UNIVERSITY OF THESSALY

DIPLOMA THESIS

Deep learning for audio-visual speech recognition
Βαθιά μάθηση για οπτικο-ακουστική αναγνώριση ομιλίας

Author:
Alexandros KOUMPAROULIS

Supervisor: Assoc. Prof. Gerasimos POTAMIANOS
2nd committee member: Assist. Prof. Antonios ARGYRIOU

A Thesis submitted in fulfillment of the requirements for the degree of Diploma Thesis

in the

Department of Electrical and Computer Engineering

March 4, 2017

Declaration of Authorship

I, Alexandros KOUMPAROULIS, declare that this Thesis titled, "Deep learning for audio-visual speech recognition" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Diploma degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this Thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the Thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Βαθιά μάθηση για οπτικο-ακουστική αναγνώριση ομιλίας

Αλέξανδρος Κουμπαρούλης

Η αυτόματη αναγνώριση ομιλίας είναι ένα θεμελιώδες πρόβλημα που πρέπει να λυθεί για να καταστεί δυνατή η φυσική επικοινωνία μεταξύ των υπολογιστών και των ανθρώπων. Οι πρόσφατες εξελίξεις στους τομείς της υπολογιστικής όρασης και της βαθιάς μάθησης προσφέρουν στους επαγγελματίες και ερευνητές της μηχανικής μάθησης νέα εργαλεία για να πειραματιστούν, και το σημαντικότερο επιτρέπει την απόκτηση state-of-the-art αποτελεσμάτων σε σχετικά σύντομο χρονικό διάστημα. Αυτή η εργασία είναι μια εξερεύνηση των τεχνικών αυτών στο πλαίσιο της οπτικο-ακουστικης αυτόματης αναγνώρισης ομιλίας. Με τον όρο οπτικο-ακουστική εννοούμε πως για την επιλυση του προβλήματος χρησιμοποιούμε δύο κανάλια πληροφορίας, το οπτικό και το ακουστικό. Χτίζουμε ένα σύστημα βάσης χρησιμοποιώντας κλασικές μεθόδους, και το συγκρίνουμε με ένα σύστημα που βασίζεται σε τεχνικές βαθιάς μάθησης.

University of Thessaly

Abstract

Department of Electrical and Computer Engineering

Diploma Thesis

Deep learning for audio-visual speech recognition

Alexandros KOUMPAROULIS

Automatic speech recognition is a fundamental problem that must be solved to enable natural communication between computers and humans. Recent advances in the fields of computer vision and deep learning provide machine learning researchers and practitioners new tools to experiment with, and most importantly obtain state-of-the-art results in relatively short time. This Thesis is an exploration of such techniques in the framework of audio-visual automatic speech recognition (AVASR). With the term audio-visual we imply the fact that, in order to solve this problem, we exploit information from both channels, namely the auditory and visual channel. We build a baseline system based on classic methods, and compare it against a deep learning based system. We apply traditional methods and deep learning approaches in multiple stages of the AVASR pipeline.

Acknowledgements

First and foremost, I would like to extend my deepest appreciation to my advisor, Gerasimos Potamianos, for his invaluable support and guidance, which has been instrumental in the development of this thesis.
I am grateful to Argyrios Vartholomaios for our collaboration, while working on our dissertations, Adonis Gialamas, for donating his GPU in the summer of 2016 where the VFE was developed, and of course my friends (and fellow students) in Volos.
I would like to thank my family, for their support and patience throughout the last 5 years.
Last but not least, I would like to thank Ioanna I. for her patience, love and for making Volos a far more interesting place than I could ever imagine.

Contents

List of Figures

List of Tables

List of Abbreviations

AVASR	**A**udio-**V**isual **A**utomatic **S**peech **R**ecognition
ASR	**A**utomatic **S**peech **R**ecognition
BPTT	**B**ack **P**ropagation **T**hrough **T**ime
CNN	**C**onvolutional **N**eural **N**etwork
CTC	**C**onnectionist **T**emporal **C**lassification
GMM	**G**aussian **M**ixture **M**odel
GRU	**G**ated **R**ecurrent **U**nit
HMM	**H**idden **M**arkov **M**odel
LSTM	**L**ong **S**hort **T**erm **M**emory

to my dear iacovaki.

Chapter 1

Introduction

1.1 Audio-visual speech recognition

Over the years, we have experienced the enormous growth of computer capabilities. Computers have only become faster, smaller, and much power-efficient. Cray-1, a famous supercomputer from the 1975 is (and by a great margin) less powerful and less power efficient than a smartphone device in 2017. And yet, to communicate with a computer device we keep working with the same keyboard and mouse. So the question is, what is hindering us from replacing all these communication devices with a (more natural) to us, interface, speech? The answer to that is the limitations in automatic speech recognition performance. No one would replace a fully working keyboard with one that works 90% percent of the time. To date, continuous speech recognition remains a challenging problem, especially in noisy acoustic environments. Speech recognition systems usually rely only on the auditory channel. To tackle this problem however, there is no reason why one should not also exploit the visual channel. Without a doubt, the nature of human speech production and perception is bimodal. In audio-visual speech recognition we seek to use information that is present in both the auditory and the visual channels, in order to remove ambiguities introduced by noise, overlapped speech sources, etc.

From a conceptual point of view, an automatic speech recognition (ASR) system is composed of three major modules. First, the front-end module is responsible for extracting features from the input signals. Then, from those features a speech model assigns probabilities to a set of labels. The last part, the decoder acts as a transducer of a sequence of label set probabilities to a word sequence.

In an audio-visual automatic speech recognition (AVASR) system the above structure is shared, apart from the front-end module which is further partitioned into an auditory, a visual, and a feature fusion submodule. The first two submodules handle information from their respective channel, while the fusion submodule prepares the two feature streams into the final feature stream.

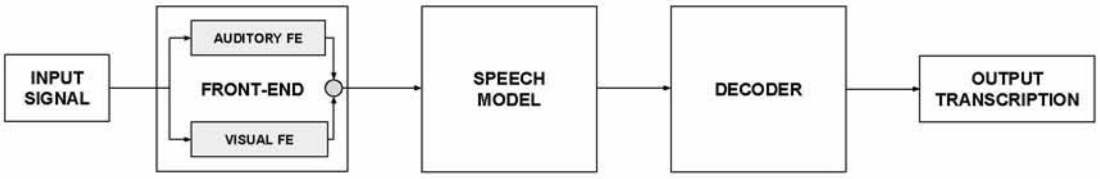

FIGURE 1.1: AVASR system diagram [1].

[1]Please do note that most textbooks refer to Speech Model as Acoustic Model, but we prefer the term Speech because of the audio-visual context.

While the promise of AVASR is of great importance, helping us communicate even in the presence of acoustic noise, several challenges must be first addressed. The visual submodule, before generating the required features, must reliably localize the region-of-interest (i.e., the speaker's mouth). This requires robust face detection, facial landmark localization, and tracking. In an ideal-case scenario (e.g., studio-like with controlled lighting conditions), face detection might be considered a trivial problem, but that is not the case in an unconstrained environment, with variations in lighting, background, human pose, etc. Furthermore, the two streams of features must be combined, without negatively affecting the system overall performance. Possible asynchrony of the two streams might also become a problem, if later in the ASR pipeline we assume that the two streams are synchronized.

1.2 Literature review

Most attempts to solve AVASR have focused on how to extract and represent visual information, as well as how and when to fuse the two information streams. While a complete literature review is out of the scope of this Thesis, we present some relevant work in this area.

Visual Information Extraction

- **Geometric features**
  Geometric information of the speaker's mouth, e.g., width,height,area, and perimeter;Lip image moments;Deformable templates, etc.

- **Video pixel features**
  An image transform (DCT, PCA, DWT, whole ROI) trained from the data to obtain a compressed representation of the ROI images. Lately convolutional neural network applied on the ROI frames have also been used to obtain a compact representation.

- **Model-based features**
  A model of the visible articulators is built, and the model parameters are used as visual features. Example of this approach include active appearance models and active shape models.

Fusion

- **Feature fusion**
  The simplest solution to represent the two channels of information is to concatenate the two feature streams. If the two have different sampling rates, interpolation is commonly used. For example, the auditory feature vectors are typically extracted at 100Hz rate, while the visual feature vectors at the videoframe rate, typically 25 or 30 fps. In this case the visual features are interpolated at 100fps. Afterward a single classifier (e.g., single-steam HMM) is used.

- **Decision fusion**
  A combination of two separate classifiers is used. In this case, each information stream has its own classifier. An example of this, is the multi-stream HMM.

For a more in depth review, the interested reader might consult [1], [2].

1.3 Thesis contribution

The main contribution of this thesis is the exploration of deep learning solutions for several steps in the AVASR pipeline. More specifically, we create a new deep learning based visual front-end based on raw images and facial feature points, which outperforms its classic counterpart, an AdaBoost solution more details are given in Chapter 3. Apart from the visual front end, we develop an auditory-only ASR and a visual-only ASR based on the HMM/GMM framework. We compare the auditory-only and visual-only ASR systems against a recurrent neural network based system (Chapter 5) and find that the later outperform the former.

1.4 Thesis Organization

The present Thesis is divided into six chapters, each of them focusing on a specific aspect of an AVASR system.

- ***Chapter*** *2* provides an overview of the databases used for AVASR in the literature, as well as details about the database used in this work.
- ***Chapter*** *3* presents the visual front-end. Emphasis is given on a comparison between a traditional method and a new deep learning based approach.
- ***Chapter 4*** presents the baseline systems (system details, results, metrics) developed for audio-only and video-only speech recognition using HMM/GMMs.
- ***Chapter*** *5* presents a deep learning approach for AVASR, using CNN and LSTM models. Analysis on the experimental results is also provided for audio-only, video-only, and audio-visual systems.
- Finally, ***Chapter 6*** summarizes and concludes the Thesis.

Chapter 2

Audio-Visual Databases

In this chapter, we provide a brief overview of the datasets used for audio-visual speech recognition in the literature, as well as detailed information about the dataset mainly used in this work.

2.1 Datasets in the literature

While several small and large-scale audio-only corpora exist, for example the freely available LibriSpeech corpora (1k hours) [3], or the well known TIMIT and SWITCHBOARD by the LDC [4], [5], this is not the case for audio-visual corpora, since suitable databases for AVSR research are very limited (in terms of number of subjects or size of vocabulary). The reasons for the lack of suitable AV corpora are several. First, compared to the audio-only speech recognition, AVASR is a relatively new field. More importantly, the time and resources it takes to record a multi-modal data corpus can be significant. Storage and distribution of a high-quality AV corpus has also been cited as issue [1], but with the high-speed internet available today and terabyte-sized hard disks, this may not remain an issue. In the following table, we summarize most of the AV datasets that have been used in the literature.

Name / Inst.	ASR Task Details										Notes
	Vowel	Consonant	Digit	Letter	Word	Phrase	C/I	Subj	Lang	RGB-D	
ICP [6]	x							1	FR		
ICP [7]	x	x						1	FR		
DAVID [8]	x	x	x	x		x		124	UK		
Tulips1 [9]			x				I	12	US		1
M2VTS [10]			x				I	37	FR		
UIUC [11]			x				C	100	US		
CUAVE [12]			x				C/I	36	US		
IBM-Digits			x				C	50	US		16
AVICAR [13]			x				C	10	US		
QuLips [14]			x				C	2	UK		11
BAVCD [15]			x				C	15	GR+UK	x	15
IBM-CHIL [16]			x				C	38	US		23
IBM-IH [17]			x				C	79	US	x	
XM2VTSDB [18]			x			x	C	295	UK		2
OuluVS2 [19]			x			x	C	52	US		
MIRACL-VC1 [20]					x	x	C/I	15	UK	x	
GRiD [21]			x	x	x	x	C	34	UK		
WAPUSK20 [22]			x	x	x	x	C	20	DE-UK		14
U.Karlsruhe [23]				x			C	6	DE		
U.LeMans [24]				x			I	2	FR		3
U.Sheffield [25]				x			C	4	UK		4
AT&T [26]				x			C	49	US		
AVLetters1-2 [27]				x			I	10	US		
UT.Austin [28]					x		I	1	US		5
AMP-CMU [29]					x		I	10	US		6
ATR [30]					x		I	1	JP		
LRW [31]					x		I	1000+	UK		

CMU-AVPFV [32]					x		I	10	US		12
DUTAVSC [33]			x	x	x			8	DU		10
HIT-AVDB-II [34]	x		x		x			30	CN		13
AusTalk [35]		x			x				AU		
AGH AV Corpus [36]						x		20	PL		18
AV-TIMIT [37]						x		1	US		7
AV-TIMIT (2) [38]						x		223	US		
Rockwell [39]						x		1	US		
VidTIMIT [40]						x		43	AU		
OuluVS1 [41]						x	C	20	US		
PKUVS [42]						x		30	CN		
MC-WSJ-AV [43]						x		45	UK		
LiLiR [44]						x		12	UK		8
TCD-TIMIT [45]						x		62	IE		9
AVOZES [46]						x		20	AU		
MODALITY [47]						x		35	UK		17
MOBIO [48]						x		150	UK		19
UNMC-VIER [49]			x			x		123	UK		20
BL-Database [50]						x		17	FR		21
IV2 [51]						x		300	FR		22
IBM-IH [17]					x			113	US	x	
UWB-05-HSAVC [52]						x		100	CZ		
IBM-AV-ViaVoice [53]						x		290	US		LVCSR

TABLE 2.1: List of databases for AVASR (some information taken from table in [1], [54], [45], [47])

Notes:

1. Only 4 digits
2. 2 digit phrases + 1 word phrases, same for all subjects.
3. 200 utterances
4. "Each talker repeated each of the letters A to Z three times, a total of 312 utterances"
5. 500 words
6. 78 words
7. Captured at 60 frames-per-second, containts 450 TIMIT sntences (TIMIT 1988)
8. Each speaker reciting 200 sentences from Resource Management Corpus (vocabulary size 1000 words)
9. 6913 total sentences
10. Includes Variations in speech rate, spelling and a small amount of prompts whispering.
11. Multiview dataset. "The resulting dataset allows for controlled comparisons between angles despite using only two cameras. 180 digits are available for each of the 10 angles and each speaker, giving a total of 3600 digits"
12. "Our vocabulary consists of 150 words from the Modified Rhyme Test (MRT) ". Subjects repeat the 150-word list 10 times.
13. "The corpus of HIT-AVDB-II includes digits, Chinese poems, tongue twisters of Chinese and English, Greek alphabets, music notes, mandarin vowel." . Simultaneously recorded in 4 different views.
14. English pronounced by German natives.
15. Bilingual with depth information.
16. 10 digits. Zero has two pronunciations (i.e., zero and oh).
17. 168 commands (isolated), recorded at 100 fps, stereo camera, microphone array
18. 25/50 fps, Isolated words and numerals, Polish Language.
19. 32 questions, recorded on mobile devices, varying head pose and illumination. Composed of over 61h of recordings of 150 speakers.
20. 12 XM2VTS sentences, varying quality, speech rate, expressions, ullumination, head poses.
21. 238 French sentences, depth camera, highlighted lips.
22. 15 French sentences, stereo frontal and profile views, iris images, 3D scanner data, head pose and illumination variations.
23. Recorded using two microphones (one head-mounted and the other mounted on a wall near the recorded subject) and three PTZ cameras (one frontal and two side views of the subject).

As observed in [1], one of the effects of the limited number of AV datasets is the difficulty to compare many algorithms that have been suggested in the AVASR literature, as they are "rarely tested on a common audio-visual database". The lack of common ground (point of reference) for AVASR databases is a commonly-cited issue in AVASR research [1], [46], [54] .

