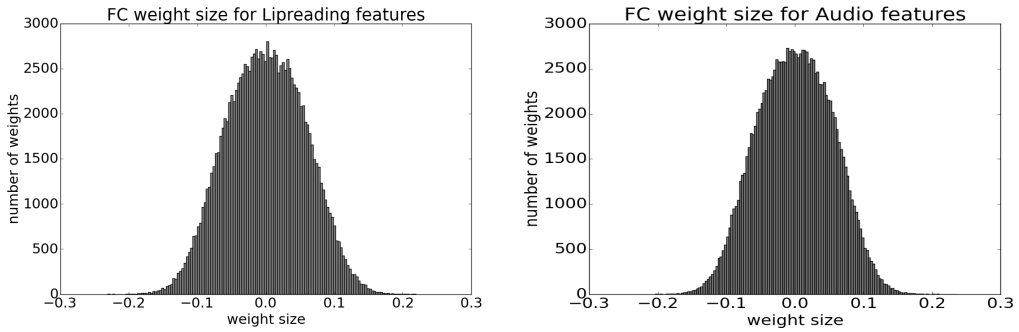


Figure 5.4: Comparison between the networks. Audio: 256 / 2 LSTM. Lipreading: WLAS (RF) + 256 / 2 LSTM.
Multimodal: Audio (RF) + Lipreading (PF) + FC 512 /3.
Multimodal with E2E training: Audio (RF) + Lipreading (RF) + FC 512 /3



(a) Weights for lipreading

(b) Weights for audio

Figure 5.5: Weights sizes for audio and lipreading subnetworks in the first layer of the FC classification network

	Lipreading	Audio
Mean	0.0470	0.0447
Median	0.0416	0.0398
Rms	0.0574	0.0545

Table 5.2: Comparison lipreading and audio weights in FC combination layer Mean and median are computed with the absolute values of the weights

5.3.4 Relative importance Lipreading and Audio

The output layer of the audio and lipreading subnetworks is for each an LSTM layer of 256 LSTM units. This makes sure that there are an equal number of features for both subnetworks so that the size of the weights in the classification layer can be compared. The weights in the first Fully Connected layer that combines the audio and lipreading features can then give an indication of the relative importance attached to the two subnetworks.

The weights distribution of these FC weights for lipreading is shown in Figure 5.5a and the distribution for audio is shown in Figure 5.5b. A comparison of the mean, average and root-mean-square of the weights for each subnetwork is shown in tTable 5.2. The weights for the lipreading features are slightly higher, which is surprising as lipreading is a harder task and top 1 and top 3 performances were lower on the lipreading network than for the audio network. Therefore audio was expected to have a higher relative importance. Possibly the features produced by the lipreading network are larger in size than those for the audio network, but this was not analyzed further.

The figures 5.5a and 5.5b also show that many of the weights are close to zero, indicating some possibilities for optimization through pruning.

5.4 Impact of Noise

5.4.1 When trained with clean audio

When noise is added to the audio data, the performance of the audio network decreases sharply. On the other hand, performance of the lipreading network is not affected. It is expected that the multimodal network will not be as heavily influenced as the audio network because of the lipreading network.

Performance figures under different noise conditions are compared for the audio, lipreading and multimodal networks in Figure 5.6 (white noise) and Figure 5.7 (other speakers interference). Performance with white noise is significantly worse than with noise from other speakers, just like for the audio-only networks in Chapter 3. This shows that the multimodal network still depends quite strongly on the performance of the audio subnetwork.

Consider Figure 5.6. Going from clean audio to an SNR of 0dB, performance for the audio-only network drops by an absolute percentage of 34.76%, while the drop for the multimodal network is only 24.16%. The multimodal network performs significantly better than the audio-only network not just with clean audio, but also under noisy conditions. As audio noise increases, the multimodal network starts performing worse. At the highest noise levels it performs worse than the lipreading network separately.

An explanation could be that the multimodal network relies more on the audio than on the lipreading (since audio-only recognition performance is better than lipreading-only performance). See Figure 5.5 and Table 5.2 for a comparison of the relative weights in the FC network that combines the audio and lipreading features. The lipreading and audio networks seem to attributed similar weights, which is surprising considering that the audio network achieves higher classification scores.

Because the network has not trained on noisy audio, it cannot deal with it. However, if the network were trained on low-quality audio the combination network would attribute more weight to the lipreading predictions, and it would not make full use of the audio network when audio is of high quality. Because the weights of the classification network are fixed after training, the network cannot adapt to changing noise circumstances.

The end-to-end trained networks perform significantly worse when evaluated on noisy audio (Figure 5.8). This is probably because the subnetworks an no longer be trusted to function well separately. The audio network performs badly on noisy audio, and the lipreading network will suffer degradation as well. When not trained end-to-end, the lipreading network's performance doesn't decrease and can compensate partially for the audio network.

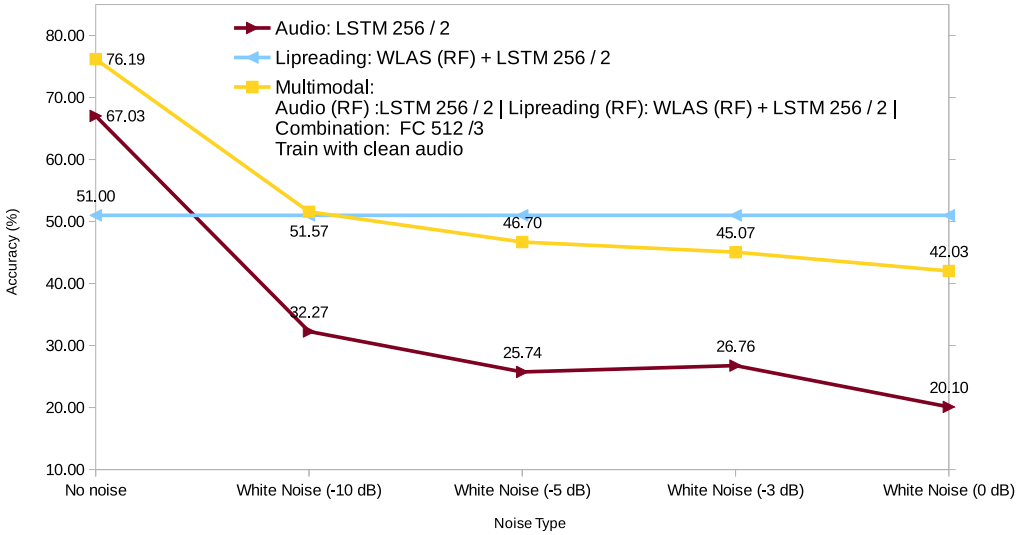


Figure 5.6: Audio is degraded with white noise. Performance of the multimodal network is better than for the audio network at all noise levels, but lower than the lipreading network at very high noise levels.

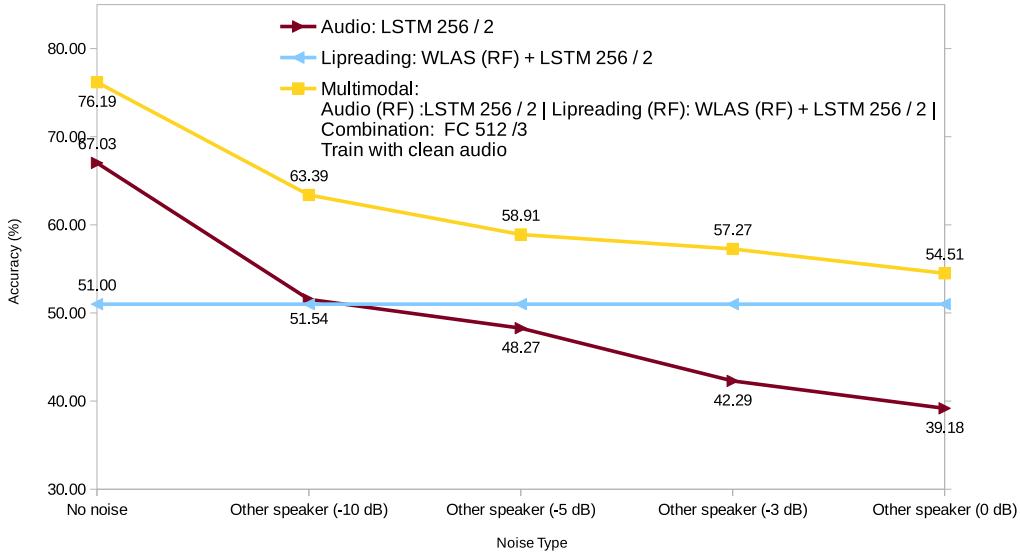


Figure 5.7: Audio is degraded with interference from other speakers. Performance of the multimodal network is better than the audio network and than the lipreading network, also at very high noise levels.

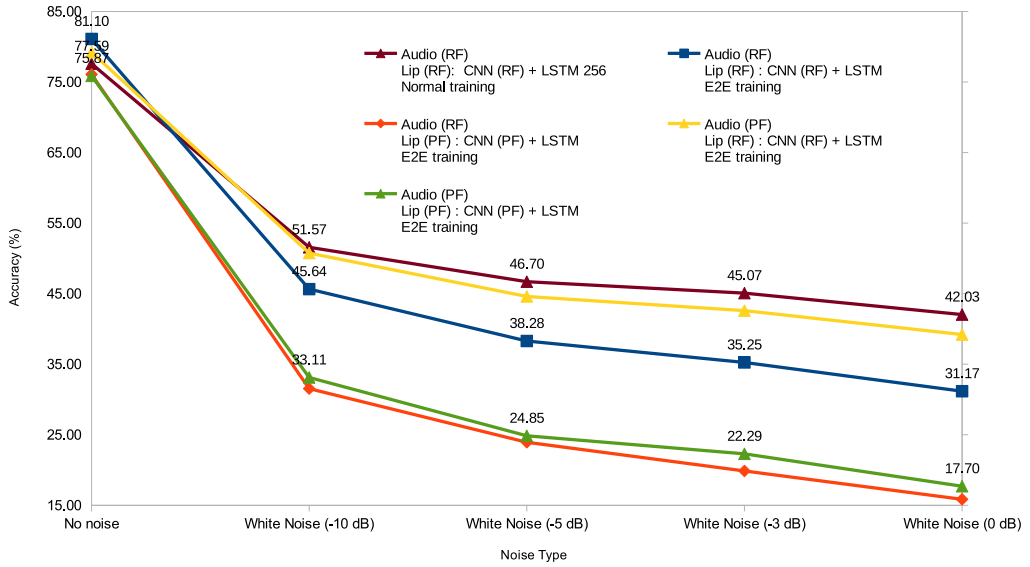


Figure 5.8: Comparison with noisy audio (white noise added) between multimodal networks trained end-to-end and networks with fixed audio and lipreading subnetworks.

5.4.2 Adapting to noisy audio: Attention networks

As seen in Section 5.4.1, performance is degraded quite significantly when noise is added. The network performs worse than the lipreading subnetwork. The reason for that is that is not able learned to take change the relative weights attributed to the lipreading and audio networks. Intuitively, it makes sense to make more use of the lipreading network when the audio quality is low (since then the lipreading works much better than the audio recognition). Similarly, the audio network should have a higher relative weight than the lipreading network when audio quality is high (since then it performs much better than the lipreading network).

As the weights in a normal feedforward FC network are fixed, it is not possible to do this when this type of network is used to combine lipreading and audio recognition.

A solution is to add an attention network (AN) that decides the relative importance of the visual-only and audio-only outputs. Attention networks are fully connected networks somewhat separate from the normal network structure (see Figure 5.9. The audio-only output features AF and lipreading-only output features LF are provided as inputs to the AN. The number of audio and lipreading features should be of the same size and type for the attention network to perform optimally (either preclassified phoneme predictions or LSTM output features), as then their 'meaning' is equivalent. The AN's job gets much harder if one network produces phoneme predictions and the other produces raw LSTM output features, as there is not a linear relationship between the two.

