University of Thessaly
Greece, Winter 2020

Audio-Visual Speaker Diarization in Meeting Data

Οπτικο-ακουστική Καταλογοποίηση Ομιλητή σε Δεδομένα Συναντήσεων

Giorgos Gkountouras

Supervisor: Gerasimos Potamianos

Committee Members: Nikolaos Bellas, Georgios Stamoulis

Diploma Thesis

Department of Electrical and Computer Engineering

University of Thesssaly

Volos, Greece

This Thesis is submitted to University of Thessaly as part of the requirements for the Diploma of Electrical and Computer Engineering.

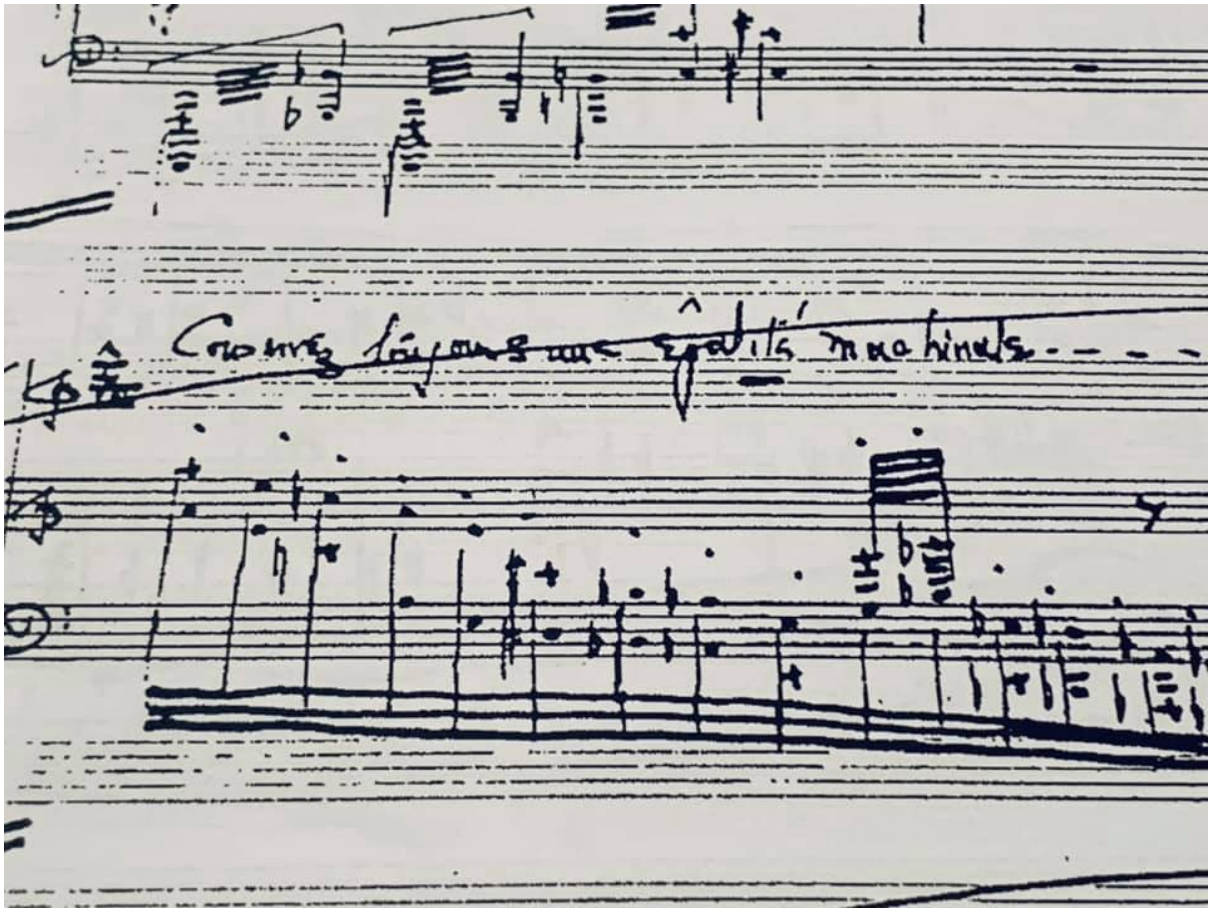


Figure 0.1: Conservez toujours une égalité machinale...