2.2 IBM corpora

Three datasets were considered for training the visual front-end (VFE), namely, the office-DIGIT, the automobile-DIGIT, and the studio-DIGIT. The office-DIGIT and the automobile-DIGIT datasets were collected especially with the goal of benchmarking the capabilities of VFE under noisy visual conditions. The office-DIGIT [1] dataset was collected in typical offices using a cheap camera (frame rate: 30Hz, resolution: 320x240 pixels). Sample frames from this dataset are shown in Figure 2.1 (first row). We can clearly see that capturing with a cheap camera resulted in color distortion. The automobile-DIGIT was recorded inside moving automobiles, and is especially challenging because of changing poses, extreme lighting changes, and shadowing [55].

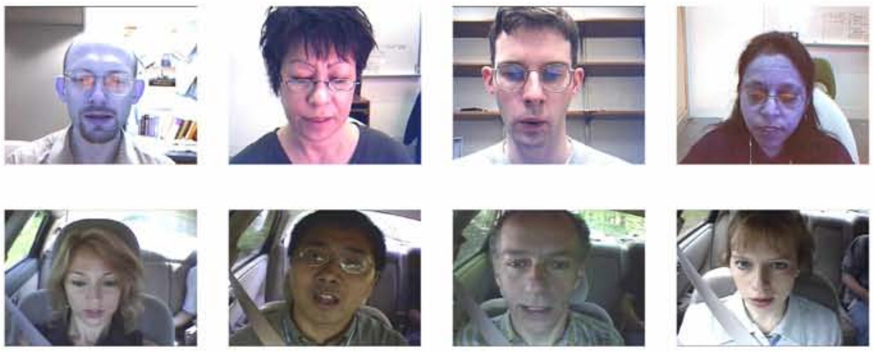
FIGURE 2.1: Sample images from the office-DIGIT (first row) and automobile-DIGIT (second row) datasets.

2.3 The studio-DIGITS dataset

Most of the work described in the rest of this Thesis, is based upon the studio-DIGITS dataset. The studio-DIGITS dataset can be classified into the small-vocabulary ASR category, since there are only 10 words/digits (the zero digit has two pronunciations "zero" and "oh") in the vocabulary. It consists of 50 subjects uttering connected digits (usually 7 or 10 digits strings), and contains about 6.7k utterances (10h). The data was collected in a quiet studio environment, with low background computer noise. The video resolution is 704x480 pixels and the frame rate is 30Hz.

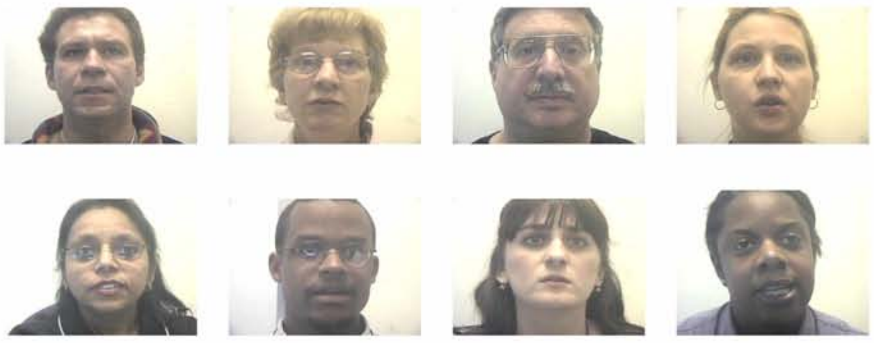
FIGURE 2.2: Sample images from the studio-DIGITS dataset.

Chapter 3

Visual Front-End Processing

3.1 Introduction

The visual front-end (VFE) is an integral part of any AVASR system. The VFE is tasked with generating feature vectors from the visual channel. In this chapter we describe the visual front-end used in Chapters 4 and 5. The main parts of a VFE are depicted in Figure 3.1. As we can see, the basic blocks have been grouped into two categories, tracking and post-processing. As the names imply, the tracking basic blocks are concerned with the face and facial landmarks localization. In Sections 3.2 and 3.3 we provide a comparison between two approaches for the tracking part. In the post-processing module, we apply consecutive data transforms on the frame pixels to obtain feature vectors. Further details are given in Sections 3.4 and 3.5. In Section 3.6 we provide the results for the two approaches for three datasets, as described in sections 2.2 and 2.3.

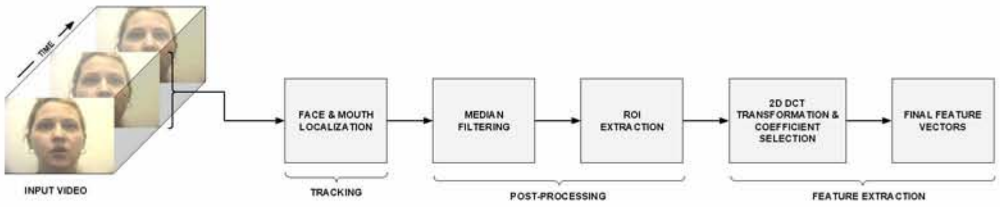

FIGURE 3.1: Visual front end block diagram.

3.2 Baseline face and mouth region tracking

An Adaboost based face and mouth detection system is developed in [56] . Essentially, this system performs a face detection step, and then it performs a mouth detection step on the returned lower half of the resulting face box. The interested reader is advised to refer to Chapter 3 in [56] for a more detailed presentation.

3.3 Deep learning for face and mouth region tracking

Recently, deep learning approaches have obtained very high performance across many different fields e.g., computer vision, speech recognition and natural language processing, among others. For the computer vision community, the tipping point to abandon classic methods and use deep learning approaches was a submission [57] in the ImageNet LSVRC-2012 Challenge. Krizhevsky et al. achieved 15.3% (top-5 test error), while the second-best entry achieved 26.2%. The model used there was a convolutional neural network (CNN) with five convolutional layers and three fully connected layers. Since then, CNNs (or variants) have been used for object recognition/localization, image segnementation, style transfer, etc. In this work, we follow a similar approach. From a dataset of full-face frontal annotated images, we train a CNN to localize facial keypoints. This CNN is later applied on individual video frames, as part of the VFE. In the following sections, we provide an introduction from perceptons to neural networks to deep learning. The term "deep" refers to the fact that the models are composed of several (stacked) layers. For a review on facial feature point detection please see [58].

3.3.1 Perceptron

In 1957, Rosenblatt invented the perceptron algorithm [59]. The perceptron is an algorithm for learning a binary linear classifier, followed by a non-linearity f to yield the class labels: $f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$

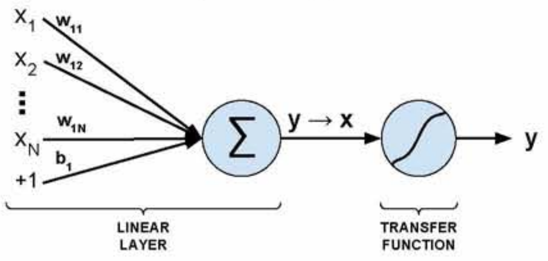

FIGURE 3.2: Perceptron diagram.

In more detail, a dot product between an extended input vector $\vec{x}$ and the weights of the perceptron $\vec{w}$ is first calculated. Then the transfer function maps the result of the previous operation to 0 or 1. Multiple perceptrons stacked in layers form a multilayer perceptron (MLP) or neural network. By using a differentiable transfer function (such as those presented in the following section), we can learn the network parameters by using the backpropagation algorithm [60]. For simplicity we omit the backward pass in the layers presented.

Linear layer

The linear layer, simply put, is a matrix-vector multiplication, where the matrix holds the model parameters. When M = 1, the layer is equivalent to the linear layer of a single-layer perceptron.

Input: $\vec{x}$, size Nx1

Output: $\vec{y}$, size Mx1

Operation: $\vec{y} = A \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$

Parameters: $A = \left[\begin{array}{cccc|c} w_{11} & w_{12} & \cdots & w_{1N} & b_1 \\ w_{21} & w_{22} & \cdots & w_{2N} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MN} & b_M \end{array}\right]$, size Mx(N+1)

Transfer function

The transfer function (also called activation function) applies (element-wise) a non-linear function to the input vector. Usual choices are the sigmoid, tanh and ReLU functions. We only present here the ReLU function.
Input: $\vec{x}$, size Nx1

Output: $\vec{y}$, size Nx1

Operation: $\vec{y} = ReLU(\vec{x})$

$$ReLU(x_i) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0 \end{cases}$$

Parameters: None

3.3.2 Convolutional neural networks

Convolutional neural networks (CNNs) are biologically-inspired variants of MLPs. In a CNN, there is at least one convolutional layer, where the discrete convolution operation is applied. CNNs were first introduced (as a network architecture) in 1969 [61], but it wasn't until 1998 [62] when backpropagation was used to learn the parameters of the convolutional layers.

Convolutional layer

For the purposes of this presentation, and without loss of generality, we assume an M-by-N RGB image input (L=3 planes), a 3x3 convolution kernel, and a single output plane (i.e., there is only one convolution kernel), as shown in Figure 3.3. The output of a convolutional layer is usually referred to as a feature map.

Input: I, size LxMxN

Output: S, size MxN

Parameters: $K_{l,m,n}$

Operation:

$$S_{m_0,n_0} = \sum_{l=l_0-1}^{l=l_0+1} \sum_{m=m_0-1}^{m=m_0+1} \sum_{n=n_0-1}^{n=n_0+1} \mathcal{I}_{l,m,n} * K_{l-l_0+1,m-m_0+1,n-n_0+1}$$

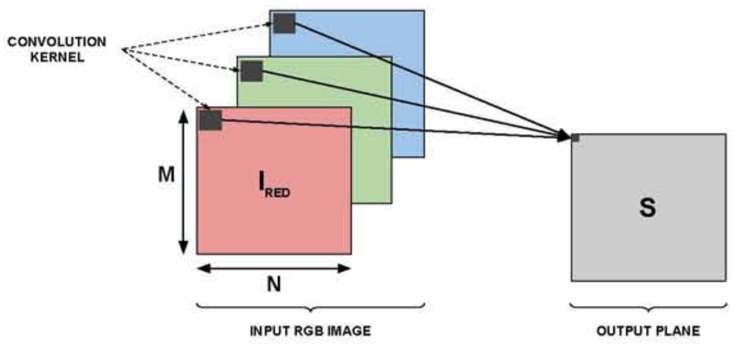

FIGURE 3.3: Convolutional layer.

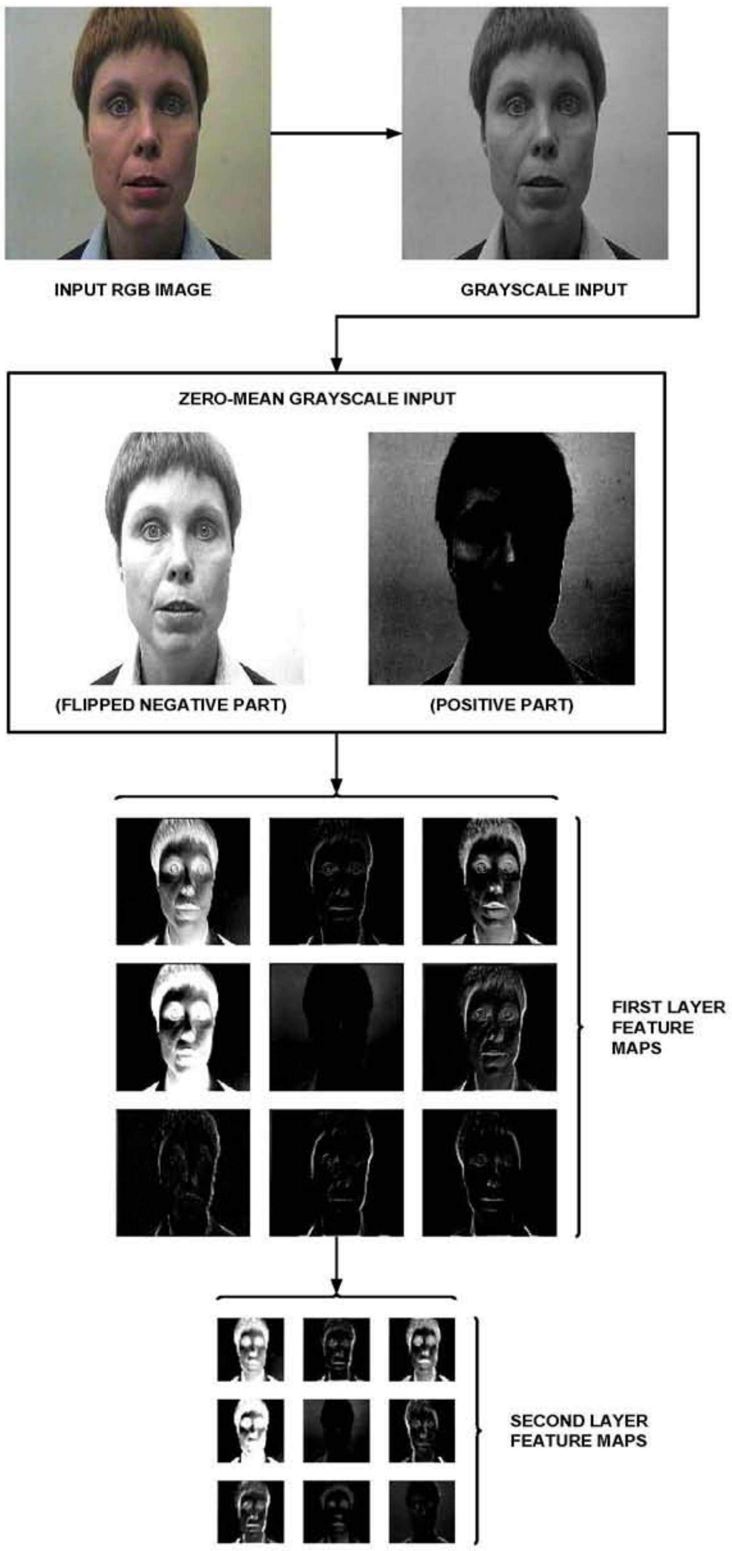

FIGURE 3.4: Example of feature maps in two CNN layers.[1] [2] [3]

[1] Please do note the spatial downsampling due to the presence of non unity stride in the convolution operation (even in the feature maps of the first layer, before any pooling operation) – unlike the simplified presentation of the convolutional layer.

[2] The input image to the first convolutional layer is grayscale, i.e. there is only one plane (L=1), in comparison to an RGB image, where there are three planes (L=3).

[3] Only the positive part is shown in the feature map images.

In figure 3.4 we present feature maps from two CNN layers. First the RGB input image is convert to grayscale, then the grayscale image is zero-mean centered (zero-mean centering results in pixels with negative values which can not be displayed, for this reason we present two images, one with the positive pixels and one with the inverted negative pixels). The zero-mean image, is fed into the CNN generating the shown features maps, in the first two CNN layers. We can clearly in see in the first layer, that some convolutional kernels perform edge detection.

Pooling layer

A pooling layer replaces a region of features (of a feature map) by a single feature value. The objective is to down-sample an input representation and to provide a form of translation invariance to the model. Common pooling layers are the mean Pooling, max Pooling, L2 Pooling, etc. We show in Figure 3.5 the result of mean and max pooling layers. Please do note that in this example, the regions of features are of size 2x2, and there is no overlap.