For each of the feature pairs AF_i, LF_i (one feature from audio and one from

lipreading) the AN will produce pairs of weights L_i, A_i, where the sum of each output pair equal to one. Each pair is multiplied pointwise with the AF_i, LF_i feature pairs to produce an output feature which is a weighted sum of the audio and lipreading features.

$$OF_i = L_i * LF_i + A_i * AF_i \tag{5.1}$$

The AN will learn during training that when the audio network isn't sure of its prediction (see Figure 2.13a), it is better to increase the weight of the lipreading network and vice versa. This leads to better performance as more weight is given to audio when audio quality is high, and more weight to lipreading when audio quality is low. See Figure 5.9 for an illustration [62].

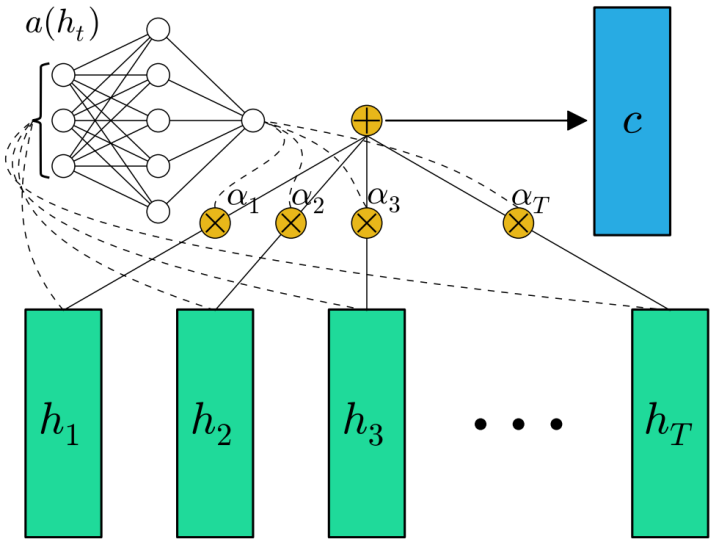


Figure 5.9: An attention network (AN). The output features in c are a weighted combination of the input features h_i, using the weights α_i produced by the AN. $a(h_t)$ is the mapping from input features to relative weights, and is learned by the AN through training [62].

An intuitive explanation of its inner workings can be helpful here. Let's assume that both the lipreading and audio networks produce classified phonemes as outputs. When audio quality is high, the audio network will produce one feature with a much higher value than the others; . The AN will notice this, and the relative weights for the audio network will be high. The lipreading network will also provide predictions, but the values will be more 'spread out' as it will not be as accurate as the audio network (see Chapter 4). The AN will notice this, and the weights for the lipreading network features will be lower than those for the audio network features.

A multimodal network with such an attention network can be trained just like the other networks described. In order for it the attention network to learn, it needs to train on noisy audio data as well as on clean data (see Section C.2.4 for the implementation details).

When the multimodal network is trained on both clean and noisy data, the performance curve is very flat and performance for noisy audio is improved to close to 50%. This indicates that the attention network does work, but attributes too much weight to the lipreading network. This issue can be resolved by first training the network on clean audio, and then retraining it with a low learning rate with noisy data. During the retraining step, the learning rate is divided by 10 before the first epoch, and then continues as described in Section 5.3.1.

Figure 5.10 shows that after this improvement, performance of the multimodal network (without pre-classification) is better than for lipreading-only network for all noise levels tested. The performance at low audio SNR is almost 10 percentage points better than without the attention network. The performance for clean audio is 72.93%, compared with 76.19% for the same network without an attention mechanism. It is also significantly higher than for audio-only networks, especially at low audio SNR (51.42% compared to 20.10%).

When the same network with attention mechanism is retrained end-to-end, it performs better: 74.60% for clean audio and 58.55% at 0dB audio SNR.

These results show that adding an attention mechanism can make a network much more robust to variations of audio quality without sacrificing much performance.

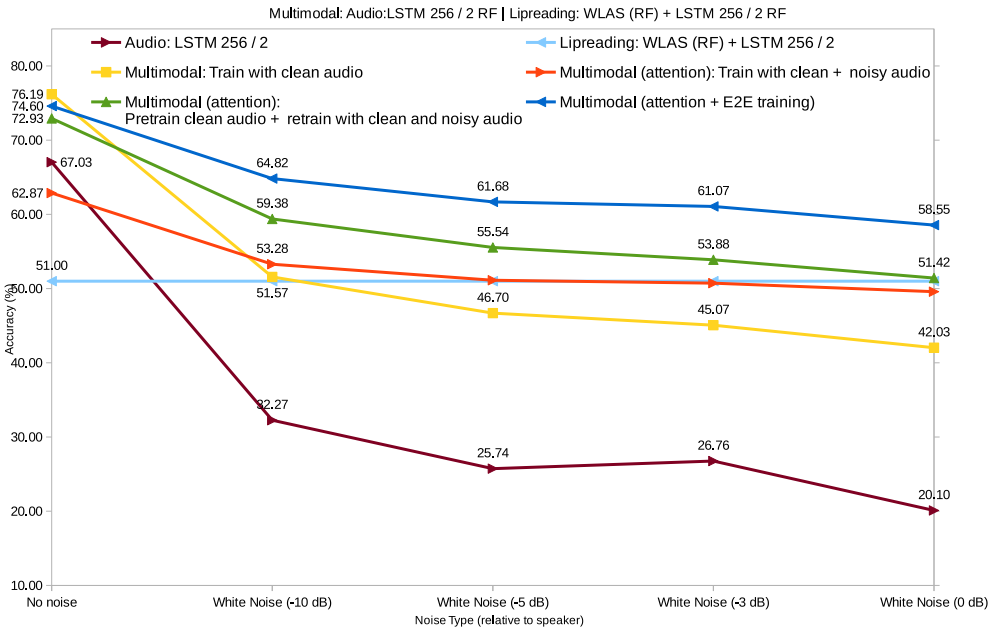


Figure 5.10: Performance for audio with white noise.
Comparison between audio-only, lipreading-only and multimodal networks. The multimodal networks' architecture is 'Audio (RF):LSTM 256 / 2 | Lipreading (RF): WLAS (RF) + LSTM 256 / 2'. They are evaluated without attention, with attention and with attention plus end-to-end training.

5.4.3 Speaker Dependence

Performance of the best performing network trained on the professional lipspeakers is evaluated on the test set of TCDTIMIT (containing only volunteers). This is compared to the performance of networks trained only on the volunteers. Figure 5.11 shows that performance is much lower than on the lipspeaker dataset. This is probably because the lipspeaker dataset is smaller (containing only 3 speakers) and therefore it's hard for the networks trained only on lipspeakers to generalize to the volunteers.

The second reason is speaker dependence. The lipspeaker networks are trained and tested on videos from the same speakers. As shown in Figure 5.11, validation and test performance are very close for the lipspeaker network. Therefore the lipspeaker networks aren't overfitting.

In contrast, the volunteer test set contains videos from speakers of which there aren't any videos in the train set. In other words: the volunteer test set also measures speaker independence of the trained networks. Figure 5.11 shows that there is a large gap between validation and test performance for volunteer networks. The lipspeaker networks did not overfit and the volunteer networks were trained in the same way on more data, so should not be overfitting either. The remaining explanation is that the volunteer networks are speaker dependent and don't perform very well on the speaker-independent volunteer test set.

Speaker independence is generally hard to achieve for lipreading, as also shown in Chapter 4, so the lipreading part of the network will be reducing the performance.

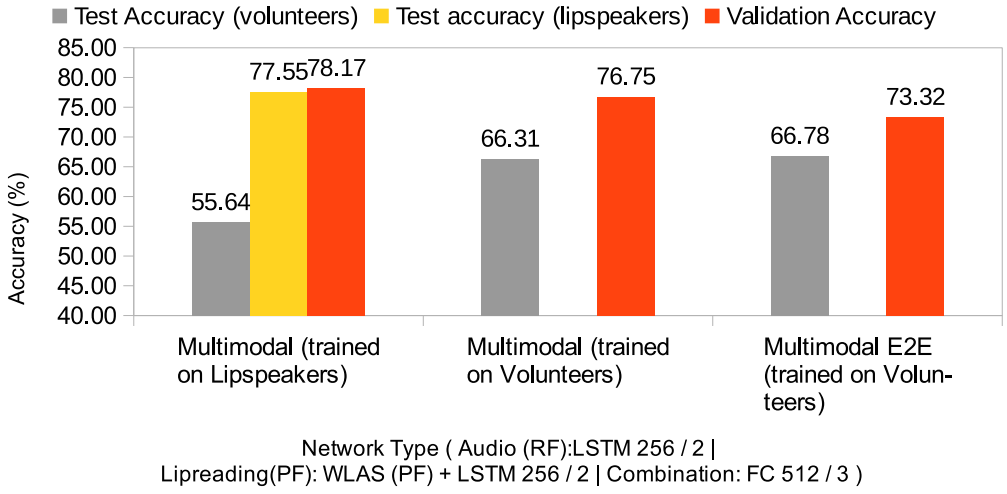


Figure 5.11: Performance for volunteers of TCDTIMIT.
The first network is trained on the lipspeakers, and evaluated on the volunteers. Its performance on the lipspeaker test set is shown as well. The other two networks are trained and evaluated on the volunteers.

5.5 Computational considerations

The computational complexity of the combination network is fairly low: the number of input features is limited to the sum of the number of output features from the audio and lipreading subnetworks. These correspond to the sizes of the last LSTM layers of those subnetworks, which are generally 256 or lower. For networks using pre-classification this is even reduced to 39 features.

The audio network was chosen for a good balance between complexity and performance (Chapter 3) and is the same for all trained multimodal networks. The CNN was also chosen for a good balance (Chapter 4) and is equal for all of the networks. CNNs require lots of computations. For CNN-based classifiers, over 90% of runtime is spent in the convolutional layers [60].

Therefore most of the variation in complexity comes from the type of connections between the subnetwork. The best performing network uses raw features in all parts of the networks (lipreading CNN output, audio output, lipreading output). It performs well, but requires lots of weights for all of the connections.

Table 5.3 summarizes the number of parameters in each part of the network, and the achieved accuracy. Weights in the CNN are reused many times per image, and there's one image for each timeframe. Audio LSTM weights are used once per audio frame, of which there are more than timeframes. Lipreading LSTM weights and weights in the FC combination layers are used only once per timeframe.

If resources are limited and performance under noisy conditions is not a concern, the audio-only network is the best option. It does not use a CNN which saves a lot of computation (see Section 4.5), and still performs fairly well. If the issues with TCDTIMIT labeling (see 3.6) can be resolved, similar performance to TIMIT state-of-the art might be possible (around 82% correctness) with only audio.

Multimodal networks can improve performance, especially under noisy audio conditions. Two main types of multimodal networks achieved high scores.

The first one used pre-classification after the CNN and the lipreading LSTM (but not after the audio LSTM). This means that the number of weights required to connect between the stages of the multimodal network is limited as only the phoneme predictions are used as features. Performance is quite high at 77.55%.

The second type does not use any pre-classification. It achieves 76.19% for clean audio. If trained end-to-end, it achieves the highest score of all trained networks for clean audio (81.10%), but does not perform well for noisy audio (31.17%). It requires a lot more connection weights than the network with pre-classification. If trained with an attention network, it does perform very well under noisy conditions (58.55%).