Περίληψη

Ένα ενδιαφέρον πρόβλημα Μηχανικής Μάθησης είναι η καταλογοποίηση ομιλητή. Η καταλογοποίηση απαντάει το ερώτημα του "ποιός μιλάει και πότε " σε δεδομένα ήχου ή βίντεο όταν η διάρκεια ομιλίας και ο αριθμός των ομιλητών δεν είναι γνωστά εκ των προτέρων. Τα τυπικά συστήματα οπτικο-ακουστικής καταλογοποίησης ομιλίας αποτελούνται από πολλά στάδια: πρώτα εξάγονται χρήσιμα χαρακτηριστικά από τα δεδομένα ήχου και εικόνας, στη συνέχεια μια σειρά από βήματα διαχωρισμού κατηγοριοποιούν την ομιλία και τους ομιλητές, και τέλος το πλήρες σύστημα αξιολογείται.

Στην παρούσα διπλωματική εργασία αναπτύξαμε 3 ξεχωριστές μεθόδους καταλογοποίησης ομιλητή. Η πρώτη από αυτές χρησιμοποιεί μόνο δεδομένα ήχου, η δεύτερη χρησιμοποιεί μόνο δεδομένα εικόνας, και η τελευταία χρησιμοποιεί τον συνδυασμό και των δύο. Για την προσέγγιση με τα ηχητικά δεδομένα χρησιμοποίησαμε ενα εργαλείο καταλογοποίσης ομιλητή που αναπτύχθηκε στο Πανεπιστήμιο του Le Mans της Γαλλίας και λέγεται LIUM SpkDiarization toolkit [1]. 13 συντελεστές MFCC χρησιμοποιήθηκαν ως είσοδος για αυτό το σύστημα.

Στην οπτική προσέγγιση, υλοποιήσαμε ένα διαφορετικό σύστημα. Σε αυτό το σύστημα, η περιοχή γύρω από το στόμα επιλέχθηκε ως περιοχή ενδιαφέροντος. Υλοποιήθηκε μια τεχνική οπτικής ροής (optical flow) η οποία ανιχνεύει τις μετατοπίσεις που σημειώνονται στα pixels της περιοχής ενδιαφέροντος από καρέ σε καρέ. Αυτές χρησιμοποιούνται ως οπτικά χαρακτηριστικά του συστήματος. Μία Μηχανή Διανυσματικής Στήριξης (SVM) ταξινομεί τους ομιλητές σε 2 ομάδες, ανάλογα με το αν μιλούν ή όχι.

Το οπτικο-ακουστικό σύστημα καταλογοποίησης ομιλητή συνδύασε και τους δύο τύπους πληροφορίας χρησιμοποιώντας μια μέθοδο πρώιμης ένωσης. Ειδικότερα, η οπτική μέθοδος βελτιώθηκε με την προσθήκη ακουστικής πληροφορίας με τη μορφή συντελεστών MFCC. Έπειτα, ένα άλλο SVM εκπαιδεύτηκε και χρησιμοποιήθηκε για την ανίχνευση ομιλίας.

Όλα τα συστήματα καταλογοποίησης αξιολογήθηκαν μετρώντας το Ρυθμό Σφάλματος Καταλογοποίησης (DER) σε βίντεο που κατασκευάστηκαν ενώνοντας μαζί βίντεο κοντινής λήψης από συναντήσεις, τα οποία προέρχονται από τη συλλογή AMI [2].

Abstract

An interesting Machine Learning problem is speaker diarization. Diarization answers the question of "who is talking and when" in audio or video data when the duration of speech and the number of speakers isn't known in advance. Typical audio-visual speech diarization systems consist of multiple stages: first, the features are extracted from audio and video data, then a series of segmentation steps classifies the speech and speakers, and finally the "end-to-end" system is evaluated.

In this Thesis we have developed 3 distinct speaker diarization methods. One of them uses audio-only data, another utilises video-only data, and the last uses a fusion of both. In the audio-only approach, we employed a speaker diarization toolkit developed by The University of Le Mans, which is based in France. It is called LIUM SpkDiarization toolkit [1]. 13 MFCC coefficients were used as input for this system.