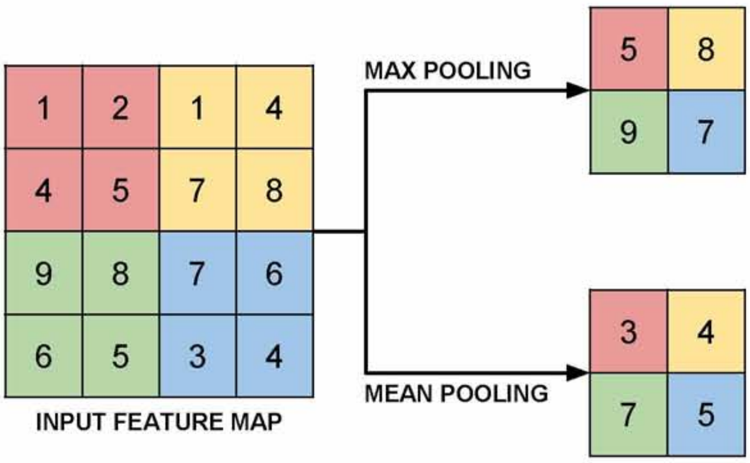

FIGURE 3.5: Max and mean pooling layers.

To summarize, we present a very early [62] CNN (namely, LeNet-5) used for digit classification. As we can see from the network structure, there are two convolutional layers, two pooling layers, and 3 fully connected layers. LeNet-5 is the earliest documented case of a CNN obtaining a very high performance (>99.3% accuracy) on an image classification task.

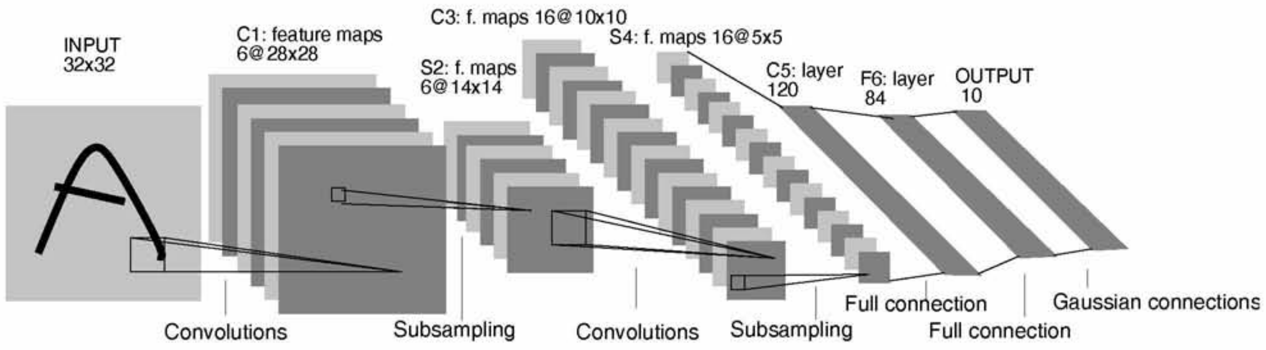

FIGURE 3.6: Example CNN LeNet-5 – Figure from [62].

3.3.3 VGG-Face

The first model we trained is based on [63]. Due to lack of computational resources at the time, we choose to take a transfer learning approach. More specifically, we use an already trained model (the VGG-Face model) as a fixed feature extractor. By that we mean that we remove the (last) fully connected layer of the VGG network, and we use the output of the last convolutional/pooling layer as a feature vector for a new classifier that we train to localize the points of interest. The results were satisfactory as will become apparent in Section 3.6.2.

3.3.4 CNN from scratch

We also experimented with training a CNN from scratch, as discussed in more detail in section 3.6.3.

3.4 ROI post-processing

When every frame of the input video has been passed through our CNN, we ontain for each frame a set of points – the predictions of the network. To go from those predictions to the final feature vectors used by the speech model, we need to perform few post-processing steps. We first apply a smoothing function on the network predictions – to absorb any noisy predictions – and then from the corrected predictions we extract a rectangle from every frame. This way, we create a new video that only contains the Region-Of-Interest (i.e., the speaker mouth).

For the smoothing function we used a median filter [64]. Median filtering is a nonlinear digital filtering technique, often used to remove noise. Typical scenarios involving median filtering can be found in a pre-processing steps in many digital image processing applications.

After obtaining the smoothed coordinates of each facial keypoint for every frame, we calculate from those points the mean mouth height and width. We also calculate the center point of the mouth. Based on these values, we extract a rectangle and then resize it to 64x64 pixels. The final ROI consists of 64x64 sized frames.

3.5 Feature extraction

At this stage, we need to convert the ROI images into feature vectors. For this reason we follow the approach described in [65]. More specifically, for each input frame we calculate the 2D-DCT transform. We then select 15 coefficients from the ***even*** columns. The selection of the coefficient indices is achieved by first calculating the 2D-DCT tranform of multiple frames, then raising each DCT coefficient to the power of 2, and then summing all DCT coefficients across all frames. Afterwards we select the top-15 (based on value) DCT coefficients that are located in the even columns. We store those top-15 indices in a separate file for later use.

If we apply the above transform and then only from the selected coefficients apply the inverse DCT we reconstruct (to some degree) the original signal. In Figure 3.7 we observe the original image (left) and the reconstructed image (right). Clearly, we can see symmetrical mouth in the right image, while the mouth in the original image are slightly asymmetrical. The 2D-DCT equation (eq. 3.1) and the

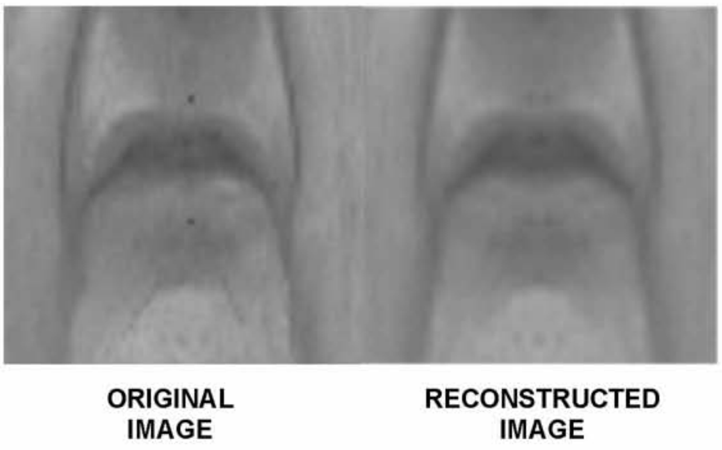

FIGURE 3.7: Original vs DCT reconstructed image (upsampled).

2D-IDCT (eq. 3.3) are given below:

$$D(i,j) = \frac{1}{\sqrt{MN}} C(i)C(j) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} p(x,y) \cos(\frac{(2x+1)i\pi}{2M}) \cos(\frac{(2y+1)j\pi}{2N}) \quad (3.1)$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases} \quad (3.2)$$

$$p(x,y) = \frac{1}{\sqrt{MN}} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} C(i)C(j)D(i,j) \cos(\frac{(2x+1)i\pi}{2M}) \cos(\frac{(2y+1)j\pi}{2N}) \quad (3.3)$$

To gain some intuitive understanding of how the above transform works with respect to the number of coefficients, we've reconstructed the input signal with 1 to 64 DCT coefficients. We calculate the PSNR (Figure 3.8) and present the resulting images on the next page (Figure 3.9). Given two MxN grayscale images $\mathcal{I}_1$ and $\mathcal{I}_2$ PSNR is defined as:

$$MSR = \frac{1}{MN} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [\mathcal{I}_1(i,j) - \mathcal{I}_2(i,j)]^2 \quad (3.4)$$

$$PSNR = 10 \log \frac{256^2}{MSE} \quad (3.5)$$

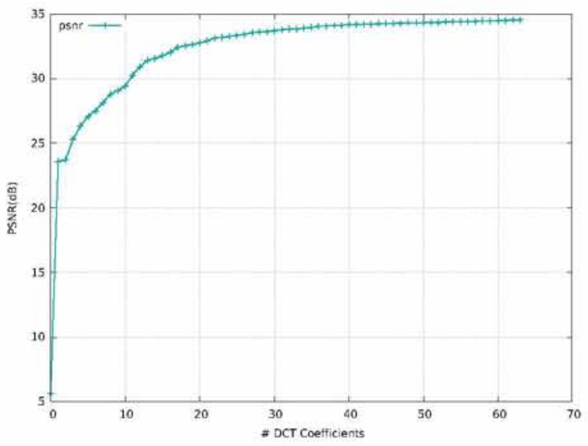

FIGURE 3.8: PSNR of the inverse DCT.

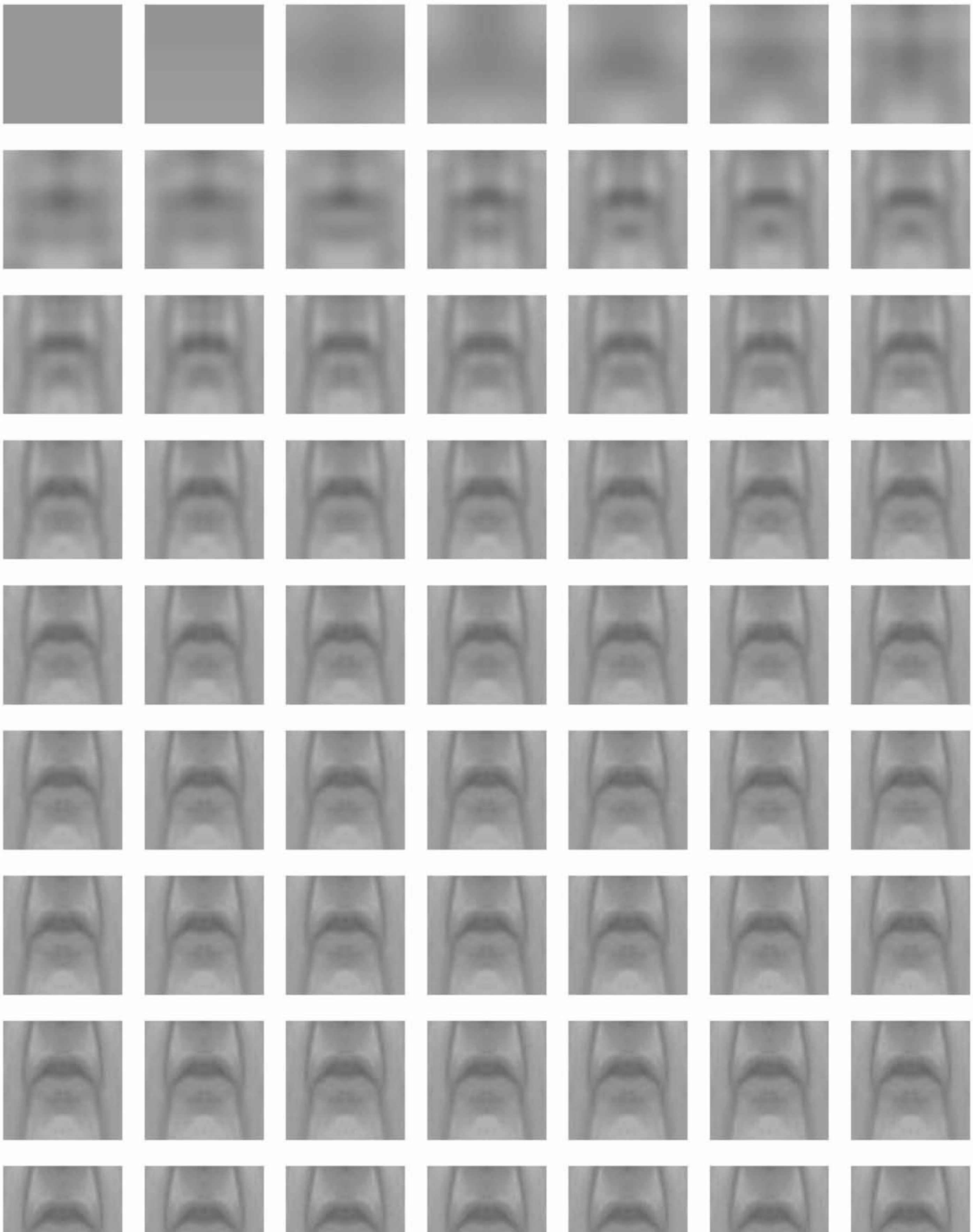

FIGURE 3.9: Reconstructed images using the inverse DCT and a variable number of DCT coefficients (1-64).

Having the DCT coefficients, we calculate the delta, and acceleration for each coefficient, resulting in 45-dim feature vectors.

3.6 Experimental tracking results

We now present the tracking results for the baseline system and all CNN architectures that we experimented with. First we introduce the evaluation metric that we used, the experimental framework, and then our results.

3.6.1 Evaluation metric

In order to evaluate performance, we use the evaluation metric specified in [66]. More specifically, for each bounding box (face, mouth) we calculate the precision and recall according to:

$$Precision(G_i, P_i) = \frac{Area(G_i) \cap Area(P_i)}{Area(P_i)} = \frac{Intersection(G_i, P_i)}{Area(P_i)} \tag{3.6}$$

$$Recall(G_i, P_i) = \frac{Area(G_i) \cap Area(P_i)}{Area(G_i)} = \frac{Intersection(G_i, P_i)}{Area(G_i)} \tag{3.7}$$

where G_i denotes the groundtruth and P_i the predicted bounded box (see Figure 3.10). Then, the $\mathrm{F}_{1\,\mathrm{score}}$ is calculated as:

$$F_{1\,\mathrm{score}}(G_i, P_i) = 2 * \frac{Precision(G_i, P_i) * Recall(G_i, P_i)}{Precision(G_i, P_i) + Recall(G_i, P_i)} \tag{3.8}$$

We only report the average $\mathrm{F}_{1\,\mathrm{score}}$, defined as:

$$F_{1\,\mathrm{score}} = \mathop{\mathbf{E}}_{i}[F_{1\,\mathrm{score}}(G_i, P_i)] \tag{3.9}$$

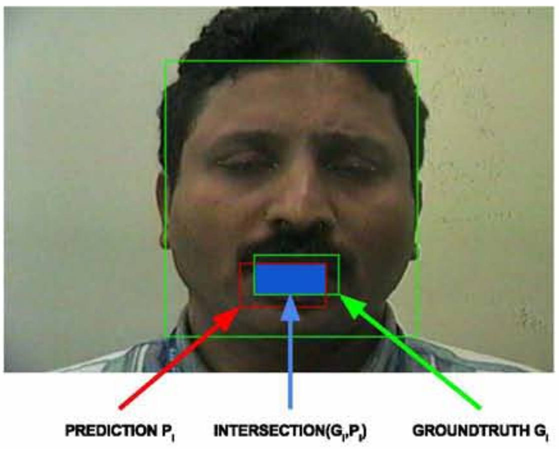

FIGURE 3.10: Ground-truth, prediction and intersection mouth bounding boxes.

3.6.2 Experimental framework

We split all the datasets into training and test sets as shown in table 3.1. We then merge the studio-DIGIT and studio-LVCSR training and test sets and train on those new sets. From those splits, we further create two version of the files. The first is a grayscale, mean zero normalized version (transformation suggested in [67]) and the second is an RGB mean-zero normalized according to the channel mean and standard deviation values given with the VGG-face model. In the next tables we refer to the dataset with their corresponding codenames for brevity. We downsample every 704x480 pixels image (of AVD/AV31/AVA) to 224x224 pixels. Before downsampling, we crop 16 pixels from left and right of the image. The AVR dataset images are of size 320x240, in this case we do not downsample, we only crop 16 pixels from left and right to obtain a 288x240 pixels image.

Dataset	Codename	Total	Train		Test	
		#	#	pct.	#	pct.
studio-DIGIT	AVD	600	200	33	400	67
studio-LVCSR	AV31	3703	2960	80	743	20
automobile-DIGIT	AVA	2575	2060	80	515	20
office-DIGIT	AVR	2620	2096	80	524	20

TABLE 3.1: Database splits for face and mouth detection experiments.