5.6 Comparison with TCDTIMIT baseline

The baseline performance under influence of audio noise from [1] of audio-only and audio-visual baseline networks trained and evaluated on the lipspeakers are shown in Figure 5.12. The correctness scores can be compared to the scores in Figure 5.10.

Network Type	CNN	CNN FC	Lip LSTM	FC	Audio	Acc. (clean)	Acc. (0dB SNR)
Lip: WLAS	6,197,760	978470	0	0	0	39.90	39.9
Lip: WLAS (RF) + 256 / 2 LSTM	6,197,760	0	27,017,767	10,023	0	51.02	51.02
Lip: WLAS (PF) + 256 / 2 LSTM	6,197,760	978470	1,367,591	0	10,023	48.65	48.65
Audio: 256 /2 LSTM	0	0	0	10,023	1,357,568	67.03	20.10
Multimodal: Audio (RF) + Lip (PF): WLAS (PF) + 256 / 2 LSTM	6,197,760	978470	1,367,591	686,848	1,357,568	77.55	27.73
Multimodal: Audio (RF) + Lip (RF): WLAS (PF) + 256 / 2 LSTM	6,197,760	0	27,017,767	797,952	1,357,568	76.19	42.03
Multimodal E2E: Audio (RF) + Lip (RF): WLAS (RF) + 256 / 2 LSTM	6,197,760	0	27,017,767	797,952	1,357,568	81.10	31.17
Multimodal (attention + E2E): Audio (RF) + Lip (RF): WLAS (RF) + 256 / 2 LSTM	6,197,760	0	27,017,767	1M	1,357,568	75.70	58.55

Table 5.3: Comparison of phoneme classification performance and computational complexity (number of weights) between different networks. The number of weights in the network is a good indication of the number of MAC operations for FC and LSTM layers (for the CNN, see Section 4.5), and required memory to store the network. The number of weights in each part of the network is shown, as well as the network performance for clean and noisy audio.

The baseline audio network was trained on lipspeaker data only, while the neural networks were trained on the whole TCDTIMIT train set. Audio performance was compared in Chapter 3 and lipreading performance in Chapter 4.

For clean audio, baseline performance on the lipspeakers is lower than for the audio-only network (59% compared to 73%). The baseline audio network was trained on lipspeaker data only, and achieved around 73% (compared to 65.47% for the baseline network trained on the whole dataset and evaluated on the official TCDTIMIT test set (which contains only non-lipspeakers)). The audio networks in this chapter were all trained on the TCDTIMIT non-lipspeaker data. The performance of these networks on the lipspeakers is similar to the performance on the non-lipspeaker test set (around 67%).

The baseline multimodal network was a single HMM trained on the concatenation of MFCC features and visual DCT features as inputs. It achieved around 59% correctness for clean audio and 44% correctness for audio at an SNR of 0dB.

In contrast, the neural networks trained in this work consist of separate sub-networks for acoustic and visual processing of which the output features are then combined and classified. The performance of these networks for clean audio is several percentage points better than performance of the audio-only network separately (74.60% compared to 67.03%, see Figure 5.4). If the multimodal network is trained end-to-end (weights of the subnetworks are trained as well as the combination layers) without attention mechanism and only on clean audio data, even higher performances are possible (81.10%, see Table 5.1), but then performance for noisy audio goes down to 31.17%.

These results suggests that it is much better to use specialized networks to extract features from each type of data (audio or visual), and then combine them instead of classifying everything at once.

The performance of the baseline multimodal network goes down to about 44% at 0dB audio SNR. The multimodal network without attention from this work achieved 42.03% correctness. The network with an attention mechanism, trained end-to-end achieved performance of 58.55%, an improvement of over 14% compared to the baseline and more than 16% compared to the network without attention.

Table 5.4 summarizes the performance scores.

Network Type	Clean audio	Noisy audio (SNR 0dB)
TCDTIMIT baseline	59	44
Multimodal (RF)	76.19	42.03
Multimodal (RF) with E2E training	81.10	31.17
Multimodal (RF) with attention + E2E	75.70	58.55

Table 5.4: Lipspeaker correctness scores for clean and noisy audio for different networks, compared with the baseline performances. The multimodal networks shown do not use any pre-classification.

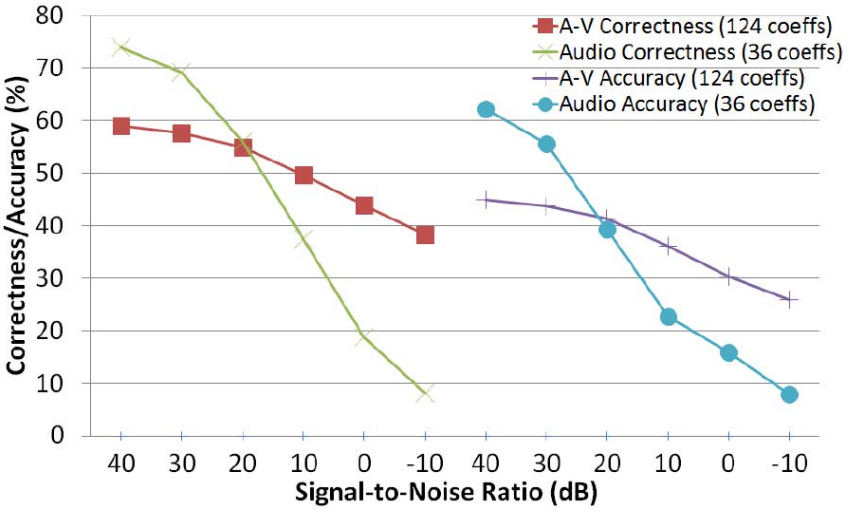


Figure 5.12: Baseline TCDTIMIT phoneme classification (audio-only and audio-visual) performance for lipspeakers. Audio correctness is the performance of network trained on lipspeaker data only. The baseline multimodal network performed worse than the baseline audio-only network.

5.7 Conclusion

Different network architectures were evaluated, using several types of audio and lipreading subnetworks. These were combined either by using the raw output features of these subnetworks, or after a pre-classification of audio and lipreading separately. Different combination networks were analyzed as well. Networks trained end-to-end were compared with networks where the weights of the subnets were fixed, and shown to perform better in many cases (Section 5.3.2).

The performance of the multimodal network on TCDTIMIT was evaluated and shown to be significantly higher than for either lipreading-only or audio-only networks (Section 5.3.3).

The performance of the multimodal, lipreading-only and audio-only networks on phoneme classification for noise-degraded audio was analyzed. The multimodal significantly outperformed the audio network, but failed to perform better than the lipreading-only network at very high noise levels (Section 5.4.1).

By adding an attention network the multimodal network could be trained to adapt itself to variations in audio quality, and performance at all noise levels was significantly higher than either lipreading-only or audio-only networks (Section 5.4.2).

The trained networks performed very well on the lipspeaker subset of TCDTIMIT, but not as well on the volunteers set. This is likely due to the speaker dependence of the networks. A similar observation was made by the TCDTIMIT authors [1].

The trained networks outperform baseline results on this dataset by a large margin for both clean and noisy audio (Section 5.6).

Chapter 6

Discussion

6.1 Conclusion

Chapter 2 presented an introduction to classification using several types of neural networks

Chapter 3 discussed audio recognition with LSTM-based neural networks. The audio networks that were trained and evaluated on TIMIT closely approached state-of-the-art performance. They improve on the baseline performances of TCDTIMIT by several percentage points, achieving 67.03% compared to 65.47% correctness for the baseline (see Chapter 3). The large performance gap between TCDTIMIT and TIMIT indicates an issue with the TCDTIMIT labels.

Chapter 4 discussed lipreading with CNN-based neural networks. The lipreading baseline results were 57.85% correctness. Several CNN networks were compared with the trade-off between complexity and performance in mind. The WLAS network developed by Chung et al. [56] was deemed to offer the best trade-off. CNN-only networks achieves 54.42%, and the CNN-LSTM networks achieved 68.46%, an improvement of over 10 percentage points over the baseline. When applied to phoneme classification, these scores naturally go down due to the many-to-one mapping of visemes to phonemes. Phoneme classification scores are 39.90% for the CNN-only (WLAS) network, and 51.02% for the CNN-LSTM network.

Chapter 5 discussed multimodal speech recognition by combining lipreading and audio. Several techniques for combining the subnetworks were analyzed, for example the impact of adding pre-classification FC layers after the subnetworks. The impact of training the networks end-to-end was discussed. Noise performance of the multimodal network was shown to be significantly better than audio-only networks. Adding attention mechanisms allow the network to perform well under both noisy and noiseless scenarios. When the audio SNR was 0dB, it achieves 58.55% correctness, compared to just 20.10% for the audio-only network. It achieves 75.70% correctness for clean audio. This is a significant improvement over the baseline multimodal results on TCDTIMIT, which were 44% for noisy audio and 59% for clean audio.

6.2 Future work

For **Audio** networks: In order to bring performance on TCDTIMIT to similar levels as achieved on TIMIT, the labeling issue should be resolved. If the issue is due to misalignments, using CTC for automatic selection of valid predictions could help a lot because it does not make use of the timestamps in the labels. Only which phonemes are spoken and their order is relevant.

Achieving higher performance and better speaker-independence can be achieved if the networks are trained on larger and more varied datasets. This was attempted by combining TIMIT and TCDTIMIT, but did not work well (likely also due to differences and inconsistencies in TCDTIMIT labeling compared to TIMIT labeling). This might also be resolved if CTC is used. Other datasets could be used as well.

To make the speech recognition network more useful in practice is would be interesting to add a language model on top of the phoneme prediction network. Another option is to train the network on word-level predictions from the start, although this does make it harder to combine with lipreading networks.

As binary networks have shown some promise for CNNs, the same techniques could be used to binarize LSTM network so that those networks can also be evaluated more efficiently.

For **Lipreading** networks: Only a few CNN types were explored and it could well be that other networks or variations on these networks perform better. Networks with reduced-precision weights, especially binary networks, show a lot of promise even though their performance impact varies depending on the network architecture. Determining what factors exactly influence the performance could limit the accuracy loss.

Attention networks are being researched for image recognition as a way to save computation and improve performance. It would be interesting to add a visual attention mechanism for lipreading [63].

For **Multimodal** networks: When the audio subnetwork is improved, multimodal performance will likely improve as well. In addition, the attention mechanisms have shown a lot of promise. Further research in that area can further improve performance.

For resource-constrained systems, the audio-only subnetwork could be used as long as the network is 'sure' of its predictions (see Figure 2.13a). If it is not, the lipreading network can be enabled for multimodal recognition.

Appendices

Appendix A

Nederlandstalige Samenvatting

Goede communicatie tussen individuen van een diersoort is evolutionair bijzonder interessant. Het laat toe samen te werken aan een gemeenschappelijk project, ideeën uit te wisselen en relaties te vormen met anderen. De menselijke spraak is vrij uitzonderlijk in de natuur en heeft een grote rol gehad in de menselijke dominantie van onze planeet.

Spraakherkenning door computers is een technologie waarvan al gedroomd werd sinds de jaren '50. Ze zou de interactie met computers bijzonder veel vergemakkelijken. Over de jaren heen werden steeds betere systemen ontwikkeld, maar ze bleven slechts inzetbaar in beperkte domeinen en vereisten veel manuele tuning om goed te werken. Neurale netwerken vormden een aantrekkelijk alternatief, maar vereisten te veel rekenkracht en geheugen .

Dankzij de Wet van Moore en enkele theoretische innovaties (bijvoorbeeld de introductie van de LSTM unit en het concept van 'deep learning' (zie Hoofdstuk 2) werden neurale netwerken steeds aantrekkelijker. Dankzij het internet en de komst van 'big data' werden grote hoeveelheden informatie eenvoudig toegankelijk, en konden grote neurale netwerken goed worden getraind. Dit resulteerde in innovaties in beeldverwerking (diepe convolutionele netwerken), spraakherkenning (diepe bidirectionele recurrent netwerken), en AI (bijvoorbeeld IBM Watson of AlphaGo van Google Deepmind).