In the visual-only approach, we implemented a different system. In that system, the mouth area was chosen as the region of interest. An optical flow technique was developed that tracks the frame-by-frame pixel displacements in the region of interest. These are used as features of the visual method. An SVM is employed in order to classify the speakers into 2 classes, depending on whether they speak or not.

The audio-visual diarization system combined both types of information using an early fusion approach. Specifically, the visual-only method was augmented with audio information in the form of MFCC features. Then, another SVM was trained and employed to determine the speech status.

All diarization systems were evaluated by measuring the Diarization Error Rate (DER) in multi-camera clips that were constructed by splicing together close-camera meeting videos from the AMI corpus [2].

Acknowledgments

First, I would like to express my deep gratitude to my Principal Thesis Advisor, associate professor Potamianos Gerasimos for his guidance and imparting his invaluable knowledge and expertise during the development of this diploma thesis.

I would like to give special thanks to my fellow friend and colleague Charalampos Vossos for our collaboration and the insights we shared during the past year.

Without the unending support of my family, this work would not have been possible.

Finally, I would also like to express my appreciation for my friends and colleagues at the University of Thessaly for their support, encouragement and advice that helped this effort.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Speaker Diarization Task

The Speaker Diarization task is a common problem in pattern recognition. It has received considerable attention in the literature during the last decades. The problem can be concisely described as finding out who speaks and when in an audio or video stream. In our increasingly data-driven world, diarization is expected to reach commodity hi-tech devices such as mobile phones, laptops and cameras. Thus, we are tasked to develop robust solutions that meet those needs. A typical speaker diarization system based on audio data includes the following steps:

- The audio signal is classified into either speech or non-speech segments.
- Segments of the same speaker are grouped into the same cluster.

It has been observed that it is easier for people to recognize speech when they have visual contact with the speaker. Recently, the same effect was observed in speaker diarization systems. Therefore, various new speaker diarization approaches utilize a combination of both modalities. Indeed, it is expected that audio-only or video-only solutions would struggle with challenging situations.

In particular, audio-only diarization approaches face numerous complications, such as many moving speakers that may occupy different locations, noisy environments and overlapping speech. However, video-only diarization also has problems, considering that

it only works with the information present in the face and especially around the mouth area. This makes it inappropriate for occasions where a high-quality camera to record lip movements isn't available (e.g. mobile phones).

Modern approaches in the diarization space combine both modalities to implement powerful audio-visual systems. These can have the form of early fusion, which is a combination in the feature extraction stage, or late fusion, in which both systems decide separately and then those decisions are combined. In this thesis, we present a method of the early fusion variety.

1.2 Thesis Contribution

The main focus of this thesis is to use audio-visual early fusion methods for the speaker diarization task (finding who speaks and when in a video). First, we explored systems with audio-only or video-only cues. In the audio case, speaker identities were discovered through open-source tools by the University of Le Mans Laboratoire d'Informatique de l'Université du Mans called LIUM SpkDiarization. In the video case, the motion of lips was detected and used as indication of speech. A visual solution utilizing SVMs was developed. As for the audio-visual method, the visual-only approach was augmented with audio information in the form of MFCC features. Finally, all methods we developed were evaluated in multi-panel videos that were created by editing meeting data from the AMI Meeting Corpus [2].

1.3 Related Work

The goal of speaker diarization is to find speech segments and form clusters of them that correspond to the same speaker. This can be achieved in various manners. These include using audio information, visual information, or a combination of both. A typical choice of features in the audio-only case are the Mel Frequency Cepstral Coefficients (MFCCs) in situations where each speech segment belongs to a single speaker. Such a method was implemented in [5]. The authors extracted an MFCC feature vector for each frame,

then used agglomerative clustering, to ensure that each generated cluster belongs to a different speaker. Then, consecutive speech frames were classified either into same speaker segments, or into another speaker cluster via a Hidden Markov Model (HMM).