All networks were trained with the smoothL1 loss function [68], which is less sensitive to outliers than the L2 loss. The smoothL1 is defined as (see also Figure 3.11):

$$smoothL1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{if } |x| \geq 1 \end{cases} \tag{3.10}$$

Please note the use of a shifted version of the L1 function (if $|x| \geq 1$, in eq. 3.10), in order to avoid forming a discontinuous function. Our objective is to predict the location (x and y coordinates) of 18 facial landmarks and 2 points for the face bounding box, thus we apply the smoothL1 function to the difference the network prediction and the labeled value. In total, the network predicts 20 points, i.e. 40 values (x and y coordinates). Figure 3.11 shows the difference between the L1,L2 and smoothL1 loss functions. Near zero smoothL1 behaves more like L2 while for greater x values it behaves like L1.

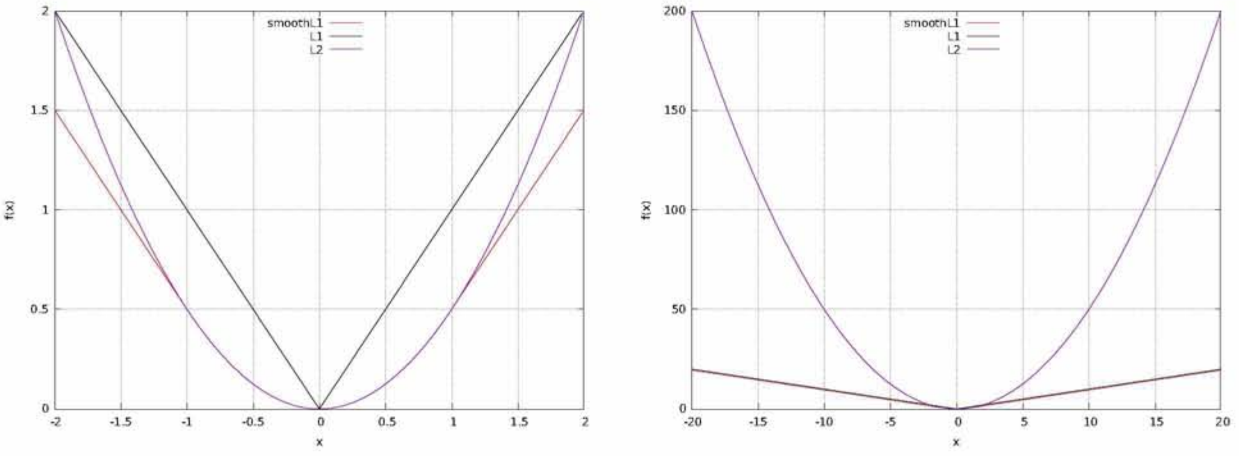

FIGURE 3.11: Plot of the L1, L2, and smoothL1 loss functions.

We now describe the CNN model architectures that we trained. These are referenced in the results section using the codenames introduced in this section.

VGG-Face based models

As we described in Section 3.3.3, initially we used the VGG-Face model as a fixed feature extractor. From 224x224 pixel images we obtain 25088-dimensional feature vectors. From those we train two networks (VGG and VGG1). VGG does not predict a face bounding box – only facial landmarks.

VGG				VGG1			
id	Layer	Input Size	Output Size	id	Layer	Input Size	Output Size
1	Linear	25088	600	1	Linear	25088	530
2	PReLU	600	600	2	PReLU	530	530
3	Dropout [69]	600	600	3	Dropout	530	530
4	Linear	600	600	4	Linear	530	530
5	HardTanh	600	600	5	HardTanh	530	530
6	Linear	600	100	6	Linear	530	100
7	Dropout	100	100	7	PReLU	100	100
8	ReLU	100	100	8	Dropout	100	100
9	Linear	100	36	9	Linear	100	40

TABLE 3.2: Characteristics of the VGG and VGG1 networks.

CNN from scratch

CNN1-5 were trained using online training. We call these models CNN1, CNN2,..., CNN5 differing in the number of training steps. More specifically, from one training session, we choose 5 networks with different epoch numbers. Later we also experimented with batched training with the following architecture:

id	Layer	Input Size	Output Size	H-parm
1	Spatial Convolution	$1 \times 224 \times 224$	$30 \times 74 \times 74$	11×11 filter, 3×3 stride
2	ReLU	$30 \times 74 \times 74$	$30 \times 74 \times 74$	
3	Spatial Max Pooling	$30 \times 74 \times 74$	$30 \times 36 \times 36$	3×3, 2×2 stride
4	Spatial Convolution	$30 \times 36 \times 36$	$156 \times 36 \times 36$	5×5 filter
5	ReLU	$156 \times 36 \times 36$	$156 \times 36 \times 36$	
6	Spatial Max Pooling	$156 \times 36 \times 36$	$156 \times 17 \times 17$	3×3, 2×2 stride
7	Spatial Convolution	$156 \times 17 \times 17$	$100 \times 17 \times 17$	3×3 filter
8	ReLU	$100 \times 17 \times 17$	$100 \times 17 \times 17$	
9	Spatial Convolution	$100 \times 17 \times 17$	$64 \times 17 \times 17$	3×3 filter
10	ReLU	$64 \times 17 \times 17$	$64 \times 17 \times 17$	
11	Spatial Max Pooling	$64 \times 17 \times 17$	$64 \times 8 \times 8$	3×3, 2×2 stride
12	Linear	4096	512	
13	ReLU	512	512	
14	Dropout	512	512	
15	Linear	512	512	
16	ReLU	512	512	
17	Dropout	512	512	
18	Linear	512	40	

TABLE 3.3: Characteristics of the CNN1-5 architecture.

id	Layer	Input Size	Output Size	H-parm
1	SpatialConvolution	$1 \times 224 \times 224$	$30 \times 113 \times 113$	5x5F, 2,2P, 3,3S
2	ReLU	$30 \times 113 \times 113$	$30 \times 113 \times 113$	
3	SpatialMaxPooling	$30 \times 113 \times 113$	$30 \times 56 \times 56$	(3x3, 2,2)
4	SpatialDropout	$30 \times 56 \times 56$	$30 \times 56 \times 56$	
5	SpatialBatchNormalization	$30 \times 56 \times 56$	$30 \times 56 \times 56$	
6	SpatialConvolution	$30 \times 56 \times 56$	$60 \times 56 \times 56$	(30 -> 60, 5x5, 1,1, 2,2)
7	ReLU	$60 \times 56 \times 56$	$60 \times 56 \times 56$	
8	SpatialMaxPooling	$60 \times 56 \times 56$	$60 \times 27 \times 27$	(3x3, 2,2)
90	SpatialDropout	$60 \times 27 \times 27$	$60 \times 27 \times 27$	
11	SpatialBatchNormalization	$60 \times 27 \times 27$	$60 \times 27 \times 27$	
12	SpatialConvolution	$60 \times 27 \times 27$	$100 \times 27 \times 27$	(60 -> 100, 3x3, 1,1, 1,1)
13	ReLU	$60 \times 27 \times 27$	$60 \times 27 \times 27$	
14	SpatialMaxPooling	$60 \times 27 \times 27$	$100 \times 13 \times 13$	(2x2, 2,2)
15	SpatialDropout [70]	$100 \times 13 \times 13$	$100 \times 13 \times 13$	
16	SpatialBatchNormalization	$100 \times 13 \times 13$	$100 \times 13 \times 13$	
17	SpatialConvolution	$100 \times 13 \times 13$	$100 \times 13 \times 13$	(100 -> 100, 3x3, 1,1, 1,1)
18	ReLU	$100 \times 13 \times 13$	$100 \times 13 \times 13$	
19	SpatialMaxPooling	$100 \times 13 \times 13$	$100 \times 6 \times 6$	(3x3, 2,2)
20	BatchNormalization [71]	3600	3600	
21	Linear	3600	512	
22	ReLU	512	512	
23	Dropout	512	512	
24	BatchNormalization	512	512	
25	Linear	512	512	
26	ReLU	512	512	
27	Dropout	512	512	
28	BatchNormalization	512	512	
29	Linear	512	40	

TABLE 3.4: Hyperparameters for batched CNN.

Dropout layer

Dropout [69] is a regularization technique for the overfitting in neural networks, by preventing complex co-adaptations on training set. A dropout layer, applied after a linear layer (or the activation layer following the linear layer) acts as a mask of zero and one on the input values. All mask positions containing zero result in zero output, otherwise the initial value is kept. The mask is sampled from a Bernoulli distribution. More specifically, if x is the input tensor, y the output, and p the mask probability:

Dropout Layer

```
1: $mask$ = tensor():resizeAs($x$)
2: $mask$:fill(Bernoulli($p$))
3: $y = x \odot mask$
```

In the above presentation and pseudo-code, we've avoided to make any assumption about the dimensions of the input x. This is a deliberately choice, since the dropout formulation does not pose any dimensionality restrictions. Of course, in some cases dimensionality restrictions might be beneficial. For example, a 2D convolutional layer with multiple convolution kernels, returns a 3D plane tensor. In this case, we would like to drop a whole plane rather than random pixel positions across all planes. Spatial dropout [70] does exactly this.

Batch normalization layer

Batch normalization [71] is a training acceleration method, enabling the use of higher learning rates. It has been long known [62] that the network training converges faster if its inputs are whitened – i.e., linearly transformed to have zero means, unit variances, and decorrelated. By whitening the data to have zero mean and unit variance, we fix the input distribution to the network. In the batch normalization term, the authors coin the term *Internal Covariate Shift* and further comment: "as the change in the distribution of network activations due to the change in network parameters during training", which practically means that the distribution of data between layers changes during training. The authors of batch normalization claim that the internal covariate shift is the major reason why deep architectures have been notoriously slow to train. This stems from the fact that deep networks do not only have to learn a new representation at each layer, but also have to account for the change in their distribution. Batch normalization is a layer that applies zero mean and unit variance at its inputs. A decorrelation transform would be too expensive (computationally) to perform.

Input: Values of x over a mini-batch: $\mathcal{B} = x_1 ... x_m$;
Output: $y_i = BN_{\gamma,\beta}(x_i)$
Parameters: γ, β (learned with backprogation)

$$\mu_{\mathcal{B}} \leftarrow \sum_{i=1}^{m} x_i \qquad \text{(mini-batch mean)} \quad (3.11)$$

$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2 \qquad \text{(mini-batch variance)} \quad (3.12)$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{(normalize)} \quad (3.13)$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta(x_i)} \qquad \text{(scale and shift)} \quad (3.14)$$

3.6.3 Results

All reported numbers concern the respective test set only. All numbers are the average across all test samples, for the given metric. First we present the results obtained from the AdaBoost-based [56] method and then from the CNN approach.

Dataset	Face			Mouth		
	Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)	Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)
AV31	73.57	91.84	81.69	40.05	87.32	54.91
AVD	79.39	89.07	83.95	64.58	68.28	66.38
AVA	76.84	59.11	66.81	51.29	44.03	47.38
AVR	74.43	81.50	77.80	54.27	62.03	57.89

TABLE 3.5: Face and mouth tracking results using the AdaBoost approach [56].

Dataset	Model	Face			Mouth		
		Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)	Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)
AV31	VGG	–	–	–	86.34	85.97	85.23
	VGG2	76.55	81.49	76.32	80.94	82.65	80.09
	ENS1	96.25	94.74	95.36	85.44	86.14	84.84
	ENS2	95.11	94.51	94.64	87.05	86.95	86.10
	CNN1	95.43	94.75	94.93	86.53	85.55	85.09
	CNN2	96.82	95.22	95.86	86.59	85.46	85.07
	CNN3	96.04	95.19	95.46	86.15	86.36	85.32
	CNN4	97.10	93.35	95.03	82.84	81.14	81.05
	CNN5	95.20	94.89	94.89	86.29	85.54	84.98
AVD	VGG	–	–	–	85.68	85.53	84.68
	VGG2	83.04	75.37	77.22	78.80	80.36	77.88
	ENS1	95.83	92.51	93.62	84.01	83.78	82.74
	ENS2	95.37	91.39	92.76	85.02	82.24	82.42
	CNN1	95.24	91.57	92.84	84.14	81.34	81.54
	CNN2	96.54	91.33	93.29	83.57	80.66	80.63
	CNN3	95.85	91.61	93.16	83.61	81.04	81.06
	CNN4	97.02	89.85	92.72	82.62	79.19	79.39
	CNN5	95.06	91.57	92.74	84.00	81.17	81.38
AVA	CNN	93.09	92.42	92.44	73.18	72.69	71.95
AVR	CNN	96.95	96.43	96.62	81.41	82.97	81.32

TABLE 3.6: Face and mouth tracking results using CNNs.

Dataset	Face			Mouth		
	Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)	Precision (%)	Recall (%)	$F_{1\text{ score}}$ (%)
AV31	96.49	97.15	96.75	85.55	87.17	85.37
AVD	97.16	96.88	96.97	85.81	83.66	83.81
AVA	93.43	93.91	93.39	80.82	79.98	79.36
AVR	96.85	97.09	96.90	83.46	83.01	82.47

TABLE 3.7: Face and mouth tracking results using the batched CNN.

Chapter 4

HMM/GMM Approach for AVASR

4.1 HMM/GMM presentation

Hidden Markov Models (HMMs) have been used for speech recognition since the mid-1970s [73]. For this reason, the first ASR systems we built are based on the traditional HMM/GMM approach.

4.1.1 HMM

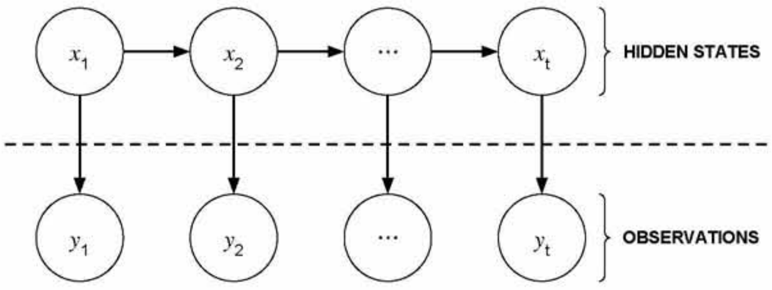

FIGURE 4.1: Schematic of an HMM.

Formally, a HMM can be described as a 5-tuple $\Omega = (\mathcal{X}, \mathcal{Y}, \mathcal{T}, \mathcal{M}, \pi)$, where :

$\mathcal{X}$	:	set	finite set of hidden states
$\mathcal{Y}$	:	set	finite set of observations
$\mathcal{T}$	:	$\mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$	transition probabilities
$\mathcal{M}$	:	$\mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$	observation probabilities
π	:	$\mathcal{X} \to \mathbb{R}^+$	prior probability distribution on the initial state

TABLE 4.1: HMM definition.

There are three basic problems associated with HMM, that must be solved before the model can be applied to practical problems [74].

[0]For this chapter we used the HTK toolbox [72]

The first problem can be stated as follows: given some observation sequence $\mathcal{Y} = \mathcal{Y}_1\mathcal{Y}_2 \cdots \mathcal{Y}_T$ and a model $\lambda = (\mathcal{X}, \mathcal{Y}, \mathcal{T}, \mathcal{M}, \pi)$ how do we efficiently compute the probability of the observation sequence $P(\mathcal{Y}|\lambda)$? The second problem can be stated as: given some observation sequence $\mathcal{Y} = \mathcal{Y}_1\mathcal{Y}_2 \cdots \mathcal{Y}_T$ and a model $\lambda = (\mathcal{X}, \mathcal{Y}, \mathcal{T}, \mathcal{M}, \pi)$, which is the most likely state sequence that produced the corresponding observation sequence Finally, how do we adjust the model parameters $\bar{\lambda} = (\mathcal{T}, \mathcal{M}, \pi)$ to maximize $P(\mathcal{Y}|\bar{\lambda})$?