Spraakherkenning als domein heeft zich bijna altijd gefocust op geluidsverwerking omdat dat de belangrijkste manier van communiceren is en natuurlijk aanvoelt. De meest elementaire klanken die een mens kan produceren zijn gecatalogiseerd en heten 'fonemen'. Spraakverwerking op basis van geluid heeft als nadeel dat ze niet goed presteert wanneer de audiokwaliteit laag is. Als we kijken naar hoe mensen met gehoorproblemen toch anderen kunnen verstaan, is duidelijk dat zij ook andere informatiebronnen dan geluid aanspreken, met name het liplezen. Dit leidt tot de vraag of een computer ook zou kunnen leren liplezen. Als dit gecombineerd wordt met geluidsverwerking in één systeem ('sensor fusie', het systeem heet dan 'multimodaal'), zouden de spraakherkenningssystemen zeker bij lage geluidskwaliteit veel beter kunnen werken. De grote vooruitgang in performantie van beeldverwerkingssystemen die dankzij convolutionele neurale netwerken is behaald, biedt hier een mogelijkheid.

Aangezien herkenning van fonemen de essentie is van ieder spraakherkenningssysteem, wordt hierop gefocust in dit werk. Hoofdstuk 2 bespreekt wat neurale netwerken precies zijn, welke types best geschikt zijn voor welke problemen en hoe ze te trainen. In de volgende hoofdstukken wordt onderzocht welke factoren de performantie van zowel audioherkenningssystemen (Hoofdstuk 3) als lipleessystemen (Hoofdstuk 4) beïnvloedt, en wat de mogelijke performantiewinsten zijn als de twee types netwerken worden gecombineerd (Hoofdstuk 5).

De dataset die gebruikt werd om deze netwerken te trainen is TCDTIMIT [1]. Voor audio netwerken wordt ook gebruik gemaakt van TIMIT, omdat die dataset een de facto standaard is voor spraakherkenning via audio [64]. Scores die vermeld worden zijn de 'top-1 accuracy'. Voor ieder element in de dataset produceert het netwerk een kansverdeling over alle fonemen. Het foneem met de hoogste waarschijnlijkheid wordt vergeleken met het echt uitgesproken foneem. Als de twee overeenkomen is de voorspelling 'correct', anders niet. De top-1 accuracy is dan de verhouding van het aantal correcte voorspellingen en het totaal aantal elementen in de dataset.

A.1 Audioverwerking

State-of-the-art resultaten voor spraakherkenning worden behaald door diepe, bidirectionele recurrent netwerken met LSTM eenheden in plaats van standaard neuronen, vaak afgekort als 'DBLSTM netwerken' (zie Sectie 2.4) [10]. Recurrent netwerken zijn neurale netwerken met verbindingen binnen een laag, of naar vorige lagen. Bij normale 'feedforward' netwerken zijn er enkel verbindingen naar neuronen in volgende lagen. De terugwerkende verbindingen creën een interne toestand die het netwerk toelaat informatie over de tijd heen op te slaan. LSTM eenheden zijn aangepaste neuronen die beter in staat zijn langetermijn-relaties in de datasequentie te ontdekken (zie 2.4).

DBLSTMs zijn opgebouwd als een hiërarchische structuur van verschillende lagen neuronen die een sequentie van data verwerken. Het bidirectionele aspect slaat op het feit dat deze netwerken de sequentie zowel van voor naar achter als van achter naar voor verwerken (zie Figuur 2.7). Zo kan rekening worden gehouden met informatie uit het verleden maar ook met informatie uit de 'toekomst'. Bijvoorbeeld bij het vertalen is dit erg nuttig: in het Duits komt het werkwoord meestal op het laatst. Je weet in het Duits dus pas wat de zin precies betekent nadat je de gehele zin gehoord hebt. Een bidirectioneel netwerk kan de voorspellingen voor het begin van de zin aanpassen afhankelijk van wat er op het einde komt (en omgekeerd). Ook bij andere toepassingen biedt deze techniek voordelen, zoals bij spraakverwerking.

Voor spraakverwerking wordt de ruwe data (.wav bestanden) eerst omgezet naar MFCCs (Mel-Frequency Cepstral Coefficients). Dit wordt gedaan om de data geschikt te maken voor verwerking. De Mel-Frequency schaal is een niet-lineaire schaal die rekening houdt met het frequentie-gevoelige menselijke gehoor (en dus spraak) waardoor verschillende fonemen beter van elkaar onderscheiden kunnen worden in het frequentiedomein.

De grootte van het audioverwerkingsnetwerk (het aantal LSTM eenheden in

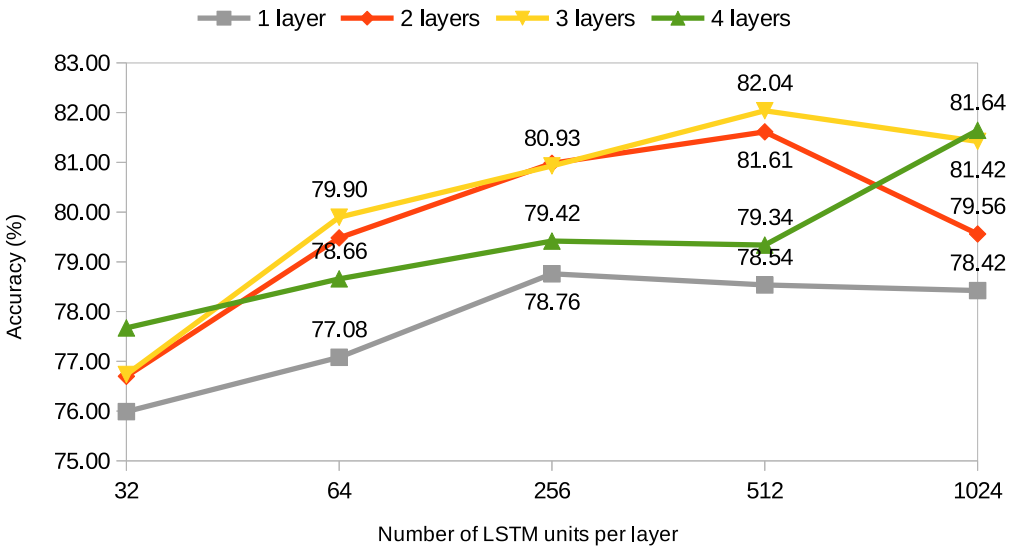


Figure A.1: Performantie voor verschillende audionetwerken. Meer LSTM neuronen per laag vergroot de performantie, maar vanaf 256 neuronen vlakt de performantiecurve af. Het aantal lagen heeft ook een impact, maar die is beperkt voor meer dan 2 lagen. De lage performantie voor de 4-lagige netwerken is mogelijk te wijten aan onvolledige training. Het netwerk met 2 lagen en 256 neuronen per laag toont een goede balans tussen performantie en complexiteit.

een laag) is sterk bepalend voor de performantie (zie Figuur 3.8 en Tabel 3.2). Grotere netwerken presteren beter, maar de winst vlakt af vanaf ongeveer 256 LSTM eenheden. Het aantal parameters in het netwerk stijgt ook exponentieel met het aantal eenheden per laag (recurrent netwerken zijn volledig verbonden), dus het is wenselijk dat aantal beperkt te houden.

Verschillende lagen kunnen opeengestapeld worden om het netwerk toe te laten hiërarchische structuren te ontdekken in de datasequentie (dit heet 'deep learning') [25]. Enkele netwerken zijn getraind en hun performantie is weergegeven in Figuur A.1. Er is een groot verschil tussen de performantie van een één-lagig netwerk en een twee-lagig netwerk, maar daarboven is de winst beperkt.

Het aantal gewichten in een netwerk is een goede maat voor de rekencomplexiteit. Tabel 3.3 toont overzicht hiervan, en Figuur 3.10 toont een grafische weergave van de afweging performantie/netwerkgrootte.

Het netwerk met 2 lagen van 256 LSTM units per laag behaalt goede performantie zonder al te veel gewichten te hebben. Het wordt daarom ook gebruikt in de multimodale netwerken.

Wanneer witte ruis toegevoegd wordt aan de audio, behalen de netwerken nog slechts classificatie scores van iets boven de 20% bij een SNR van 0dB (tov 70-80% voor audio zonder ruis).

De hoogt behaalde score op TIMIT is 82.04%, vergelijkbaar met het state-of-the-

art resultaat van 82.35% [10] (zie Sectie

A.2 Liplezen

De netwerken voor liplezen maken gebruik van convolutionele neurale netwerken (CNNs). Drie types CNN zijn getest: een netwerk ontwikkeld door onderzoekers van de universiteit van Oxford en Deepmind [56] (verder 'WLAS' netwerk genoemd, naar hun paper), een netwerk ontwikkeld voor de CIFAR10 benchmark in [51] (verder VGG netwerk genoemd), en een netwerk met 'residual blocks', met 50 lagen [7] (verder 'ResNet50' genaamd).

Systemen voor liplezen gebruiken normaal geen fonemen voor classificatie. Analoog aan de fonemen, kunnen we 12 verschillende klassen van visueel verschillende standen van de mond onderscheiden, genaamd 'visemen'. De mens kan verschillende fonemen produceren zonder de stand van de mond te veranderen. Verschillende fonemen horen dus bij eenzelfde viseem (zie Tabel 4.2).

Het WLAS CNN is iets groter dan het VGG netwerk maar veel kleiner dan het ResNet50 CNN (kleiner betekent dat het netwerk minder gewichten bevat). Het VGG CNN 'comprimeert' het beeld ook niet zoveel, waardoor na het CNN nog de netwerken erna nog zeer veel gewichten hebben (zie Tabel 4.1 en Tabel 4.2). Performantie van de drie ligt dicht bij elkaar (zie Figuur 4.10).

De CNNs op zich verwerken ieder beeld apart. Het zou veel beter zijn mochten ze gebruik maken van de informatie die vervat zit in de volgorde van beelden uit eenzelfde video. De fonemen van die beelden zijn immers niet onafhankelijk aangezien ze behoren tot dezelfde zin. Dit kan bereikt worden door een LSTM netwerk toe te voegen dat de outputs van het CNN voor ieder beeld in de sequentie verwerkt. Er zijn twee mogelijkheden om het CNN met het LSTM netwerk te verbinden: ofwel worden de CNN outputs eerst geclassificeerd om foneem voorspellingen te bekomen (zie Figuur 4.9), ofwel worden de CNN outputs rechtstreeks aan het LSTM doorgegeven (zie Figuur 4.8).

Een vergelijking tussen de twee types is gegeven in Figuur 4.13. De netwerken met classificatie na het CNN presteren minder goed dan die zonder classificatie, maar vereisen wel veel meer gewichten in het verbindingsnetwerk (zie Tabel 4.3).

De CNN-LSTM netwerken presteren vaak heel wat beter dan enkel de CNN netwerken, zoals getoond in Figuur A.2.

De meeste rekencomplexiteit zit in het convolutionele deel van het netwerk [60] (zie Sectie 4.5), dus als rekenkracht de beperking is, is het beter geen voorclassificatie te doen voor de beste prestaties. Als geheugen beperkt is is het grote aantal gewichten wellicht een probleem, en is het beter wel voorclassificatie te doen en wat performantie in te leveren.