Visual speech diarization was addressed in [6]. The authors proposed a method for the detection of speech activity in the context of television panel conversations. First, they detected the facial features with a detector based on the Active Shape Model (ASM) [7]. Then, they extracted the facial features using the OpenCV library implementation of the Viola-Jones algorithm. This was followed by facial pose estimation by matching the faces to a pre-trained location model with 68 facial landmarks. The lip region was specifically selected as the region of interest (ROI). They calculated pixel displacement in the ROI with optical flow techniques in order to measure lip motion. Subsequently, they summed the entropy of pixel displacement in each region. The final step was a binary classification into 'speech' and 'non-speech' classes in two different ways. One was a simple thresholding function, and the other involved computing the mean squared difference between pixels in consecutive mouth regions. The introduced method performed better in real data than simply measuring the difference between pixels. Thus, mouth activity was shown to have a positive correlation with the presence of speech.

When it comes to audio-visual diarization, one approach was presented in [8]. Its authors implemented two synchronization methods of audio and visual features. The first was mutual information and the second was Canonical Correlation Analysis (CCA). The results of the two methods were compared separately. They employed MFCCs for audio features. For visual features, they utilized the Kanade–Lucas–Tomasi (KLT) Optical Flow algorithm along with skin color detection techniques to find the lips region motion. Their solution was efficient, since they only used useful parts of the image, rather than the entire image. Following that, they used vertical and horizontal movements and measured their correlation with the audio features. They arrived at the conclusion that audio features are more correlated with the vertical motion of lips. During the evaluation step of their research, they employed mutual information and Canonical Correlation Analysis.

Another multimodal speaker diarization approach on a dataset featuring talk shows was introduced in [9]. The proposed solution was split into feature extraction (audio and visual), model creation, and speech segment classification for the show. The extracted

features in the audio domain were MFCCs. In the visual part, they found that features that characterize the participants' clothing work best. During the training step, they collected shots with large duration that contain faces in the foreground and detected the lip activity in them. Afterwards, the data were clustered in order to have a different speaker for each cluster. Finally, all parts of the talk show not used so far were employed in a classification step involving an SVM classifier. This paper showcased the effectiveness of kernel-based methods in speaker diarization.

Yet another multimodal speaker diarization approach was proposed in [10]. As in this thesis, the training and testing dataset contained meeting videos. In those videos all participants are seen in full body motion and in a non-frontal facial pose. For audio features, they used MFCCs. In the video domain, the features were extracted using the grey-scale difference image algorithm for computational efficiency. In the next step, the audio and visual features were concatenated. Afterwards, an agglomerative clustering algorithm was employed in order to group the different speakers. Lastly, the system was evaluated on meeting data featuring 2 subjects. The method was important because it featured real-world meeting data, which usually contains more side views of the participants than television panels.

1.4 Thesis Overview

This thesis is split into 7 chapters. After this introductory chapter, it has the following structure:

- In **chapter 2**, we describe the meeting dataset we used in detail, the edits we made to create the video clips, and the type of the accompanying transcription files. Additionally, we explain how we parsed the transcriptions to create ground truth annotations.
- In **chapter 3**, we present the audio-only approach. We introduce the LIUM SpkDiarization pipeline. Furthermore, we provide a short description of the extraction of audio features, the probabilistic models used, and the clustering algorithms employed.

- In **chapter 4**, we analyze a video-only approach. We introduce Convolutional Neural Networks (CNNs). Then, we present a meticulous description of our method (face detection through the dlib library, cloud deployment, and optical flow). Finally, we describe the steps involved in the implemented diarization system.
- In **chapter 5**, we introduce our main contribution, the combination of audio and visual modalities. To that end, we augment the visual-only approach with MFCC information from the audio domain to improve the detection of short speech segments.
- In **chapter 6**, we present the results of all 3 implemented diarization methods, compare and comment on the results.
- In **chapter 7**, we conclude with a summary of our findings. We also provide some ideas for future work on speaker diarization.