4.1.2 GMM

A Gaussian mixture model (GMM) is a probabilistic model, which consists of a finite number of Gaussian distributions. It models the data observations as the linear combination of several generative Gaussians. The probability density function (pdf) of a GMM is given by:

$$f(x) = \sum_{i=1}^{k} w_i \, \mathcal{N}(x, \mu_i, \Sigma_i) \tag{4.1}$$

where k is the number of Gaussian components and w their respective weights. Finally, the function $\mathcal{N}(x, \mu_i, \Sigma_i)$ represents the pdf of the normal distribution:

$$\mathcal{N}(x, \mu_i, \mathbf{\Sigma}_i) = \frac{1}{\sqrt{(2\pi)^{|x|}|\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \mathbf{\Sigma}_i{}^{-1}(x-\mu_i)\right) \tag{4.2}$$

4.2 The audio-only ASR baseline

4.2.1 Audio features

In this section, the basic steps involved in transforming a speech waveform into a sequence of feature vectors is described. Figure 4.2 depicts the basic blocks of the acoustic front-end.

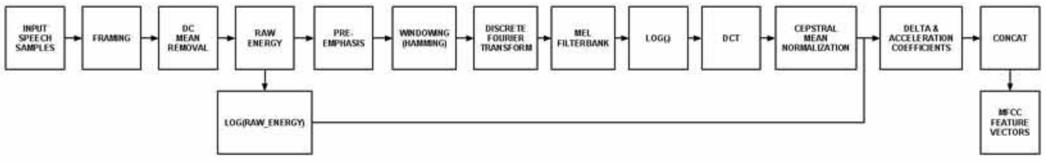

FIGURE 4.2: The acoustic front-end.

1. **Framing**

   The input signal is first split into frames, which translates into bucketing samples into their respective frames. For an input signal sampled at 16kHz (1ms $\leftrightarrow$ 16 samples), a window length of 25ms and a frame shift of 10ms, each frame has 25*16 = 400 samples. The first frame has coefficient indices in the range [0,399], the second [159,558], the third [319,718], etc. Note the partial overlap in the indices of successive frame.

2. **DC removal**

   The next step involves calculating the mean value of the input frame and subsequently subtracting the mean value from each sample. For a frame stored in a vector $\vec{x}$:

$$\mu = E[\vec{x}], \quad \text{mean of the elements of } \vec{x} \tag{4.3}$$

$$x[i] = x[i] - \mu, \quad \text{substract mean from every element} \tag{4.4}$$

3. **Raw log energy**

   This step calculates the raw energy of the input signal (zero mean):

$$raw\_energy = \sum_{n=0}^{399} x[n] * x[n] \tag{4.5}$$

   (indices of the above sum are relative to thecurrent frame). Next, the log of the raw energy is calculated:

$$raw\_energy \leftarrow \log{(raw\_energy)} \tag{4.6}$$

4. **Pre-emphasis**

   It is common practice to pre-emphasize the signal by applying the first order difference equation

$$x'[n] = x[n] - 0.97 * x[n-1] \tag{4.7}$$

   Care must be taken to apply the above equation with descending indices to avoid calculations with overwritten values.

5. **Windowing**

   Next the input frame is windowed. We use the Hamming window defined by:

$$w[n] = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{K}), & 0 \leq n < K \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

   As stated before, the input signal is sampled at 16kHz, thus each frame contains 400 samples. In this case the Hamming window is shown in Figure 4.3.

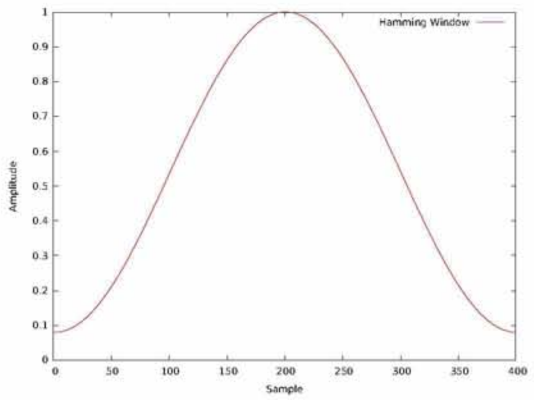

FIGURE 4.3: Hamming window for 400 samples.

   In this case the output signal can be written as:

$$x[n] = w[n] * x[n], \quad 0 \leq n < 400 \tag{4.9}$$

   (indices are relative to the current frame).

6. **Discrete Fourier Transform**

   The Discrete Fourier Transform is applied to the input signal $x[n]$:

$$X[k] = \sum_{n=0}^{399} x[n] \exp \frac{-j2\pi nk}{400}, \quad k = 0, 1, ..., 399 \tag{4.10}$$

7. **Mel filterbank**

   The magnitude of the Fourier transform is passed through a Mel filterbank. The Mel Filterbank consists of triangular filter, equally spaced along the Mel-scale (the Mel-scale is defined by (4.11)). The triangular filters are spread over the whole frequency range form zero upto the Nyquist frequency (8 kHz):

$$Mel(f) = 1127 \ln(1 + \frac{f}{700}) \tag{4.11}$$

   To go from Mels back to frequency:

$$Mel^{-1}(m) = 700(\exp \frac{m}{1125} - 1) \tag{4.12}$$

   Each one of the triangular filters is defined by:

$$H_i[k] = \begin{cases} 0 & k < f_{b_{i-1}} \\ \frac{k - f_{b_{i-1}}}{f_{b_i} - f_{b_{i-1}}}, & \text{for} \quad f_{b_{i-1}} \leq k \leq f_{b_i} \\ \frac{f_{b_{i+1}} - k}{f_{b_{i+1}} - f_{b_i}}, & \text{for} \quad f_{b_i} \leq k \leq f_{b_{i+1}} \\ 0 & \text{for} \quad k > f_{b_{i+1}} \end{cases} \tag{4.13}$$

   The boundary points f_{b_i} of the filters are defined by:

$$f_{b_i} = \frac{N}{F_s} Mel^{-1}\Big(Mel(f_{low}) + i\frac{Mel(f_{high}) - Mel(f_{low})}{M+1}\Big) \tag{4.14}$$

   where N is the number of points in the DFT, F_s is the sampling frequency, M is the number of filters, f_{low} and f_{high} are respectively the low and high boundary frequency for the entire filter bank (i.e., 0 Hz and 8000 Hz). From 4.11 we convert the lower and upper frequencies of the filterbank to Mels. In this case, 0 Hz is 0 Mels and 8000 Hz is 2834.99 Mels. We use 26 filters (NUMCHANS=26), which means we need 26 points spaced linearly between 0 Mels and 2834.99 Mels. From this list and equation 4.12 we obtain a new set of frequencies f. From this set and (4.13) we obtain the final filterbank (see also Figure 4.4).

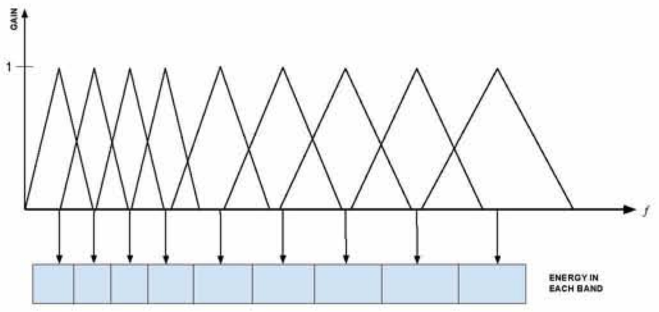

FIGURE 4.4: Example Mel-scale filterbank.

After the application of the Mel filterbank on the squared magnitude of (4.10), we obtain a vector of size 26×1 (for each frame).

8. **Log**

   The log function is applied on every element of the output of the filterbank.

9. **DCT**

   The next step is the calculation of the Mel-frequency cepstral coefficients (MFCCs), which is achieved by using the DCT:

   $$c_i = \sqrt{\frac{2}{N}} \sum_{j=1}^{N} m_j \cos\left(\frac{\pi i}{N}(j - 0.5)\right) \tag{4.15}$$

   where N is the number of the filterbank channels (26) and m_j is the output of the previous step. We use 12 cepstral coefficients, thus $1 \leq i \leq 12$. Note that this formula differs from those found in most textbooks. More specifically, it's augmented with a normalization factor $\sqrt{\frac{2}{N}}$. However this exact formula (with the normalization factor) is used in HTK version 3.5 as can found in file HSigP.c, in function FBank2MFCC and lines 610-624.

10. **Cepstral mean normalization**

    Next, cepstral mean normalization is performed on the resulting coefficients.

11. **Delta and delta delta coefficients**

    The modified DCT coefficients and the log(raw_energy) are augmented with the first and second order temporal derivatives, defined by:

    $$d_t = \frac{\sum_{\theta=1}^{\Theta} \theta(c_{t+\theta} - c_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{4.16}$$

    $$a_t = \frac{\sum_{\theta=1}^{\Theta} \theta(d_{t+\theta} - d_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{4.17}$$

    For the delta coefficients we use $\theta = 3$ and for the delta delta coefficients we use $\theta = 2$.

12. **Concatenation**

    The DCT coefficients, the log(raw_energy) as well as, the first and second derivatives are all concatenated to form a 39-dimensional feature vector for each frame of the input speech signal.

13. **Feature extraction with HTK**

    The described feature extraction process can be achieved with the `HCopy` tool. More specifically we use the following hcopy_configuration file:

```
TARGETKIND = MFCC_E_D_A_Z # MFCC+ENERGY+DELTAS+ACCEL
SOURCERATE = 625 # 625->sampling freq == 16k
TARGETRATE = 100000.0
LOFREQ = 0
HIFREQ = 8000
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T
ZMEANSOURCE = T
SOURCEFORMAT = WAV
```

and the command:

```
HCopy -A -D -V -T 1 -C ../configs/hcopy -S filelist
```

where `filelist`, is the list of files to be converted. Each line contains two columns, the input file location and the output file location.

4.2.2 Architecture

After coding the data, we define the task grammar and dictionary. We define the task grammar as follows:

```
$digit = one | two | three | four | five | six |
seven | eight | nine | zero | oh ;
( {sil} < $digit {sil} > )
```

As we can see, there are 11 words, since both "zero" and "oh" are used. We also allow silence at the start and end of the utterance, as well as between digits. We use the `HParse` tool from the HTK toolbox to compile the above grammar into a wordnet, with the following command:

```
HParse gram wdnet
```

The pronunciation dictionary is a text file with the pronunciation for each word of the vocabulary, for this step we used the CMU dictionary [75].

```
eight            ey t
five             f ay v
four             f ow r
nine             n ay n
oh               ow
one              w ah n
seven            s eh v ih n
six              s ih k s
three            th r iy
two              t uw
zero             z iy r ow
sil          []  sil
```

To generate the above file we use the HDMan tool and the following command:

```
HDMan -A -m -w wordlist -n monophones1 -g configs/global.ded
-l dlog dict orig/cmudict
```

This command, will generate the above dict file, the monophones1 (which is just a list of unique monophones present in the dict file), plus the dlog file which the log of the process. The wordlist is the list of words in the vocabulary, so essentially for our task contains 11 words (one-nine,zero,oh). The files configs/global.ded contain the configuration for HDMan, describing any transformation needed by the user.

```
AS sp
RS cmu
MP sil sp sil
MP sil sil sp
```

The AS sp directive inserts a short-pause (sp) at the end of every pronunciation. The RS cmu directive removes any stress marks and the MP directives replace any sequence of sil and sp phones into a single sil.

We also need to convert the transcription files from word sequences into phoneme sequences. The HLEd tool is able to do this. We provide the following configuration file:

```
EX
IS sil sil
ME sil sil sil
ME sil sil sil sil
DE sp
```

and run the following command (only the command for the train set transcriptions is shown):

```
HLEd -A  -l '*' -d dict -i phones0.mlf configs/mkphones0.led
labels/train.mlf
```

Next we start from flat start monophone HMMs, then based on those trained monophones we realign the training data and train triphone HMMs. At last, since the previously mentioned HMMs involve a single Gaussian model, we experiment with splitting the single Gaussian into more mixtures.

4.2.3 System training

Flat start monophones

We start by defining an HMM prototype, as follows:

```
~o <VecSize> 39 <MFCC_E_D_A_Z>
~h "proto"
<BeginHMM>
<NumStates> 5
<State> 2
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 3
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 4
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<TransP> 5
0.0 1.0 0.0 0.0 0.0
0.0 0.6 0.4 0.0 0.0
0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.7 0.3
0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

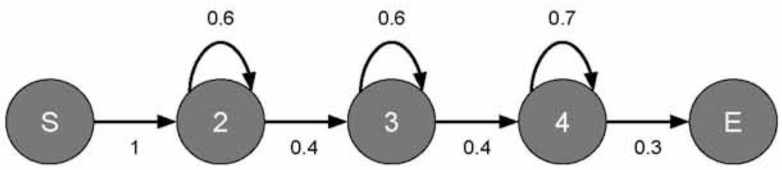

FIGURE 4.5: HMM prototype for every phone.

Next we use the HCompV tool to scan a set of data files, in order to compute the global mean and variance and set all of the Gaussians in a given HMM to have the same mean and variance, the command:

```
HCompV -C config -f 0.01 -m -S train_list -M hmm0 proto
```

Now, the hmm0 folder contains the above prototype but with different values – those estimated from the data. Next we create a *Master Macro File* (MMF) called hmmdefs containing a copy for each of the required monophone HMMs. This is constructed with the help of a bash script by copying the prototype and relabeling it (i.e., changing the first line: ~h "proto" → ~h "phoneme") for each required monophone plus "sil". Now we invoke the HERest tool and re-estimate the parameters of the HMMs.

```
HERest -A -D -C configs/herest -I phones0.mlf -t 250.0 150.0
30000.0 -S filemaps/train_mfcc -H models/hmm(i)/macros
-H models/hmm(i)/hmmdefs -M models/hmm((i+1)) monophones0
```

The command is executed for i=0,1,2.

Silence model

Next we add extra transitions from states 2 to 4 and from states 4 to 2 in the silence model. The idea behind this decision is to make the model robust to noisy data. We manually create the folder hmm4. We copy the hmmdefs file from hmm3 into hmm4 and copy the center state of the sil model to make a new sp model, the new hmmdefs file (in hmm4) has both an sp and a sil model. Next we run the following HHEd command to add the extra state transitions and tie the sp state to the center sil state.

```
HHEd -A -H models/hmm4/macros -H models/hmm4/hmmdefs
-M models/hmm5 configs/hhed monophones1
```

File `configs/hhed` contains the aforementioned directives:

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3],sp.state[2]}
```

We re-estimate the parameters (for the modified model) with `HERest`:

```
HERest -A -D -C configs/herest -I phones0.mlf_sp -t 250.0
150.0 30000.0 -S filemaps/train_mfcc -H models/hmm(i)/macros
-H models/hmm(i)/hmmdefs -M models/hmm((i+1)) monophones1
```

for i=5,6. Note the difference with the previous `HERest` command, since we now use `phones0.mlf\_sp` (which is the same as `phones0.mlf` with the difference that there is a sp between words).