Een andere techniek om de hardware vereisten te beperken is door de gewichten van het netwerk te gebruiken met verminderde precisie. Dit heeft zeer weinig impact op de prestaties tot een bepaald niveau [23]. Nog een derde manier is door 'binaire netwerken' te gebruiken, waarvan de gewichten enkel de waardes +1,0 of -1 kunnen aannemen [51]. Voor het WLAS CNN was de impact hiervan groot, maar

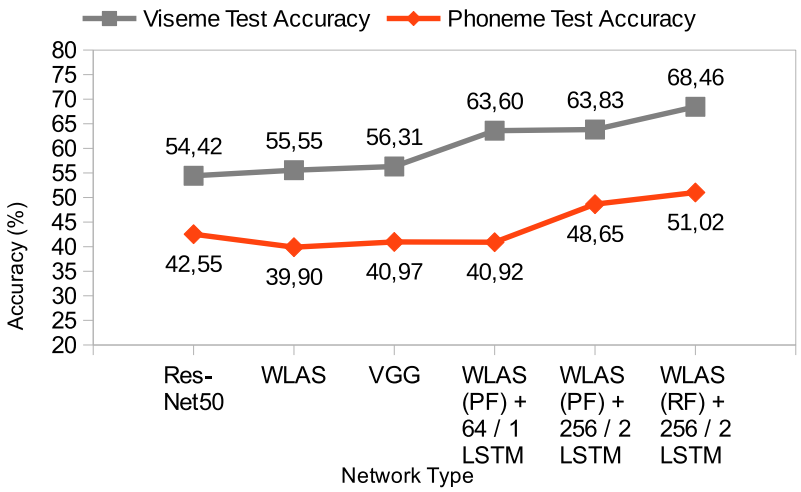


Figure A.2: Vergelijking scores voor foneem -en viseem classificatie

het gebinariseerde VGG-gebaseerde netwerk bereikte nog scores van 35.75% (in vergelijking met 40.97% in niet-gebinariseerde versie).

De netwerken behaalden goede resultaten. CNN netwerken behaalden tot 56.31% voor viseem classificatie, en CNN-LSTM netwerken zelfs 68.46%. Dit is een significante verbetering ten opzichte van de baseline resultaten op TCDTIMIT van 57.85%

A.3 Sensor fusie: Multimodale netwerken

Voor het combineren van audio en liplees-netwerken zijn er verschillende mogelijkheden, net als voor het combineren van CNN en LSTM. Een netwerk zonder classificatie na de subnetwerken voor audio en liplezen is afgebeeld in Figuur 5.3. Figuur 5.4 toont een vergelijking met de performantie van de subnetwerken met het multimodale netwerk.

Bij Multimodale netwerken is het mogelijk de deelnetwerken vast te houden en enkel de gewichten voor de combinatie te trainen, of de gewichten van het volledige netwerk te trainen. Als het volledige netwerk getraind wordt (end-to-end training) kan hogere performantie behaald worden, maar dit gaat ten koste van de prestaties bij slechte audio, waar het multimodale netwerk net voor bedoeld was. Dit is dus niet echt aan te raden.

Om het netwerk nog beter om te doen gaan met ruis, is het nodig om het relatieve gewicht van het audio netwerk en het liplees-netwerk dynamisch aan te passen naargelang de kwaliteit van de audio. Die kan indirect bepaald worden door te kijken hoe goed het audio netwerk presteert. Als het erg onzeker is (zie Figuur 2.13b), is de audio waarschijnlijk slecht en is het beter meer te vertrouwen op het

liplezen. Een manier om dit automatisch te doen is door gebruik te maken van 'attentienetwerken' (zie Figuur 5.9 en Sectie 5.4.2).

Netwerken met attentiemechanisme kunnen betere scores behalen wanneer de audio slecht is. Een deel van de performantie bij audio zonder ruis gaat wel verloren, mogelijk doordat het attentienetwerk niet perfect getraind is of te eenvoudig is. De performantie bij 0dB audio SNR is nu 51.42%, terwijl dat voor het audio netwerk alleen slechts 20.10% was. Bij audio zonder ruis scoort het multimodale netwerk 72.93%, wat nog steeds een verbetering is ten opzichte van the audio netwerk alleen (67.03%). Dit is echter minder goed dan het multimodale netwerk zonder attentiemechanisme (76.19%) en het multimodale netwerk dat end-to-end getraind is zonder attentienetwerk (81.1%). Wanneer dat netwerk met attentiemechanisme end-to-end hertraind word, gaat de performantie nog omhoog: 74.60% voor clean audio en 58.55% voor 0dB audio SNR (zie Figuur A.3).

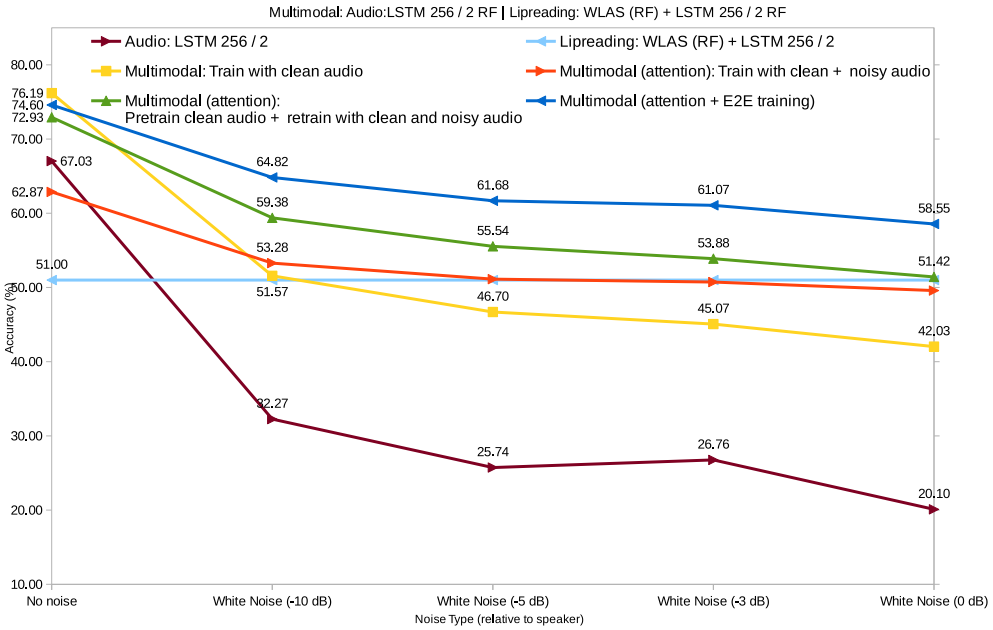


Figure A.3: Performantieverloop voor ruizige audio (witte ruis).
De multimodale netwerken hebben als architectuur 'Audio (RF):LSTM 256 / 2 | Lipreading (RF): WLAS (RF) + LSTM 256 / 2'. De resultaten worden getoond voor een standaard multimodaal netwerk, voor een netwerk met attentiemechanisme en voor een netwerk met attentiemechanisme dat end-to-end getraind is.

Deze resultaten zijn beter dan de baseline audio-visuele resultaten van TCDTIMIT (59% bij audio zonder ruis, en 44% bij een SNR van 0dB) (Figuur 5.12).

A.4 Conclusie

De bidirectionele meerlagige LSTM netwerken presteren beter dan de TCDTIMIT baseline resultaten (67.03% ten opzichte van 65.47%). Ze behalen ook zeer goede scores op TIMIT (82.04%, state-of-the-art 82.3% hoewel niet 100% direct vergelijkbaar). Het verschil in performantie tussen de twee datasets is waarschijnlijk te wijten aan fouten bij de labels van TCDTIMIT.

Convolutionele neurale netwerken presteren goed voor liplezen. Ze behalen tot 56.31% voor viseem classificatie. Het TCDTIMIT baseline resultaat is 57.85%. CNN-LSTM netwerken doen nog beter en bereiken tot 68.46%.

Multimodale netwerken kunnen zeer hoge performantie halen zowel voor audio zonder ruis als onder ruizige omstandigheden. Vooral met attentienetwerken presteren ze goed bij veel ruis (tot 58.55%). Bij goede audio combineren ze de extra informatie van het liplezen om betere scores te halen dan met audio alleen (74.60% ten opzichte van 67.03%), en bij slechte audio wordt vooral aandacht gegeven aan het liplezen waardoor de performantie behouden blijft. Netwerken die enkel gebruik maken van audio behalen dan slechts rond de 20%.

De baseline resultaten voor de TCDTIMIT lipsprekers waren 59% voor clean audio, en 44% voor een audio SNR van 0dB (witte ruis) [1].

De prijs die betaald wordt is dat het liplezen door de beeldverwerking veel rekenkracht kost, en dat de multimodale netwerken heel wat meer gewichten hebben dan netwerken die enkel audio gebruiken. Als de beschikbare hardware erg beperkt is en de performantie bij ruis niet van groot belang is, zijn audio netwerken mogelijk een betere keuze.

Appendix B

Software implementation

Explanations on how to use the software can be found on Github: multimodalSR. There are seperate folders for the audio, lipreading and multimodal networks, each with the required preprocessing scripts. The results are stored by default in the /home/user/TCDTIMIT folder.

In the folder for multimodal networks (combinedSR), the file *combinedNN.py* specifies which networks to train and/or evaluate. *combinedNN_tools.py* contains all the functions to construct the network architecture, compile Theano functions, load parameters and train the network.

It possible to only train the audio or lipreading subnetworks, or the whole network. You can choose whether to train/evaluate on the lipspeakers or volunteers. The networks to train are specified in the *networkToTrainList*. Each of these networks is an object of the *NetworkToTrain* class, which contains the hyperparameters of that network so the training functions can determine the proper architecture. The network will be automatically given a path and name corresponding to these hyperparameters. If a network has already been trained, it's stored weights are loaded before continuing training. The learning rate is then divided by 10 compared to a network which had not yet been trained. If a multimodal network is trained for the first time but its subnetworks for audio and lipreading already exist, those are loaded. It can be specified whether the network should be trained end-to-end or not.

The training data (eg validation and test cost and accuracy) are stored in a Python dictionary under $\langle networkName \rangle\_trainInfo.pkl$. The number of parameters for each part of the network is stored there as well. The network weights are stored in $\langle networkName \rangle.npz$ in numpy format.

In *combinedNN.py*, it can be specified to not train the networks, but just load the stored training data from the $\langle networkName \rangle\_trainInfo.pkl$ files. This data can be gathered from all networks and exported to an Excel file.

It is possible to run the networks for evaluation without retraining them. It can also be specified to use noisy or clean audio, and what type of noisy audio to use.

In addition to this explanation and the README on GitHub, the software is extensively commented to make it easier to use.

B.1 For CNN-LSTM networks

For the preprocessing of the TCDTIMIT data, see Section C.2.3.

When combining CNN and LSTM networks in one architecture, some issues arise. The Python library used, Lasagne, organizes CNN layer inputs in 2D, 1 dimension being the batch size and the other the input values (a 1D array) (see Section C.2.2). For lipreading using only CNNs all images from the database are put in one big array, as well as the labels. LSTM layers however, require a 3D input shape:

$$(batchSizeLSTM, nbTimeFrames, numFeatures) \tag{B.1}$$

to accommodate the extra timing information. The data arrays generated from the audio files in Section C.1.1 nicely fit this shape. The images do not as they are stored in a 2D array. To create an array structure that is usable by the LSTM networks and retain the time information for the images as well, the images have to stored in an array per video. This adds a dimension so the array is converted from 2D to 3D, corresponding to the shape of the audio data array. The labels are also reshaped in this way.

As explained in Section C.1.1, with some preprocessing, reshaping and the use of masks LSTM networks can be trained in batch just like CNNs (Section C.2.2). CNNs require the following shape:

$$(batchSizeCNN, nbFeatures) \tag{B.2}$$

When combining CNN and LSTM networks, there are some complications. The audio network produces 1D outputs: it generates predictions for a whole audio file at once. In contrast, the lipreading produces 0D outputs: it processes one image at a time. The lipreading network needs to predict all phonemes of a video in order to be able to combine the predictions from audio and lipreading networks.