Chapter 2

Dataset

2.1 Dataset Description

2.1.1 Overview

For the training and evaluation steps of this thesis we used the AMI Meeting Corpus [2]. This consists of 100 hours of videos of English meetings. 11 video clips with a total duration of 1 hour and 24 minutes were extracted from 6 of these videos. Each clip is in a Matroska video container. The video is encoded as MPEG-4 and the audio is encoded as MPEG-4 AAC at 48000 samples per second (48 kHz). All video clips have 30 frames per second (FPS) and a resolution of 1280×720. Their duration ranges from 3 to 10 minutes. All clips are continuous parts of the main meeting data. Note that the dataset contains multiple video streams and multiple audio streams (see Section 2.1.3). Stream selection is covered in detail separately (Section 2.1.5). The videos were edited using Openshot and cut using avidemux. Both programs are free, open source video editors.

2.1.2 Meeting description

Each meeting involves four participants around a table that speak in turns. It is recorded in English and includes mostly non-native speakers. An additional complication for any diarization attempt is overlapping speech between different participants in some occasions.

There might also be silent segments. There are several meeting scenarios (e.g. designing a product, selecting a movie). The participants play the different roles of a design team or take parts in a naturally occurring scenario of some domain. Each meeting begins with a short introduction of the participants and preparation of their lapel microphones. Afterwards, the meeting is conducted with the participants discussing the topics according to their roles. Note that earlier topics may be repeated as new information arises in a natural fashion. The meeting ends with a conclusion and removing the recording devices.

2.1.3 Streams and additional files

Multiple video and audio streams are present in the corpus. The video streams include a close camera for each of the four participants (Figure 2.1a), plus additional cameras depending on the scenario. The close camera focuses on their face, but it shows their entire upper body. The audio streams include far-talking microphones (2 microphone array streams of 8 microphones each) and close-talking microphones worn by each participant (lapel and headset). There is also a mix of the headset recordings and a mix of the lapel recordings. The microphone arrays are placed on the table around which the meeting is conducted. Additionally, the dataset includes written notes by the participants, projection slides used to aid the discussion, data from a whiteboard used for sketches and short notes, and shared computer files (spreadsheets, e-mails, documents, databases).

2.1.4 Transcriptions

The AMI corpus includes automatic as well as high quality, manual, orthographic transcriptions of the dialogue in the meetings. It also contains annotations for various events such as dialog acts and head movement. The manual transcriptions were transformed to speaker annotations and used for the system's training and the evaluation of its performance (Section 2.2).

2.1.5 Stream and file selection

In this thesis, the four close camera streams were stitched into one video, resembling the arrangement of broadcast news videos (Figure 2.1b). Audio from the first microphone of the first microphone array was initially used. However, we later discovered that individual face cameras and personal headset microphones performed better for our use case (Section 3.4.3). No other video streams, audio streams or additional files were used, with the exception of transcriptions, during the evaluation stage.

2.1.6 Clip selection

The corpus includes many videos, but only some clips of them were used for the purposes of this thesis. The clips were chosen to be frontal face views from the close cameras. Participants are allowed to turn sideways toward other participants while they speak, look at the slides as they discuss them, move their hands as they speak, approach the camera or move away from it, cover their mouth with their hands while speaking, take written notes with their pens, use their laptops, play with their pens, or, in one case, chew gum. This allows assessment of the algorithm's performance in a realistic environment. In contrast, clips where the participant gets up and turns toward the whiteboard as they present or leave the room while speaking are considered outside the scope of this thesis, while presenting an interesting challenge for later work involving location of the participants.

2.1.7 Video scenarios

These are the videos used, followed by a short description of the corresponding scenario.

- **ES2002a** - Design of a remote control. The participants go over the project's stated goals, draw on the whiteboard for the remote's design, raise their concerns over possible issues with the project's goals and come up with initial ideas for the remote control's design.
- **ES2002c** - Design of a remote control. The participants raise concerns over equipment issues, watch the trends of the European market, discuss the proposed

features of the remote control and evaluate other existing products in the market. This is followed by a presentation on industrial design, an in-depth discussion of the remote's components, materials and the power source, decisions about its conceptual design and finally a discussion of the jog dial as a user interface.