Forced alignment

The phone models created so far can be used to realign the training data and create new transcriptions. `HVite` is the tool that can create a forced align transcription from the original:

```
HVite -A -l '*' -o SWT -b sil -a -H hmm7/macros -H hmm7/hmmdefs
-i aligned.mlf -m -t 250.0 150.0 1000.0 -p -100 -s 10.0 -y lab
-I labels/train.mlf -S filemaps/train_mfcc dict monophones1
```

This command uses the `hmm7` models to transform the input word level transcription (`labels/train.mlf`) to a new phone level transcription (`aligned.mlf`). The `-b` flag inserts a silence model at the start and end of each utterance. We re-estimate (based on the new `aligned.mlf`) the HMM set parameters two more times:

```
HERest -A -D -C configs/herest -I aligned.mlf -t 250.0 150.0
30000.0 -S filemaps/train_mfcc -H hmm(i)/macros -H hmm(i)/hmmdefs
-M hmm((i+1)) monophones1
```

for i=7,8.
Given the available monophone HMMs (from the last model trained), the next step is to build a context-dependent triphone HMM. First we create triphone transcriptions based on the monophone transcriptions.

Transcription conversion

To convert the monophone transcriptions into triphone transcriptions we use the `HLEd` tool:

```
HLEd -n triphones1 -l '*' -i wintri.mlf mktri.led aligned.mlf
```

where `aligned.mlf` is the forced alignment file we created before, and `mktri.led` is a configuration file with the following options:

```
WB sp
WB sil
TC
```

The two `WB` directives define `sp` and `sil` as *Word Boundary* symbols, while the `TC` directive (*Triphone Conversion*) will instruct the actual conversion. As an example, applying the `TC` directive to a sequence `sil s eh v ih n ...` will return the transformed sequence `sil+s sil-s+eh s-eh+v eh-v+ih v-ih+n ....` This style of triphone transcription is referred to as *word internal*.

Monophone HMMs to triphone HMMs

Next we use the HMM editor `HHEd` to convert the monophone models into triphone models.

```
HHEd -B -H hmm9/macros -H hmm9/hmmdefs -M hmm10 mktri.hed
monophones1
```

where `mktri.hed` is a script that contains a clone command `CL` and tie command `TI` to tie all of the transition matrices in each triphone set. We re-estimate the triphone HMM parameters twice (for i=10,11):

```
HERest -A -D -C configs/herest -I wintri.mlf -t 250.0 150.0
30000.0 -S filemaps/train_mfcc -H hmm(i)/macros -H
hmm(i)/hmmdefs -s stats -M models/hmm((i+1)) triphones1
```

The estimated triphone HHMs share the same transition matrix.

Tied-state triphones

The next step is to tie states within triphone sets in order to share data and thus be able to obtain robust parameter estimates. We use again the HMM editor `HHEd` tool to modify the latest HMM model (`hmm12`):

```
HHEd -A -H hmm12/macros -H hmm12/hmmdefs -M hmm13
configs/tree.hed triphones1
```

where `config/tree.hed` contains the instructions regarding which contexts to examine for possible clustering. HTK provides a script for automatically generating this file (`mkclscript`). The resulting file is rather long, thus it has been omitted. One important option of the `tree.hed` file is the `CO` directive, which is used to compact the model set by finding all identical models and tying them together, producing a new list of models placed in `dictionary/tiedlist.list` – this file is later used in place of `triphones1/monophones1` in `HERest`. We re-estimate the paratemers for the new models (for i=13,14):

```
HERest -A -D -C configs/herest -I wintri.mlf -t 250.0 150.0
30000.0 -S filemaps/train_mfcc -H hmm(i)/macros -H hmm(i)/hmmdefs
-s stats -M hmm(i+1) dictionary/tiedlist.list
```

Multiple mixture component HMMs

Up until now we've only considered a single Gaussian for each state. Next, we consider mixture GMMs. From our experiments, multiple Gaussians seem to improve system performance. In HTK mixture increasing can be achieved with the help of the HMM editor tool (HHEd). More specifically:

```
HHEd  -H ${OLDDIR}/hmmdefs -H ${OLDDIR}/macros -M ${NEWDIR}
tmp.hed dictionary/tiedlist.list
```

where in our experiments we use \${OLDDIR} = hmm15 (the last tied-triphone model we trained). Also, tmp.hed contains the split directives, for example:

```
MU 4 {sil.state[2-4].mix}
MU 2 {*.state[2-4].mix}
```

This could split the sil GMM into 4 components and the rest of the phone set into two components. We experiment with 2,4,8,16,32 components. Each time we reestimate the parameters 3-5 times.

4.2.4 Decoding

Decoding is performed with the help of the Viterbi algorithm, which is implemented in the HVite tool of the HTK toolbox. Furthermore, we evaluate our results with the HResults utility. We provide here two sample commands for both tools:

```
HVite -A -H hmm(i)/macros -H hmm(i)/hmmdefs -p $PENALTY
-S filemaps/test_mfcc -l '*' -i results/res_(i).mlf -w wdnet
-s 5.0 dict triphones1

HResults -I labels/test.mlf triphones1 results/res_(i).mlf
```

$PENANLTY corresponds to the insertion penalty at the phone level, in same cases we tune this value to get optimal values (typical value: -110). HVite returns a mlf file containing the transcription for each entry in the filemaps/test_mfcc. The returned mlf file is compared against the reference (labels/test.mlf) and the results are output on standard output.

4.3 Extensions for AVASR

4.3.1 Visual feature post-processing – interpolation

As we've already mentioned, the acoustic front-end generates features at a 100Hz rate. The visual front-end on the other hand generates features at 25Hz or 30Hz, which is usually the frame rate of camera recorders. In some cases, we require (in later steps) the two streams to be synchronized, for this reason, we interpolate the visual stream to 100Hz, in order to match its audio counterpart.

4.3.2 Feature fusion

Feature fusion is one of the approaches followed for audio-visual integration in AVASR systems. Essentially, the audio and visual modalities are handled as one. There are several ways this can be done, for example a simple feature concatenation at each timestep (the approach we follow on this thesis), or feature weighting (selectively increase or decrease the influence of each data stream). More sophisticated methods also exist, as discussed in [1]. Since there is only one (final) stream of information, a single classifier is trained on the combined feature stream (audio and video features). This requires the two streams to be synchronized, thus if the two feature streams have different rates, the interpolation step previously described is necessary.

4.3.3 Multi-stream architecture for decision fusion

Multi-stream architectures were first introduced in single-channel audio-only ASR, for modeling each acoustic band separately [76]. In the theory of computation, "product construction" is a standard construction for finite-state automata. Multi-stream HMMs can be considered as the stochastic equivalent. In the AVASR context, two HMMs are used to model the acoustic and visual channels.

4.4 Results

4.4.1 Error metric

The error metric we used to evaluate our models is the word error rate (WER). WER corresponds to the amount of insertions (I), deletions (D) and substitutions (S) that we must apply to a test sequence in order to obtain a reference sequence. More specifically:

$$WER = \frac{S+I+D}{N}100\% \tag{4.18}$$

where N is the number of words in the reference.

4.4.2 Experimental framework

In the following table, we present the three data splits that were used in experiments with HTK. Please note that all splits contain overlapped speakers, i.e. one speaker might be present in both sets (training and test), but on different utterances. The split names are used to present the experiments in the next section.

SplitName	# Train	%	# Test	%
ftk_split	5688	85.1	1000	14.9
ftk_split_remap	5887	88.1	800	11.9
ftk_split_remap2	5173	77.4	1514	22.6
ftk_split_remap3	4674	69.9	2013	30.1

TABLE 4.2: Dataset splits.

4.4.3 Results

id	SplitName	WER (%)	Feature			GMM Mixtutes	# passes
			MFCC	MFCC13	DCT		
1	ftk_split	3.85	x			64_32	3
2	ftk_split_remap	3.86	x			64_32	3
3	ftk_split_remap2	1.49	x			8	8
4	ftk_split_remap3	3.39	x			8	5
5	ftk_split_remap2	2.04		x		32	5
6	ftk_split_remap2	33.08			x	32_16	61

TABLE 4.3: HMM-GMM results.

Result notes

In Table 4.3 we present the results for five audio-only and one video-only ASR systems. All systems are triphone HMM-GMMs with multiple GMM components, as described previously. With MFCC we denote 39-dim features (12 MFCC coefficients + energy, deltas, and delta deltas) generated as we described in Section 4.2.1. MFCC13 is the same as MFCC, but there are 13 MFCCs instead of 12. DCT are 45-dim features (15 DCT coefficients, deltas, and delta deltas) as described in 3.5. In the GMM mixture column, we use the notation NUMA_NUMB meaning, NUMA GMM mixtures for silence and NUMB GMM mixtures for the rest of the speech units. The last column (# pass), is the number of passes to reestimate the model parameters. In Figure 4.6, we plot the visual-only model accuracy with respect to the number of passes. The model starts from 52% accuracy and after multiple passes it reaches 66.92 % accuracy.

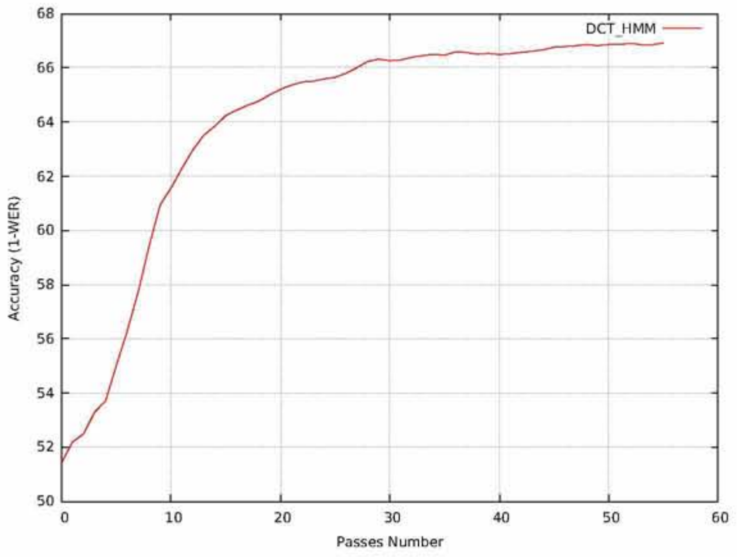

FIGURE 4.6: Video-only accuracy w.r.t. number of passes.

In Figure 4.7, we plot the best model (audio-only) accuracy with respect to the number of passes. In this case, we don't see a great benefit from re-estimating model parameters (as was the case in the video-only model), but the accuracy is

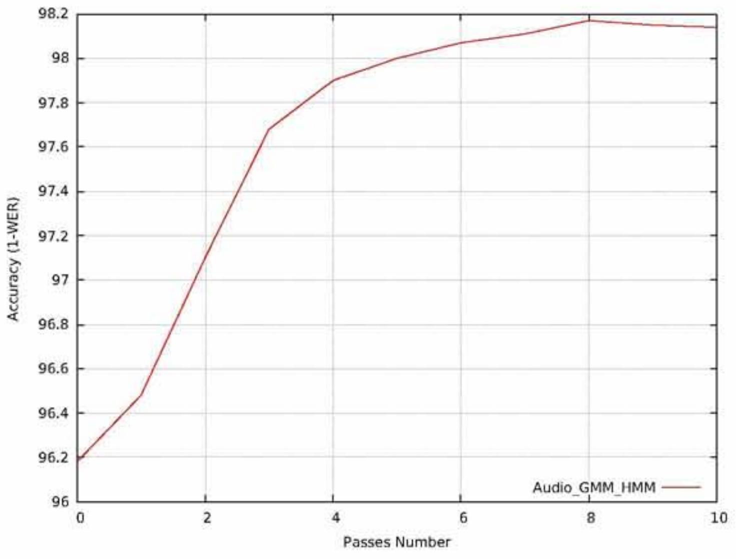

FIGURE 4.7: Audio only accuracy w.r.t. number of passes.

already high. To further increase the accuracy we tune the -p parameter in the HVITE tool. Up until now we've used a fixed value (-110), but as shown in 4.8, tuning this penalty parameter results in performance gains. For Figure 4.8, we use the model resulting from the 8th pass in Figure 4.7.

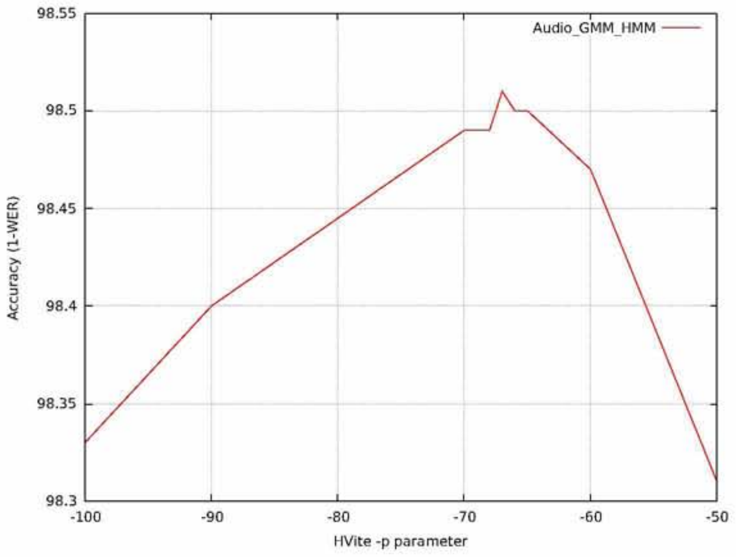

FIGURE 4.8: HVITE parameter search for audio-only ASR.

Chapter 5

LSTM Approach for AVASR

5.1 Recurrent neural networks

Deep neural networks (DNN) have been used as acoustic models and proved very successful in representing the relationship between an audio signal and the phonemes or other linguistic units that make up speech. One drawback of DNNs is the requirement of fixed input size, and therefore lack the ability to model temporal relationships present in the input signal. Recurrent neural networks (RNNs) are a family of neural nets designed for processing sequential data. A recurrent layer is defined by the following equations:

$$\mathbf{h}_t = \sigma(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h), \quad 0 < t \leq T \tag{5.1}$$

$$\mathbf{h}_0 = \vec{C}, \quad \vec{C} \text{ usually contains learned parameters or zero} \tag{5.2}$$

Simply put, the output of the RNN layer at timestep t is fed into the same RNN layer at timestep t+1, along with the input $\mathbf{x}_{(t+1)}$. In Figure 5.1 an RNN is depicted, along with the unrolled version for a number of timesteps. The attentive reader might have already noted that the weights are shared across all timesteps.

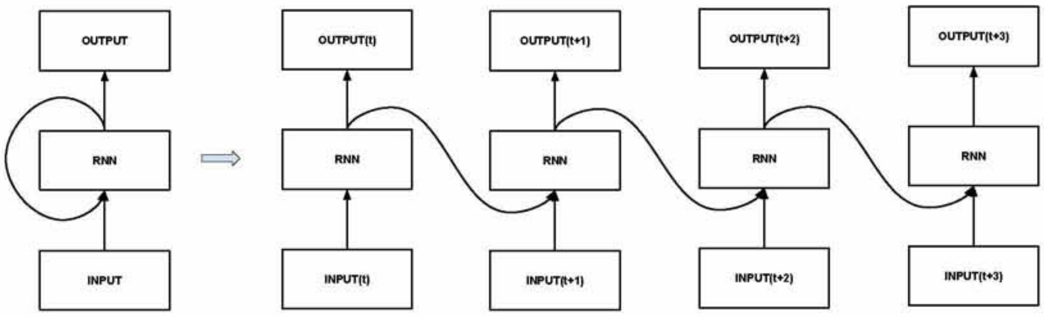

FIGURE 5.1: Unrolled RNN.

Several methods have been proposed for training recurrent neural networks, such as real time recurrent learning (RTRL) [77] and backpropagation through time (BPTT) [78]. BPTT can be considered as a generalization to the backpropagation algorithm. In BPTT, we unroll the RNN for as many steps as required by the input sequence, perform the forward pass (with shared weights across all timesteps), and in the backward pass we perform backpropagation on the unrolled network. The weight updates do not use some timestep specific gradient, but the average (or the sum) of all timestep gradients.