A way to do this is to make use of batch processing: if each video is seen as a batch of several images, Lasagne can process all images in parallel. When the image files are stored as one array per video, this is straightforward. The CNN then produces an output of shape (batchSizeCNN = nbVideoTimeFrames, nbFeatures = nbCONVfeatures). As the LSTM expects a 3D input, this has to be reshaped to (nbParallel, nbTimeFramesVideo, nbCONVfeatures), where nbParallel is the number of videos processed in parallel, nbTimeFramesVideo is the number of time frames in each video. To provide a consistent shape, nbTimeFramesVideo has to be equal for all processed videos in a batch, which is why padding was applied in Section C.1.1.

If several videos are to be processed in parallel by a CNN-LSTM network, this gets more complicated. For audio only networks, the batchSizeLSTM dimension of Lasagne LSTM networks could simply be used. The batchSizeCNN dimension for the CNN has already been used to process one video at a time. It is possible to work around this by concatenating the arrays of different videos and then splitting them again after the CNN, as the number of valid frames in each video is known, but this gets quite complicated. In addition, by processing several images in parallel the

CNN gets duplicated in memory several times. If a number of videos are processed simultaneously, the required memory space is multiplied by that number and quickly grows unmanageable. For these reasons and the ones explained in Section C.2.3, just one video was processed at a time.

The output features generated by the audio network and the lipreading network can be concatenated quite simply with the Lasagne *ConcatLayer*. Before concatenation, some reshaping is needed.

After concatenation there will be several Fully Connected layers for classification. These Lasagne layers work similarily to the CNN layer in that they work with 2D data shapes:

$$(batchSizeFC, nbFeatures) \tag{B.3}$$

The same reasoning as for the CNN applies. Each video is seen as a batch of timeframes. Each timeframe corresponds to a certain number of features that are to be classified. The different timeframes of a video can be processed in batch, for an FC input shape of (batchSizeFC = nbVideoTimeFrames, nbFeatures = nbLIPfeatures + nbAUDIOfeatures).

The LIPfeatures can be the CONV output features of the CNN, or the output features of the last LSTM layer if a CNN-LSTM architecture is used for lipreading. The AUDIOfeatures are simply the output features of the last LSTM layer in the audio network. LSTM output features need to be reshaped from (1, nbTimeFrames, nbFeatures) to (nbTimeFrames, nbFeatures) so the FC layers can use them.

B.2 Multimodal networks

All of what is mentioned in Section B.1 is also valid here. In addition, there are some other issues. In Section C.1.1 a Theano function used the mask to extract only the outputs at valid timeframes from the audio network. When generating one large network of which the audio net is a subnet, the same must be done but within the network architecture, which is constructed of Lasagne layers. There is a Lasagne layer that can perform a similar function, the *SliceLayer*, but it doesn't support slicing in batch unless the masks for each video are the same.

This is of course not the case as different videos contain different spoken sentences, so the phonemes will not be equal. In addition to the extra complexity and the memory issues discussed in Section B.1, this is another reason why videos are processed one at a time instead of in batch.

Appendix C

Datasets

A good dataset should have a large amount of different speakers, each speaking a sufficient amount of phonetically rich sentences. The recording quality should be high and preferably the dataset should be labeled phoneme-wise. Two datasets that fulfill these requirements were used for this work: TIMIT for audio-only SR, and TCDTIMIT for audio-visual SR.

Both datasets suggest a train/test split [1] [49]. These splits are also used in this work.

C.1 TIMIT

TIMIT is a dataset created with funding from DARPA and contains audio recordings of 630 American English speakers, as well as phoneme and word level labels. It has long been a standard speech recognition dataset due to the variety and size of the dataset, as well as the quality of the labeling. Each speaker reads ten sentences, bringing the total size to 6300 sentences. The speech data are recorded as 16-bit, 16kHz waveform files. This dataset is used as a baseline to compare results of the trained networks for audio-only recognition [49]. The best-performing networks are then used to train on the audio of the TCDTIMIT dataset (see Chapter 3)

The TIMIT dataset contains a wide variety of speakers and is manually labeled, while the TCDTIMIT dataset has a limited number of speakers and is mostly automatically labeled. TIMIT contains 6300 spoken sentences, TCDTIMIT contains 6913 sentences.

The phonetic labels in TIMIT originally use a phoneme set of 61 phonemes. Lee and Hon [65] proposed to reduce the 61 phonemes to 39 since several of the original phonemes occurred only a few times or are very similar to each other, which makes classification hard. For this work the TIMIT 61 phonemes were converted to the reduced set. TCDTIMIT also uses this reduced phoneme set.

Table 2.2: Reduced Phoneme Set of Lee and Hon [49]

/Original Phoneme/	/Final Phoneme/
/ao/	/aa/
/ux/	/uw/
/axr/	/er/
/hv/	/hh/
/ix/	/ih/
/el/	/l/
/em/	/m/
/zh/	/sh/
/eng/	/ng/
/en/, /nx/	/n/
/ax/, /ax-h/	/ah/
/pcl/, /tcl/, /kcl/, /bcl/, /dcl/, /gcl/, /h#/, /#h/, /pau/, /epi/	/sil/
/q/	none

Figure C.1: Reducing 61 phonemes to 39 phonemes

C.1.1 Preprocessing of audio data

An important step to be able to train an audio recognizer is to divide the raw audio file in segments that can then be processed seperately. A very common technique to do this is by using Mel-Frequency Cepstral Coefficients [66].

MFCCs are the outputs of a cosine transformation of the logarithm of the short-term energy spectrum, expressed on a mel-frequency scale. This scale takes into account human perception of sound, in contrast to a linear frequency scale. They are a good way to express human speech information and are therefore used very often for speech recognition [67].

MFCC coefficients are derived as follows:

1. Take the Fourier transform of (a windowed excerpt of) a signal.
2. Map the results to the mel frequency scale with overlapping windows, and take log of the powers at the mel frequencies
3. Use these log powers as inputs for a discrete cosine transformation

Different MFCC variations are possible: the number of coefficients extracted, the overlap of the used time windows, whether derivatives of the MFCCs are considered and how many. A fairly standard approach is to use 12 MFCCs plus the normalized energy of the signal during each time window, and to add the first and second derivatives. An example is shown in Figure C.2. The phoneme labels corresponding to each MFCC frame are generated from the labelfiles included in the dataset. Each video has one labelfile, containing start-and end timestamps for the successive phonemes. These timestamps can be mapped to the MFCC timeframes when the MFCC window length is chosen (the standard in literature is 10ms).

Performance variations depending on the preprocessing is discussed in Section 3.4.2.

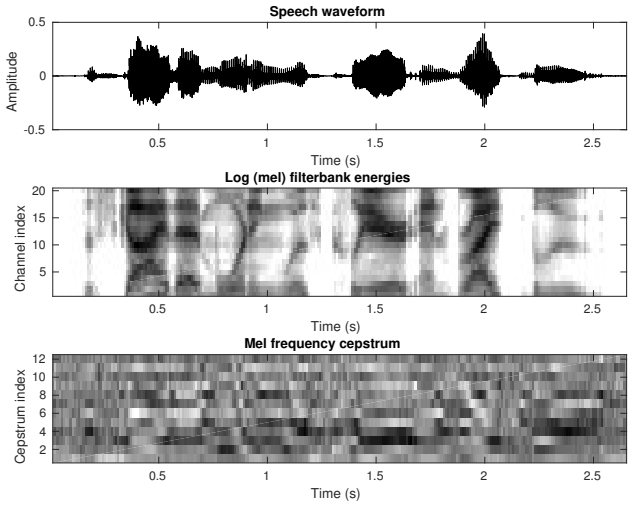


Figure C.2: A speech signal, its log filterbank energies and MFCCs

Resampling The TCDTIMIT audio files were recorded with a 48kHz sampling rate. To make comparison of performance easier, as well as reduce the data size, these files are resampled to a 16kHz sampling rate.

MFCC generation and normalization A simple way of generating MFCCs is provided by the Python library python_speech_features. The derivatives are calculated using a Python script. One MFCC is calculated for each sentence video. After the whole dataset is converted to MFCCs, the average and standard deviation is calculated. Each file is then normalized using these characteristics.

Data shapes The resulting array for each video have the shape (nbFrames, nbFeatures). nbFrames is variable per video, and nbFeatures is generally 39 (see Section C.1.1). The arrays for all the videos are put in a array, so the eventual data array has a 3D shape (nbVideos, nbFramesVideo, nbFeatures).

The arrays for the labels are arranged per video as well, but as there's only one label per timeframe (in contrast to the 39 MFCC frames), the resulting label array is 2D instead of 3D.

C.2 TCDTIMIT

In order to train a network for audio-visual speech recognition, a suitable dataset is needed. There are very few large publicly available datasets with of high quality video data and labels that are suited for lipreading. The main reason is that it's very expensive and time-consuming to record labeled datasets manually. This cost is a major reason why many datasets reported in scientific papers on lipreading are not publicly available [68].

Many audio-visual datasets have low video quality, which makes it hard for networks to train properly on the data. TCDTIMIT was chosen for this work because other publicly available audio-visual databases are either much smaller, or contain a much smaller variety in phonetic richness. Some other well-known datasets are GRID and VidTIMIT. GRID has the disadvantage that the sentences spoken are very simple, repetitive and not at all representative of real-world speech. VidTIMIT improves on this but has low video quality.

TCDTIMIT is an audio-visual dataset created by Trinity College Dublin, containing audio and video data of 62 Irish speakers. It is available online [69]. It consists of 13826 video clips in MP4 format of 62 speakers reading a total of 6913 sentences from the TIMIT corpus. In total the raw video data amounts to about 425GB. In contrast to TIMIT, the phoneme labels are not manually created, but generated using force-alignment using the HTK toolkit. This dataset is used for audio-only recognition (Chapter 3), lipreading (Chapter 4) and multimodal recognition (Chapter 5). [1]

Video quality is high (1920x1080, 30fps), with uniform lighting and a green screen as background. The video is centered on the speakers head. Audio quality is high as well, using as sampling rate of 48kHz. This sampling rate increases data sizes and computational complexity of processing the data, while such a high sampling rate is not useful for speech recognition. Therefore the audio data is resampled to 16kHz before usage.

The TCDTIMIT are mostly volunteers, both males and females. The dataset also includes three professional lipspeakers, who each speak 377 sentences. It is suggested that lipreading performance could be higher on these speakers than on the volunteers. This is evaluated in Chapter 4 for lipreading, and in Chapter 5 for audio-visual recognition.

The authors suggest to use the following train/test split:

Table 3.1: Main TCD-TIMIT train-test split used

TRAIN	01M, 02M, 03F, 04M, 05F, 06M, 07F, 08F, 09F, 10M, 11F, 12M, 14M, 16M, 17F, 18M, 19M, 20M, 22M, 26M, 32F, 33F, 38F, 39M, 40F, 41M, 42M, 44F, 45F, 46F, 48M, 49F, 50F, 52M, 56M, 58F, 59F
TEST	13F, 15F, 21M, 23M, 24M, 25M, 28M, 29M, 30F, 31F, 34M, 36F, 37F, 43F, 47M, 51F, 54M, 55F, 57M

Figure C.3: TCDTIMIT train/test split for volunteers

An example frame from a TCDTIMIT video file is shown in Figure C.4. The extracted face of the speaker is shown in C.5a and the extracted mouth region is shown in Figure C.5b.