5.1.1 Vanishing gradients

The vanishing gradient problem is a difficulty found in training RNNs after several timesteps (and also very deep neural networks) with gradient-based backpropagation. Backpropagation essentially calculates for every parameter of the model w_i the error with respect to some error function O. For this purpose it calculates: $\frac{\partial O}{\partial w_i}$, and to do so, the product of several partial derivatives is used. If all partial derivatives are less than one, the gradient corresponding to weights in the first timesteps decreases exponentially with time, and for this reason becomes difficult to train RNNs for many timesteps. If all partial derivatives are greater than one, then the gradient at the first time steps increases exponentially with time, usually leading to overflow (exploding gradient). Hochreiter was the first to document this problem in his Diploma Thesis [79].

5.2 Long short term memory – LSTM

Long short term memory is a recurrent layer [80], specifically designed to deal with the vanishing gradients problem. A diagram of an LSTM layer is depicted in Figure 5.2.

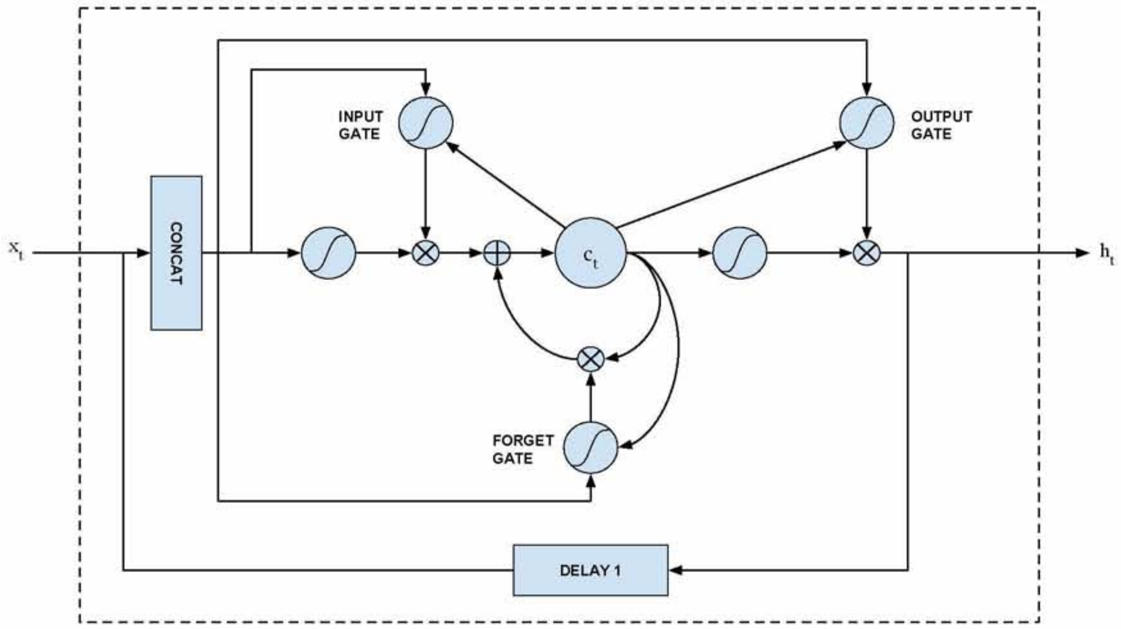

FIGURE 5.2: An LSTM layer.

5.2.1 Forward pass

The forward pass of an LSTM layer is defined in equations 5.3 through 5.7.

$$i_t = \sigma_i(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \text{input gate} \tag{5.3}$$

$$f_t = \sigma_f(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \text{forget gate} \tag{5.4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh_c(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \text{cell state} \tag{5.5}$$

$$o_t = \sigma_o(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \text{output gate} \tag{5.6}$$

$$h_t = o_t \odot tanh_h(c_t), \text{layer output at timestep t} \tag{5.7}$$

Since we opted to use the highly optimized cuDNN library for the LSTM layer, there are no peephole connections (the connections from the cell to the gates), so the figure and equations are a bit different from those used in practice.

5.2.2 Bidirectional recurrent neural networks

For the RNNs as presented until now, we assumed that the input sequence is fed at increasing timesteps into the model. This means that a forward RNN predicts timestep t given timesteps $t-1$,$t-2$,...,1. This presentation comes naturally and an approach that works for online systems. But the problem we are concerned with here is not online speech recognition, but offline. That means that at timestep t we have available all the input signal (present and past). A backward RNN [81] is able to predict timestep t given timesteps $t+1$,$t+2$,...,T. A bidirectional recurrent neural network is a recurrent network with two separate recurrent networks (backward and forward), where the training sequence is presented backwards and forwards. Both RNNs are connected to the same output layer. As a matter of fact, on the models we used in this work, we sum the (output) activations of the two networks at each timestep.

5.3 CTC loss function

In the context of ASR, an RNN is trained to predict the probability distribution of all speech units (e.g., phonemes) w.r.t. the current feature vector. So an RNN essentially is a mapping between two sequences, the input (feature vector) and the output (phoneme probabilities). This requires to know for every timestep the probability distribution of the speech units. To satisfy such a requirement we should either manually segment the audio and frame-align the transcriptions or use a pre-trained model to obtain such alignment. The first approach is time-consuming and requires significant human effort, while the second is prune to errors by the pre-trained model. An alternative solution is to question the initial requirement. Connectionist temporal classification (CTC) [82] is a loss function, enabling us to train an RNN based on unsegmented sequence data. Internally, CTC computes the probability of a sequence of phonemes for a sequence of audio frames, accounting for all possible alignments. Then we can define an objective function to maximize the probability of the phoneme sequence given the audio frame sequence, without knowing an explicit alignment between the two.

5.4 Decoding

Decoding an RNN output to words can be achieved through several approaches. One choice is to use an HMM on top of the RNN to convert phonemes into words, using the Viterbi algorithm. Another choice is the use of composition of weighed finite state transducers. Staying in an all-neural paradigm is also possible with sequence-to-sequence models, or simply performing a beam search on top of the RNN. We opt for the latter. First, for every timestep we select the most probable phoneme, obtaining a sequence of phoneme of the same length as the input sequence. Next, from this sequence we obtain a new shorter phoneme sequence, by removing consecutive same phonemes. At last, we use a reverse pronunciation dictionary to convert the phoneme sequence to a word sequence. This approach is referred to as best path decoding in the CTC paper.

5.5 RNN for ASR

Throughout the years, several ASR systems involving RNNs have been proposed. We focus on the different decoding approaches and present some relevant work, for comparison.

1. **RNN-HMM hybrid**
   One of the first RNN based ASR systems [83] was a RNN-HMM hybrid. One can consider such systems as similar to those presented in Chapter 4, with exception that the GMM part is replaced by an RNN, predicting posteriors from input feature vectors.

2. **Single RNN only**
   Such systems are usually composed of a single RNN (the acoustic model). The RNN output is used directly in conjunction with a heuristic search algorithm (e.g., beamsearch) for decoding. Language model rescoring can also be used, improving performance in practice. Examples of such systems can be found in [84], [85].

3. **RNN sequence-to-sequence (Encoder - Decoder)**
   RNNs can be used both as discriminative but also as generative models. This approach combines the double nature of RNNs into a single model. One important aspect of this class of systems is that they jointly learn spelling, language model, and the acoustic model. One of the first systems of this class can be found in [86].

4. **RNN-WFST**
   This approach uses an RNN for the acoustic model, and the decoding is achieved with the help of weighted finite state transducers (WFST). The grammar, language, and dictionary are represented as finite state transducers, and since FSTs support the composition operator, decoding is done in the search space formed from the composition of the three. EESEN [87] is a software package that supports LSTMs,CTC, and WFSTs.

5.5.1 Basic block

In this work, we've mostly used a 4-layer RNN. We present here Figure 5.3, the basic block (a single layer), which is shared among many variations of this architecture. As we can see, the input signal is applied on two (parallel) LSTM networks (a forward and backward). The activations of the two networks are summed (blue arrows). Batch normalization is applied on the summed activation (note that batch normalization was first proposed for sequential models). Batch normalization has also been applied between timesteps in recurrent networks, but we've chosen not to employ it, because that would require to use a Torch (LUA-based) LSTM implementation, and not the highly optimized CUDNN library that NVIDIA provides). The batch normalized summation of the activations is then also summed with the input signal. While the LSTM layer was designed to deal with vanishing gradients in the time domain, it was not designed to deal with this problem when multiple are stacked. We've found from our experiments that the residual connection (red arrow) reduces the training time required for the networks to converge. Use of residual connections was motivated in [88], but have also been used in speech recognition problems [89].

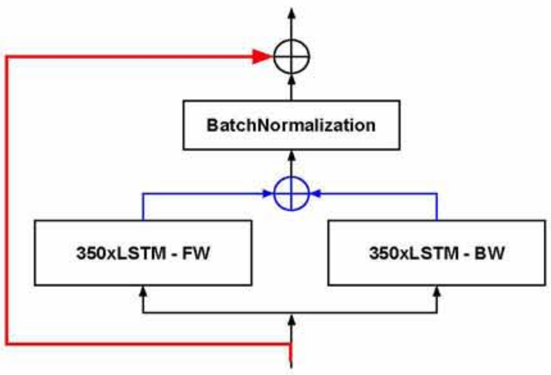

FIGURE 5.3: The resLSTM basic block.

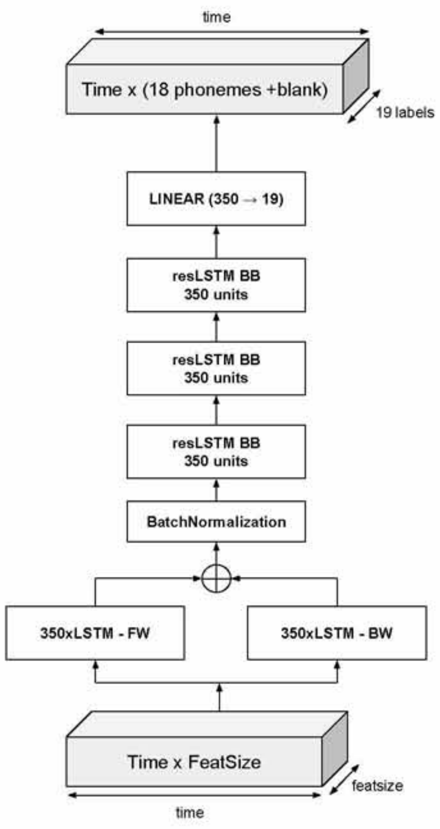

FIGURE 5.4: An LSTM network.

Figure 5.4 shows the LSTM network that we used in our experiments. It consists of four recurrent layers (with forward and backward layers, 350 nodes each, of the basic block that we presented in Figure 5.3). All recurrent layers (except from the first) have residual connections. After the last recurrent layer, we have a projection layer. Due to the use of the CTC loss function, the output labels have been augmented with the blank label addition.

5.6 Results

5.6.1 Error metric

We use the same error metric (WER) as in Chapter 4. Please refer to Section 4.4.1 for more details.

5.6.2 Experimental framework

The ftk_split_remap2 data split from Section 4.4.2 is used in all experiments presented on this section. We train models with audio features only, audio and DCT features (as discussed in Chapter 3), DCT features only and a CNN+LSTM for the visual front-end.

5.6.3 Results

In the following table we present the best WER for various network and input feature combinations. We describe the feature and network details for each entry in the text bellow the table.

ID	WER(%)	MFCC	DCT	MEL	RawVideo
net1	0.37	x	x		
net2	0.42	x			
net3	9.34		x		
net4	8.90				x
net5	0.31			x	

TABLE 5.1: LSTM-based results.

Result notes

net1
The network architecture is the one presented in Figure 5.4. We use the concatenation of the features presented in Chapter 3 (DCT features) and Chapter 4 (MFCC features). Of course, the DCT features have been interpolated at 100Hz before concatenation with the audio counterpart.

net2
The network is the same as net1, with the only difference found in the features. We only use MFCC features as described in Chapter 4.

net3
We use the same network as net1, but we only use the DCT features of the visual front-end.

net4
The network architecture is shown in Figure 5.5, essentially being a CNN followed by 3 resLSTM layers. The input is a 60x60 pixel (ROI) video.

net5
We use the same network as in net1. The log of the Mel-Filterbank is used as feature vectors, we concatenate 11 consecutive feature vectors and use the result as the input for each timestep.

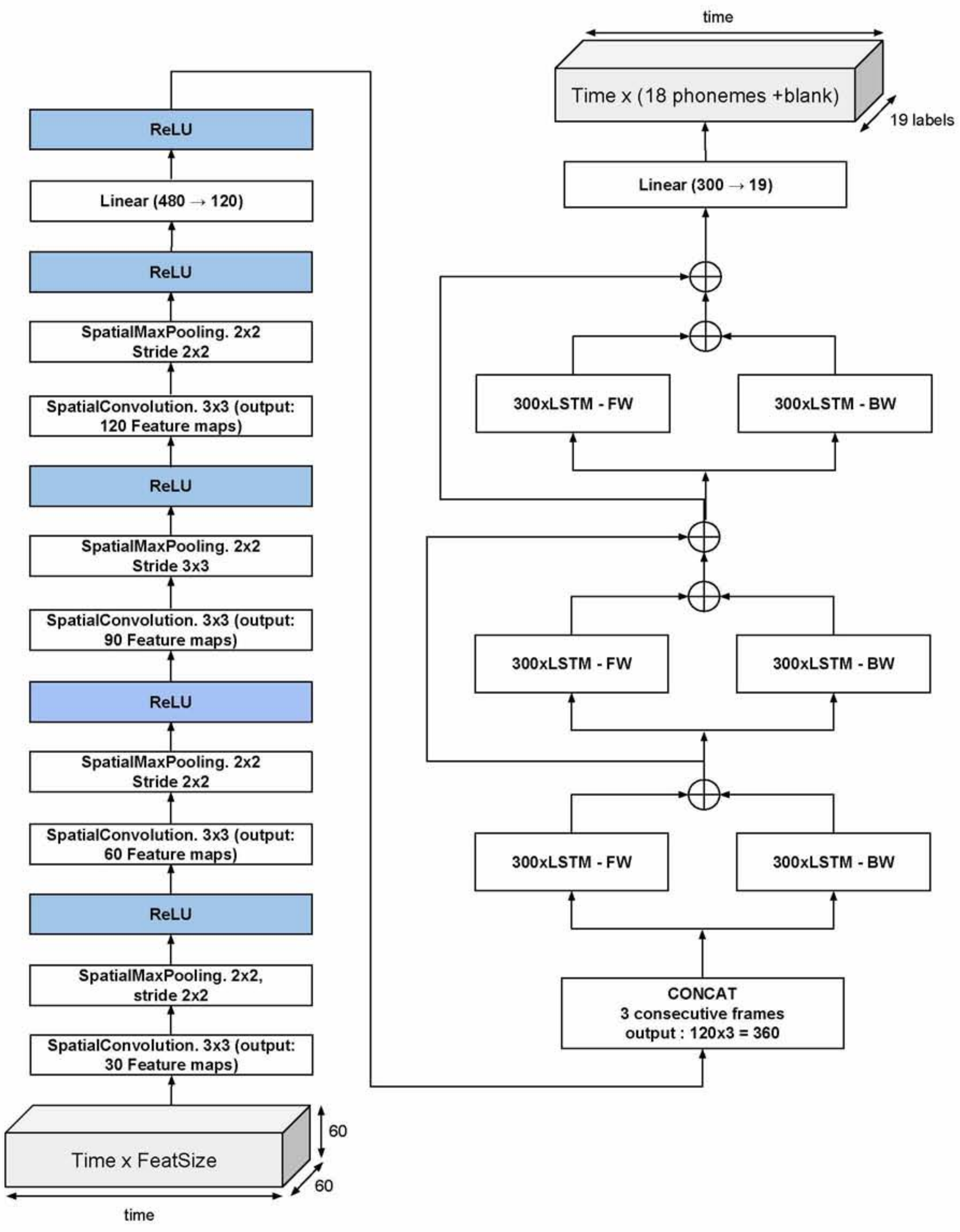

FIGURE 5.5: CNN-LSTM network for visual-only ASR.