The scripts written for this work to automatically download and preprocess the TCDTIMIT dataset are available at Github: TCDTIMITprocessing.

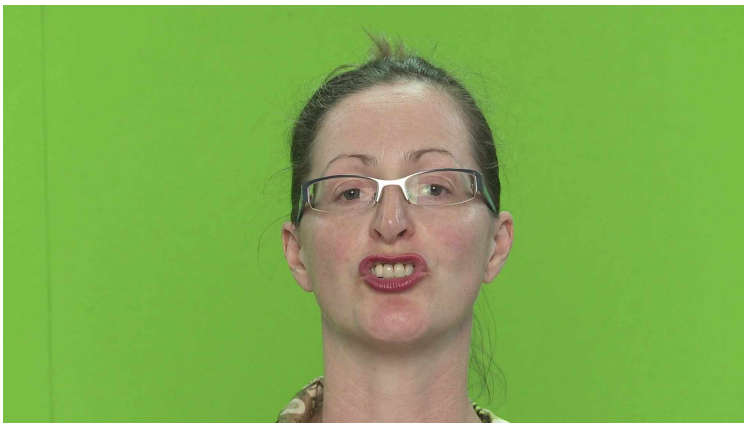


Figure C.4: A frame from TCDTIMIT

(a) A frame from a TCDTIMIT, extracted face

(b) A frame from a TCDTIMIT, extracted mouth

Figure C.5: Example images from the dataset

C.2.1 Audio Preprocessing

Preprocessing of the TCDTIMIT audio is done following the method for TIMIT. See Section C.1.1. In order to train and evaluate performance on noisy data, this data needs to be created. This is done using the Python library Pydub. Two sets were generated: one with added white noise, and one with other speakers overlaid on the actual speaker. Audio files with SNR levels of 0dB, 3dB, 5dB and 10dB were generated.

The audio with white noise was generated as follows: for each video file, the RMS power was calculated, and white noise generated added at this same RMS value. The white noise was then reduced with certain amounts (resulting in average noise power of 0dB, -3dB, -5dB, -10dB relative to the average power of the original speaker).

For the noise with other speakers, a random other audio file from the test dataset was taken and repeated until both files had the same length. It was then normalized to the same RMS audio power, reduced by the required amount and overlaid with the original audio.

C.2.2 Lipreading Preprocessing

The TCDTIMIT database contains in total about 425GB of video data. Most of that data is not useful for lipreading; the video contains a large area around the speakers head while only the region around the mouth is of relevance. Therefore, the phoneme labelfiles are used to find out the timestamps that correspond to a spoken phoneme as in Chapter 3. It is assumed that the phoneme is most clearly visible on the mouth exactly between the start and end timestamp; the timestamp of the middle of this interval is converted to the frame number in the video. When the video frame numbers of all phonemes are know, these video files are processed as follows:

1. the relevant frames are extracted using a python script that makes use of ffmpeg
2. the face of the speaker is detected using dlib, as well as the mouth
3. the face and mouth are converted to grayscale images
4. the mouth images are resampled to 120x120 pixels to provide a constant input size to the CNN
5. the image pixel values are then normalized to [0, 2] and centered around zero so results are in the interval [-1,1]

The rescaling of the images sometimes stretches the extracted mouth images, but that is not necessarily bad as it works as a crude way of data augmentation and could help generalization performance. The shape of the rescaled images is a hyperparameter, and was chosen equal to the shape used in [56] as that network achieved good performance. Conversion to grayscale images helps reduce the size of the data, which reduces training time. It also makes the network 3x smaller with little to no cost. Rescaling and normalization to [-1,1] of the image pixel values has proven to improve performance [2]. The processed data files are then converted to Numpy arrays and stored in Pythons Pickle format. The Python library used, Lasagne, organizes CNN layer inputs in 2D, 1 dimension being the batch size and the other the input values (a 1D array). Taking that into account, the data is structures in the following arrays:

1. **X**: contains the images as a numpy array with shape (colors,width,height). The images are grayscale and 120x120, so this shape will be (1,120,120). The pixel values are float32 numbers, as the backend framework Theano requires this for execution on GPU.
2. **y**: contains the true classes of the images. The values are 32-bit integers, ranging from 0 to 38 (39 phoneme classes) or from 0 to 11 (12 viseme classes).

C.2.3 For CNN-LSTM and multimodal networks

The dataset used is TCDTIMIT. When combining CNN and LSTM networks in one architecture, some issues arise. The Python library used, Lasagne, organizes CNN layer inputs in 2D, 1 dimension being the batch size and the other the input values (a 1D array) (see Section C.2.2). For lipreading using only CNNs all images from the database are put in one big array, as well as the labels. LSTM layers however, require a 3D input shape:

$$(batchSizeLSTM, nbTimeFrames, numFeatures) \tag{C.1}$$

to accommodate the extra timing information. The data arrays generated from the audio files in Section C.1.1 nicely fit this shape. The images do not as they are stored in a 2D array. To create an array structure that is usable by the LSTM networks and retain the time information for the images as well, the images have to stored in an array per video. This adds a dimension so the array is converted from 2D to 3D, corresponding to the shape of the audio data array. The labels are also reshaped in this way.

C.2.4 Attention preprocessing

The only differences with the other data is that the attention networks need to train on both noisy and clean audio data. As most of the dataset size is determined by the mouth images and these don't change when the noise is added to the audio, these images are only stored in memory once, and reused for all the different audio types.

C.3 Summary

Dataset	Speakers	Sentences	Label Method	Video
TIMIT	630	6300	fully manual	/
TCDTIMIT	59 + 3	6913	semi-automatic	1920x1080, 30fps

Table C.1: Summary of used datasets

Appendix D

Confusion Matrices

For each network a confusion matrix can be computed, where the network predictions are compared with the true classes. Several of those are shown below. The improvements from CNN only networks to CNN-LSTM networks is easily visible as the predictions are much more concentrated on the diagonal, indicating correct predictions. This only takes into account the top 1 predictions, and as shown in Chapter 4 top 3 scores are significantly higher, and those predictions can be used when the lipreading is combined with the audio network (Chapter 5). Combining the lipreading with the audio network further improves performance.

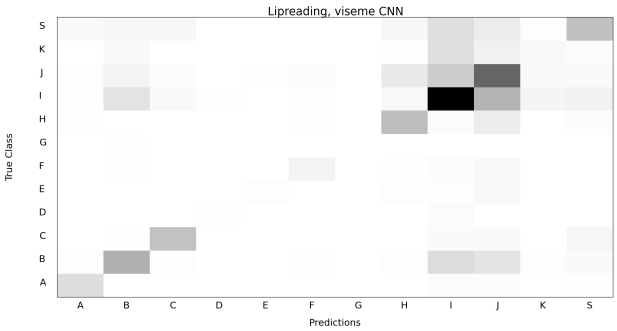


Figure D.1: Viseme Confusion Matrix for lipreading network (CNN). Some visemes hardly get any predictions while others get very many. This might be due to the phoneme-to-viseme mapping chosen (see Chapter 4)

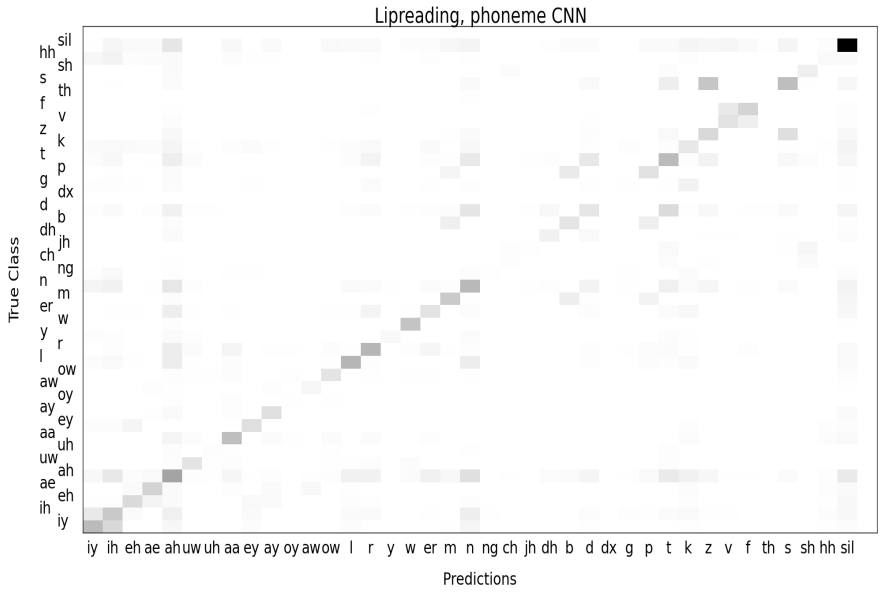


Figure D.2: Phoneme Confusion Matrix for lipreading network (CNN). There is quite a lot of confusion between phonemes (as to be expected: the same visual features can map to several phonemes (see Chapter 4))

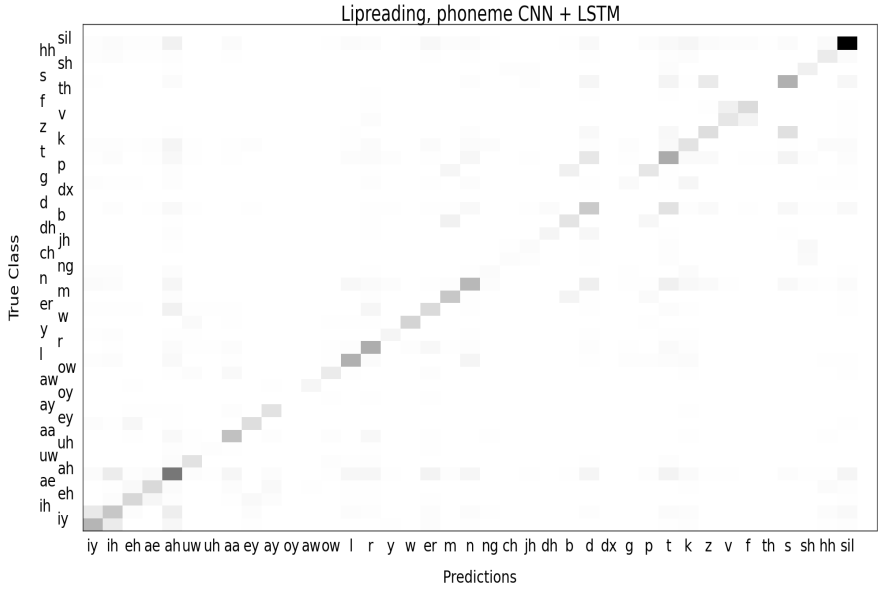


Figure D.3: Phoneme Confusion Matrix for lipreading network (CNN + LSTM). The performance is better than for CNN-only networks.

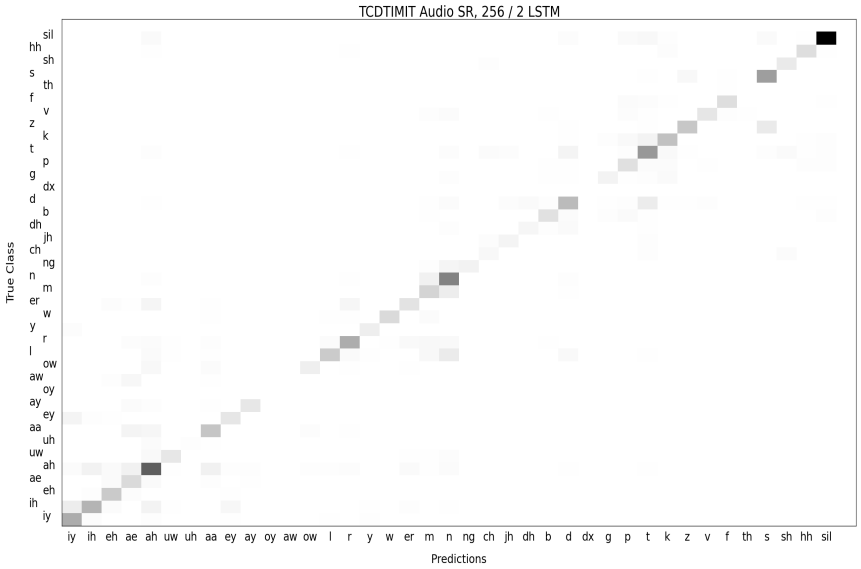


Figure D.4: Phoneme Confusion Matrix for audio-only network

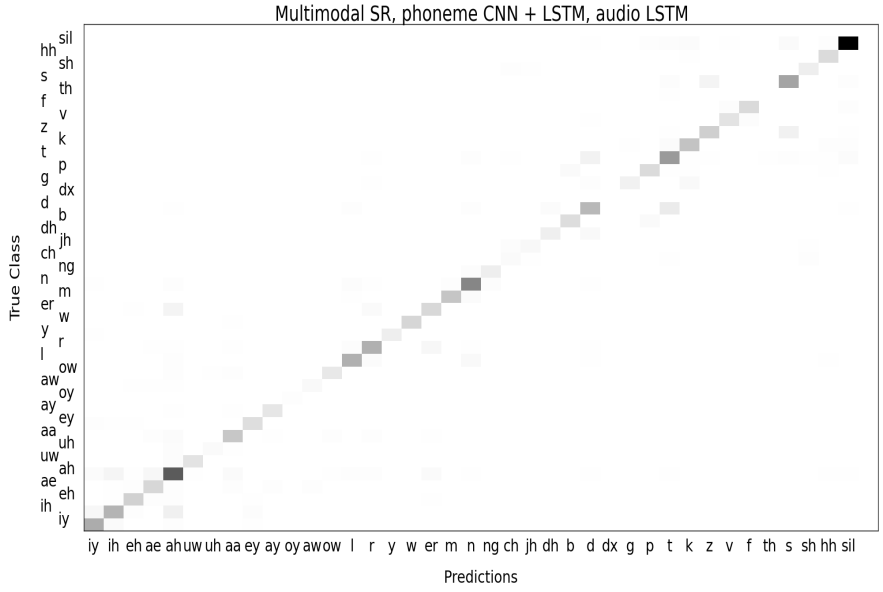


Figure D.5: Phoneme Confusion Matrix for multimodal (lipreading + audio) network. Predictions are largely concentrated on the diagonal, more so than for audio or lipreading separately. This means classification performance is higher. There is still confusion between some phonemes (for example between 't' and 'd').

Bibliography

[1] N. Harte and E. Gillen, "Tcd-timit: An audio-visual corpus of continuous speech," *IEEE Transactions on Multimedia*, vol. 17, no. 5, pp. 603–615, 2015.

[2] "Cs231n: Convolutional neural networks for visual recognition," http://cs231n.stanford.edu/, accessed: 2016-11-27.

[3] N. Kohl. Role of bias in neural networks. 2017, May 10. [Online]. Available: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks?noredirect=1&lq=1

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] I. Almajai, S. Cox, R. Harvey, and Y. 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.

[10] A. Graves, A.-r. Mohamed, and G. 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.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[12] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[13] R. Zhao, R. Yan, J. Wang, and K. Mao, "Learning to monitor machine health with convolutional bi-directional lstm networks," *Sensors*, vol. 17, no. 2, 2017. [Online]. Available: http://www.mdpi.com/1424-8220/17/2/273

[14] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *International Conference on Artificial Neural Networks*. Springer, 2005, pp. 799–804.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] "Cs224d: Deep learning for natural language processing," accessed: 2017-10-03. [Online]. Available: http://cs224d.stanford.edu/

[17] "Lstm implementation explained." [Online]. Available: https://apaszke.github.io/lstm-explained.html

[18] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid, "Convolutional kernel networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 2627–2635.

[19] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *European conference on computer vision*. Springer, 2014, pp. 584–599.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[22] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

[23] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*. IEEE, 2016, pp. 1–8.

[24] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[25] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[26] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[27] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." in *Aistats*, vol. 9, 2010, pp. 249–256.

[28] "Comparison of gradient descent algorithms," accessed: 2017-10-5. [Online]. Available: http://sebastianruder.com/optimizing-gradient-descent/

[29] S. Dieleman, J. Schluter, C. Raffel, E. Olson, S. K. Sonderby, D. Nouri *et al.*, "Lasagne: First release." Aug. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878

[30] B.-H. Juang and L. R. Rabiner, "Automatic speech recognition–a brief history of the technology development," *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, vol. 1, p. 67, 2005.

[31] O. CNX. Speech processing (part 1). 2017, March 21. [Online]. Available: http://archive.cnx.org/contents/059da0cc-99cf-4701-a4cb-1c5a4764b3c8@3/lab-9a-speech-processing-part-1

[32] K. Davis, R. Biddulph, and S. Balashek, "Automatic recognition of spoken digits," *The Journal of the Acoustical Society of America*, vol. 24, no. 6, pp. 637–642, 1952.

[33] L. E. Baum, J. A. Eagon *et al.*, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc*, vol. 73, no. 3, pp. 360–363, 1967.

[34] B. H. Juang and L. R. Rabiner, "Automatic speech recognition - a brief history of the technology development."

[35] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The htk book," *Cambridge university engineering department*, vol. 3, p. 175, 2002.

[36] W. Song and J. Cai, "End-to-end deep neural network for automatic speech recognition," Technical Report CS224D, University of Stanford, Tech. Rep., 2015.

[37] D. Palaz, R. Collobert *et al.*, "Analysis of cnn-based speech recognition system using raw speech as input," Idiap, Tech. Rep., 2015.

[38] K. J. Geras, A.-r. Mohamed, R. Caruana, G. Urban, S. Wang, O. Aslan, M. Philipose, M. Richardson, and C. Sutton, "Blending lstms into cnns," *arXiv preprint arXiv:1511.06433*, 2015.

[39] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on.* IEEE, 2015, pp. 4580–4584.

[40] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems*, 2015, pp. 577–585.

[41] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," *arXiv preprint arXiv:1512.02595*, 2015.

[42] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

[43] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning.* ACM, 2006, pp. 369–376.

[44] S. Fernández, A. Graves, and J. Schmidhuber, "Phoneme recognition in timit with blstm-ctc," *arXiv preprint arXiv:0804.3269*, 2008.

[45] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. L. Y. Bengio, and A. Courville, "Towards end-to-end speech recognition with deep convolutional neural networks," *arXiv preprint arXiv:1701.02720*, 2017.

[46] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland, *The HTK Book, version 3.4.* Cambridge, UK: Cambridge University Engineering Department, 2006.

[47] X. Tian, J. Zhang, Z. Ma, Y. He, J. Wei, P. Wu, W. Situ, S. Li, and Y. Zhang, "Deep lstm for large vocabulary continuous speech recognition," *arXiv preprint arXiv:1703.07090*, 2017.

[48] L. Cappelletta and N. Harte, "Phoneme-to-viseme mapping for visual speech recognition." in *ICPRAM (2)*, 2012, pp. 322–329.

[49] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "Darpa timit acoustic-phonetic continous speech corpus," *NASA STI/Recon technical report n*, vol. 93, 1993.

[50] W. Zheng and Y. Tang, "Binarized neural networks for language modeling."

[51] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3123–3131.

[52] G. I. Chiou and J.-N. Hwang, "Lipreading from color video," *IEEE Transactions on Image Processing*, vol. 6, no. 8, pp. 1192–1195, 1997.

[53] K. He, X. Zhang, S. Ren, and J. 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.

[54] N. U. Koster. Convolutional and recurrent neural networks. 2017, May 17. [Online]. Available: https://www.slideshare.net/nervanasys/sd-meetup-12215

[55] Y. M. Assael, B. Shillingford, S. Whiteson, and N. de Freitas, "Lipnet: Sentence-level lipreading," *arXiv preprint arXiv:1611.01599*, 2016.

[56] J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman, "Lip reading sentences in the wild," *arXiv preprint arXiv:1611.05358*, 2016.

[57] C. Neti, G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, and A. Mashari, "Audio visual speech recognition," IDIAP, Tech. Rep., 2000.

[58] Z. Zhou, G. Zhao, X. Hong, and M. Pietikäinen, "A review of recent advances in visual speech decoding," *Image and vision computing*, vol. 32, no. 9, pp. 590–605, 2014.

[59] K. H. X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[60] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International Conference on Artificial Neural Networks*. Springer, 2014, pp. 281–290.

[61] G. Potamianos, C. Neti, G. Gravier, A. Garg, and A. W. Senior, "Recent advances in the automatic recognition of audiovisual speech," *Proceedings of the IEEE*, vol. 91, no. 9, pp. 1306–1326, 2003.

[62] C. Raffel and D. P. Ellis, "Feed-forward networks with attention can solve some long-term memory problems," *arXiv preprint arXiv:1512.08756*, 2015.

[63] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, 2015, pp. 2048–2057.

[64] C. Lopes and F. Perdigao, "Phone recognition on the timit database," *Speech Technologies/Book*, vol. 1, pp. 285–302, 2011.

[65] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[66] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of mfcc," *Journal of Computer science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[67] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[68] A. G. Chiţu and L. J. Rothkrantz, "Building a data corpus for audio-visual speech recognition," *Proceedings of Euromedia2007*, pp. 88–92, 2007.

[69] S. g. Trinity College Dublin. Tcd-timit. 2017, May 29. [Online]. Available: https://sigmedia.tcd.ie/TCDTIMIT/

Fiche masterproef

Student: Matthijs Van keirsbilck

Titel: Design, implementation and analysis of a deep convolutional-recurrent neural network for speech recognition through audiovisual sensor fusion

Nederlandse titel: Ontwerp, implementatie en analyse van een diep convolutioneel-recurrent neuraal netwerk voor spraakherkenning door audio-visuele sensor fusie

UDC: 621.3

Titel van het artikel: Design, implementation and analysis of a deep convolutional-recurrent neural network for speech recognition through audio-visual sensor fusion

Korte inhoud:
Automatic speech recognition systems often only use audio even though adding visual information could improve performance, especially under noisy conditions. This can be achieved by combining recurrent and convolutional neural networks. The publicly available audio-visual dataset TCDTIMIT provides larger and more varied data than previous datasets. Phoneme classification results are reported for audio-only, visual-only and multimodal lipreading enhanced speech recognition networks.

Different LSTM networks are compared for audio-only speech recognition. Several convolutional networks are compared for visual-only speech recognition, as well as the benefits of combining convolutional and recurrent networks for lipreading.

The audio-only and visual-only networks are then combined ('sensor fusion'). The performance of the multimodal network is compared with the audio-only and visual-only performances, and shown to be significantly higher, both for high and low audio SNR. Adding an attention network further improved performance for noisy audio.

The reported results improve significantly on the TCDTIMIT baseline results for audio-only, visual-only and audio-visual classification. The best multimodal network with attention mechanism and end-to-end training achieve a phoneme recognition correctness score on the TCDTIMIT lipspeakers of 74.60%, compared to 59% for the baseline result. For an audio SNR of 0dB, the score is 58.55% compared to 44% for the baseline results. This makes these results the current state-of-the-art for the TCDTIMIT dataset.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: elektrotechniek, optie Elektronica en geïntegreerde schakelingen

Promotor: Prof. dr. ir. Marian Verhelst

Assessoren: Prof. dr. ir. W. Dehaene
Prof. dr. ir. H. Van hamme

Begeleider: Ir. B. Moons