Chapter 6

Conclusion

6.1 Summary

We have investigated the topic of audio-visual speech recognition using deep learning methodologies. This Thesis presents some of the many ways to attack this problem.

More specifically, in Chapter 3 we presented our deep learning based visual front-end and compared against an image processing based alternative. The results clearly show the advantage of a deep learning based approach. Facial keypoint recognition is a well studied problem in the literature, so future solutions in this step might be influenced by progress in the literature. A pose, lighting, and background invariant visual front-end is crucial in the development of a robust AVASR system. Chapter refChapter4 presented a traditional HMM-GMM based AVASR. We developed two systems, an audio-only and a video-only. Chapter 5 presented our deep learning based solution to the AVASR problem. One major decision was not to use a full-fledged decoder. Our systems achieved good results, surpassing those obtained in Chapter 4.

6.2 Future work

Our belief is that audio-visual speech recognition technology has prorgressed substantially to be taken out of the lab/studio environment, and be tested in the wild. Unconstrained and noisy environments is the target in AVASR applications, yet (in this work) we haven't experimented on any such dataset. Datasets originating from such conditions are crucial to the further development of robust AVASR. In the context of deep learning, large scale datasets are needed for this purpose.

In our conversation so far we haven't discussed the computational needs of a deep learning based system. The truth is that such systems are rather computationally expensive (to train and to infer), constituting one of the reasons that high performance GPUs have been crucial to the development of deep learning. In a smartphone or a tablet device with a 5-10 Watt power-fold, such a power-"hungry" hardware is out of the question. Specialized hardware such as System-On-Chip (SoC) or FPGAs and further theoretical development to decrease the computational needs (i.e., model compression) could help AVASR systems become mainstream.

Bibliography

[1] 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.

[2] 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.

[3] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015, pp. 5206–5210.

[4] J. J. Godfrey, E. C. Holliman, and J. McDaniel, "SWITCHBOARD: Telephone speech corpus for research and development", in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, vol. 1, 1992, pp. 517–520.

[5] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1", *NASA STI/Recon technical report n*, vol. 93, 1993.

[6] J. Robert-Ribes, M. Piquemal, J.-L. Schwartz, and P. Escudier, "Exploiting sensor fusion architectures and stimuli complementarity in av speech recognition", in *Speechreading by humans and machines*, Springer, 1996, pp. 193–210.

[7] A. Adjoudani and C. Benoit, "On the integration of auditory and visual parameters in an hmm-based asr", in *Speechreading by humans and machines*, Springer, 1996, pp. 461–471.

[8] C. Chibelushi, S. Gandon, J. Mason, F. Deravi, and R. Johnston, "Design issues for a digital audio-visual integrated database", 1996.

[9] J. R. Movellan and G. Chadderdon, "Channel separability in the audio-visual integration of speech: A bayesian approach", in *Speechreading by Humans and Machines*, Springer, 1996, pp. 473–487.

[10] S. Pigeon and L. Vandendorpe, "The M2VTS multimodal face database (release 1.00)", in *International Conference on Audio-and Video-Based Biometric Person Authentication*, Springer, 1997, pp. 403–409.

[11] S. M. Chu and T. S. Huang, "Bimodal speech recognition using coupled hidden markov models.", in *International Conference on Spoken Language Processing (ICSLP)*, 2000, pp. 747–750.

[12] E. K. Patterson, S. Gurbuz, Z. Tufekci, and J. N. Gowdy, "CUAVE: A new audio-visual database for multimodal human-computer interface research", in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, vol. 2, 2002, pp. 2017–2020.

[13] B. Lee, M. Hasegawa-Johnson, C. Goudeseune, S. Kamdar, S. Borys, M. Liu, and T. S. Huang, "AVICAR: Audio-visual speech corpus in a car environment.", in *International Conference on Spoken Language Processing (ICSLP)*, 2004, pp. 2489–2492.

[14] A. Pass, J. Zhang, and D. Stewart, "An investigation into features for multi-view lipreading", in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2010, pp. 2417–2420.

[15] G. Galatas, G. Potamianos, and F. Makedon, "Audio-visual speech recognition incorporating facial depth information captured by the kinect", in *Signal Processing Conference (EUSIPCO)*, IEEE, 2012, pp. 2714–2717.

[16] P. J. Lucey, G. Potamianos, and S. Sridharan, "Patch-based analysis of visual speech from multiple views", 2008.

[17] J. Huang, G. Potamianos, J. Connell, and C. Neti, "Audio-visual speech recognition using an infrared headset", *Speech Communication*, vol. 44, no. 1, pp. 83–96, 2004.

[18] K. Messer, J. Matas, J. Kittler, J. Luettin, and G. Maitre, "XM2VTSDB: The extended M2VTS database", in *International conference on audio and video-based biometric person authentication*, vol. 964, 1999, pp. 965–966.

[19] I. Anina, Z. Zhou, G. Zhao, and M. Pietikäinen, "OuluVS2: A multi-view audiovisual database for non-rigid mouth motion analysis", in *IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, IEEE, vol. 1, 2015, pp. 1–5.

[20] A. Rekik, A. Ben-Hamadou, and W. Mahdi, "A new visual speech recognition approach for RGB-D cameras", in *11th International Conference on Image Analysis and Recognition ICIAR*, 2014, pp. 21–28.

[21] M. Cooke, J. Barker, S. Cunningham, and X. Shao, "An audio-visual corpus for speech perception and automatic speech recognition", *The Journal of the Acoustical Society of America*, vol. 120, no. 5, pp. 2421–2424, 2006.

[22] A. Vorwerk, X. Wang, D. Kolossa, S. Zeiler, and R. Orglmeister, "WAPUSK20 - a database for robust audiovisual speech recognition", in *International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta: European Language Resources Association (ELRA), Mar. 2010, ISBN: 2-9517408-6-7.

[23] C. Bregler and Y. Konig, "Eigenlips for robust speech recognition", in *International Conference o Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, vol. 2, 1994, pp. 669–672.

[24] D. Alissali, P. Deleglise, and A. Rogozan, "Asynchronous integration of visual information in an automatic speech recognition system", in *International Conference on Spoken Language (ICSLP)*, IEEE, vol. 1, 1996, pp. 34–37.

[25] I. Matthews, J. A. Bangham, and S. Cox, "Audiovisual speech recognition using multiscale nonlinear image decomposition", in *International Conference on Spoken Language (ICSLP)*, IEEE, vol. 1, 1996, pp. 38–41.

[26] G. Potamianos and H. P. Graf, "Discriminative training of HMM stream exponents for audio-visual speech recognition", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, vol. 6, 1998, pp. 3733–3736.

[27] I. Matthews, T. F. Cootes, J. A. Bangham, S. Cox, and R. Harvey, "Extraction of visual features for lipreading", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 198–213, 2002.

[28] P. L. Silsbee and A. C. Bovik, "Computer lipreading for improved accuracy in automatic speech recognition", *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 5, pp. 337–351, 1996.

[29] F. J. Huang, *Advanced multimedia processing lab*. [Online]. Available: `http://amp.ece.cmu.edu/projects/AudioVisualSpeechProcessing`.

[30] S. Nakamura, H. Ito, and K. Shikano, "Stream weight optimization of speech and lip image sequence for audio-visual speech recognition", 2000.

[31] J. S. Chung and A. Zisserman, "Lip reading in the wild", in *Asian Conference on Computer Vision (ACCV)*, 2016.

[32] K. Kumar, T. Chen, and R. M. Stern, "Profile view lip reading", in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, vol. 4, 2007, pp. 429–432.

[33] J. C. Wojdeł, P. Wiggers, and L. J. Rothkrantz, "An audio-visual corpus for multimodal speech recognition in dutch language", in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, 2002, pp. 1917–1920.

[34] X. L. H. Yao and X. H. Q. Wang, "HIT-AVDB-II: A new multi-view and extreme feature cases contained audio-visual database for biometrics", 2008.

[35] S. Alghowinem, M. Wagner, and R. Goecke, "AusTalk—the australian speech database: Design framework, recording experience and localisation", in *International Conference on Information Technology in Asia (CITA)*, IEEE, 2013, pp. 1–7.

[36] "Agh university of science and technology (2014). audiovisual polish speech corpus",

[37] A. J. Goldschen, O. N. Garcia, and E. D. Petajan, "Rationale for phoneme-viseme mapping and feature selection in visual speech recognition", in *Speechreading by Humans and Machines*, Springer, 1996, pp. 505–515.

[38] T. J. Hazen, K. Saenko, C.-H. La, and J. R. Glass, "A segment-based audio-visual speech recognizer: Data collection, development, and initial experiments", in *International conference on Multimodal interfaces*, ACM, 2004, pp. 235–242.

[39] M. T. Chan, Y. Zhang, and T. S. Huang, "Real-time lip tracking and bimodal continuous speech recognition", in *IEEE Second Workshop on Multimedia Signal Processing*, IEEE, 1998, pp. 65–70.

[40] C. Sanderson and K. K. Paliwal, "Noise compensation in a person verification system using face and multiple speech features", *Pattern Recognition*, vol. 36, no. 2, pp. 293–302, 2003.

[41] G. Zhao, M. Barnard, and M. Pietikainen, "Lipreading with local spatiotemporal descriptors", *IEEE Transactions on Multimedia*, vol. 11, no. 7, pp. 1254–1265, 2009.

[42] H. Liu, X. Zhang, and P. Wu, "Regression based landmark estimation and multi-feature fusion for visual speech recognition", in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2015, pp. 808–812.

[43] M. Lincoln, I. McCowan, J. Vepa, and H. K. Maganti, "The multi-channel wall street journal audio visual corpus (mc-wsj-av): Specification and initial experiments", in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, IEEE, 2005, pp. 357–362.

[44] Y. Lan, B.-J. Theobald, R. W. Harvey, E.-J. Ong, and R. Bowden, "Improving visual features for lip-reading.", in *Auditory-Visual Speech Processing (AVSP)*, 2010, pp. 3–7.

[45] 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.

[46] R. Goecke, J. B. Millar, A. Zelinsky, and J. Robert-Ribes, "A detailed description of the AVOZES data corpus", in *Australasian Conference on Speech Science & Technology (SST)*, 2004, pp. 486–491.

[47] A. Czyzewski, B. Kostek, P. Bratoszewski, J. Kotus, and M. Szykulski, "An audio-visual corpus for multimodal automatic speech recognition", *Journal of Intelligent Information Systems*, pp. 1–26,

[48] C. McCool, S. Marcel, A. Hadid, M. Pietikäinen, P. Matejka, J. Cernocky, N. Poh, J. Kittler, A. Larcher, C. Levy, *et al.*, "Bi-modal person recognition on a mobile phone: Using mobile phone data", in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, IEEE, 2012, pp. 635–640.

[49] Y. W. Wong, S. I. Ch'ng, K. P. Seng, L.-M. Ang, S. W. Chin, W. J. Chew, and K. H. Lim, "A new multi-purpose audio-visual UNMC-VIER database with multiple variabilities", *Pattern Recognition Letters*, vol. 32, no. 13, pp. 1503–1510, 2011.

[50] Y. Benezeth, G. Bachman, G. Le-Jan, N. Souviraà-Labastie, and F. Bimbot, "Bl-database: A french audiovisual database for speech driven lip animation systems", PhD thesis, INRIA, 2011.

[51] D. Petrovska-Delacrétaz, S. Lelandais, J. Colineau, L. Chen, B. Dorizzi, M. Ardabilian, E. Krichen, M.-A. Mellakh, A. Chaari, S. Guerfi, *et al.*, "The IV 2 multimodal biometric database (including iris, 2d, 3d, stereoscopic, and talking face data), and the iv 2-2007 evaluation campaign", in *IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, IEEE, 2008, pp. 1–7.

[52] J. Trojanová, M. Hrúz, P. Campr, and M. Železny, "Design and recording of Czech audio-visual database with impaired conditions for continuous speech recognition", in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC)*, 2008.

[53] C. Neti, G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, and A. Mashari, "Audio visual speech recognition", Johns Hopkins University-CLSP, 2000.

[54] A. G. ChiŃu and L. J. Rothkrantz, "Building a data corpus for audio-visual speech recognition", *Proceedings of Euromedia2007*, vol. 91, no. 9, pp. 1306–1326, 2003.

[55] G. Iyengar and C. Neti, "Detection of faces under shadows and lighting variations", in *IEEE Fourth Workshop on Multimedia Signal Processing*, IEEE, 2001, pp. 15–20.

[56] A. Vartholomeos, "Deep learning for audio visual speaker diarization", 2017.

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

[58] N. Wang, X. Gao, D. Tao, and X. Li, "Facial feature point detection: A comprehensive survey", *ArXiv preprint arXiv:1410.1037*, 2014.

[59] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton project para.* Cornell Aeronautical Laboratory, 1957.

[60] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[61] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements", *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969.

[62] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[63] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition.", in *BMVC*, vol. 1, 2015, p. 6.

[64] N. Gallagher and G. Wise, "A theoretical analysis of the properties of median filters", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1136–1141, 1981.

[65] G. Potamianos and P. Scanlon, "Exploiting lower face symmetry in appearance-based automatic speechreading", in *Proceedings of the International Conference on Audio-Visual Speech Processing (AVSP)*, 2005, pp. 79–84.

[66] C. Wolf and J.-M. Jolion, "Object count/area graphs for the evaluation of object detection and segmentation algorithms", *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 8, no. 4, pp. 280–296, 2006.

[67] Y. LeCun, L. Bottou, G. B. Orr, and K. .-.-R. Müller, "Efficient backprop", in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50.

[68] R. Girshick, "Fast R-CNN", in *International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[69] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.", *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[70] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 648–656.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in *Proceedings of The 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[72] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, "The HTK book version 3.0", 2000.

[73] J. Baker, "The DRAGON system–an overview", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 24–29, 1975.

[74] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition", *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[75] R. Weide, "The Carnegie Mellon pronouncing dictionary", 2005. [Online]. Available: `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.

[76] H. Bourlard and S. Dupont, "A new ASR approach based on independent processing and recombination of partial frequency bands", in *International Conference on Spoken Language (ICSLP)*, IEEE, vol. 1, 1996, pp. 426–429.

[77] A. Robinson and F. Fallside, *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

[78] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity", *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[79] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen", PhD thesis, diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen, 1991.

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

[81] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks", *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[82] 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 (ICML)*, ACM, 2006, pp. 369–376.

[83] D. K. T. R. M. Hochberg, "Context-dependent classes in a hybrid recurrent network-hmm speech recognition system", 1995.

[84] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks", in *Proceedings of The 31st International Conference on Machine Learning (ICML)*, vol. 14, 2014, pp. 1764–1772.

[85] A. L. Maas, Z. Xie, D. Jurafsky, and A. Y. Ng, "Lexicon-free conversational speech recognition with neural networks", in *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2015.

[86] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio, "End-to-end continuous speech recognition using attention-based recurrent nn: First results", *ArXiv preprint arXiv:1412.1602*, 2014.

[87] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding", in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, IEEE, 2015, pp. 167–174.

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

[89] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition", *ArXiv preprint arXiv:1610.03022*, 2016.