- **IB4010** - Selection of a movie. The participants judge the movie options presented based on many criteria, including their plot, cast, soundtrack, genre, popularity, directors, length, posters and awards received.
- **IS1009b** - Design of a remote control. The participants discuss the product's marketing strategy, its functionality and industrial design considerations. They suggest designs, go over the components of the remote and its user interface, focus on the target group for remotes and present alternative designs.
- **IS1009c** - Design of a remote control. The participants suggest the option of a speech recognition interface for the device as well as other user interface options. They go over its basic functionality, its components, industrial design and a marketing presentation. They conclude with decisions on how to proceed.
- **IS1009d** - Design of a remote control. The participants present a prototype, describe its look and feel and list its features. Consequently, they evaluate the prototype, and talk about the meeting's agenda. Finally, they give an estimate of the product's cost and the cost of adding speech recognition to it, rate the prototype's properties and evaluate the entire project.

Note that in the product design videos the participants are role-playing. They are assigned the roles of a fictitious design team (industrial designer, interface designer, marketing, or project manager). In contrast, the movie selection video involves a naturally occurring meeting. As we've mentioned previously, continuous clips are utilized rather than the entire videos mentioned here. More than one of those clips may belong to the same video.

2.1.8 Additional cameras for each scenario

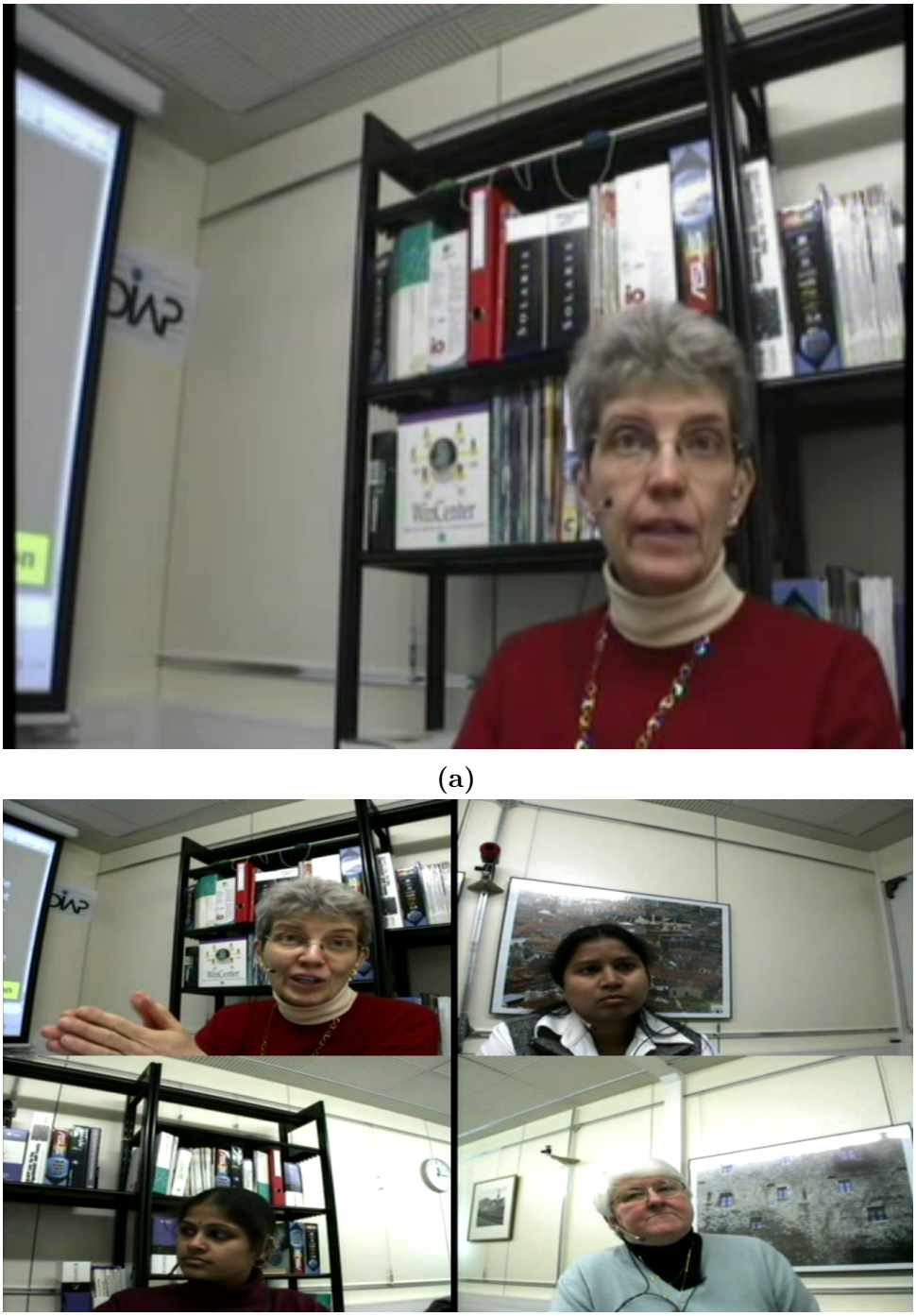
Each scenario has cameras specific to that scenario:

- **ES2002a, ES2002c** - An overhead camera with a top-down view of the table and

the participants, and a corner camera with a view of the room and its participants.

- **IB4010, IS1009b, IS1009c, IS1009d** - Left and right cameras, each focusing on two of the four participants, as well as a center camera showing a side view of all four participants and the projected slides.

As mentioned before, these cameras are not used for this thesis.



(a)

(b)

Figure 2.1: Two cases of video frames. In (a) there is a frame from the personal camera of a single participant and in (b) a frame from the edited video with all four personal cameras.

2.2 Transcription Files

Each of the 6 meeting videos used to create the clips corresponds to an XML transcription file. This file contains the topics of discussion, as they come up naturally during the meetings. The beginning of this file holds a header with metadata, including encoding and XML version. In our case the encoding is ISO-8859-1 (Latin, ASCII-based). Each topic is split into turns of speech. Each turn has a speaker, the first word's identifier and the last word's identifier. If there is just one word in that speaker's turn, only that word's identifier is included instead.

The word identifiers are references to different files containing the words themselves. Each of the 6 videos has 4 XML word files, corresponding to the 4 meeting participants in the video. Similarly to the transcription file, the beginning of a word file contains metadata. It is also encoded as ISO-8859-1. The word file for a speaker includes the words spoken by that speaker, along with their unique identifiers, their start times and their end times. It also includes information about whether the transcribed word is a punctuation mark, which we did not utilize.

2.3 Ground Truth

In a supervised machine learning system training is accomplished via target outputs that are known to be correct. These targets are known as the ground truth of the system. We expect a machine learning system trained on the ground truth to be able to generalize its insights to real data it has never seen before. Importantly, ground truth labels are also used for the evaluation of the system. We can compare its predictions to the ground truth and measure the error as the difference.

In our case, the files belonged to the XML document type. XML is a hierarchical format that can be parsed to a tree data structure using programming language utilities. Thus, it is possible to cross-reference a topic file and its corresponding word files (see Section 2.2). For that purpose, we created a parser in python. The parser outputs a reference speech file. Each line of that file contains a turn, with its start time, its end time and

the participant that was speaking during the turn. Note that for each clip, we only parse from the start time to the end time of that clip. In case of overlapping speech, we can only find that the turn times overlap after this parsing step. It is not obvious from the topic file alone.

Special care must be taken to ensure that the ground truth (training targets) is in a suitable format for the supervised learning algorithm to learn from. In this thesis, we have chosen binary labels that indicate the presence or lack of speech for the corresponding participant in a frame. In order to create the target data, we used the parsed values from the transcriptions (start time, end time, frames, speaker name) in the following manner:

- A numpy vector is initialized with zeros. This vector has a size equal to the number of speakers multiplied by the total number of video frames.
- For each frame where the transcription indicates speech, we set the speaker's target label for the current frame to 1.

Obviously, in overlapping speech cases we set the target values to 1 for all overlapping speakers in the frame.

2.4 Reference Annotations

Subsequently, we converted the values from the reference speech file (speech start, speaker end, speaker id) into an annotation (see Figure 2.2) by assigning the speaker id to the corresponding time of speech. The annotation was created using the pyannote python library. The speech segment had to be converted into an appropriate pyannote.core.Segment class in order to be inserted into the annotation. This describes a segment with the following form:

[00:00:00.000 – 00:01:30.425]

The time is formatted as hours:minutes:seconds.milliseconds.

Afterwards, we created the reference annotations. Each annotation is represented by the

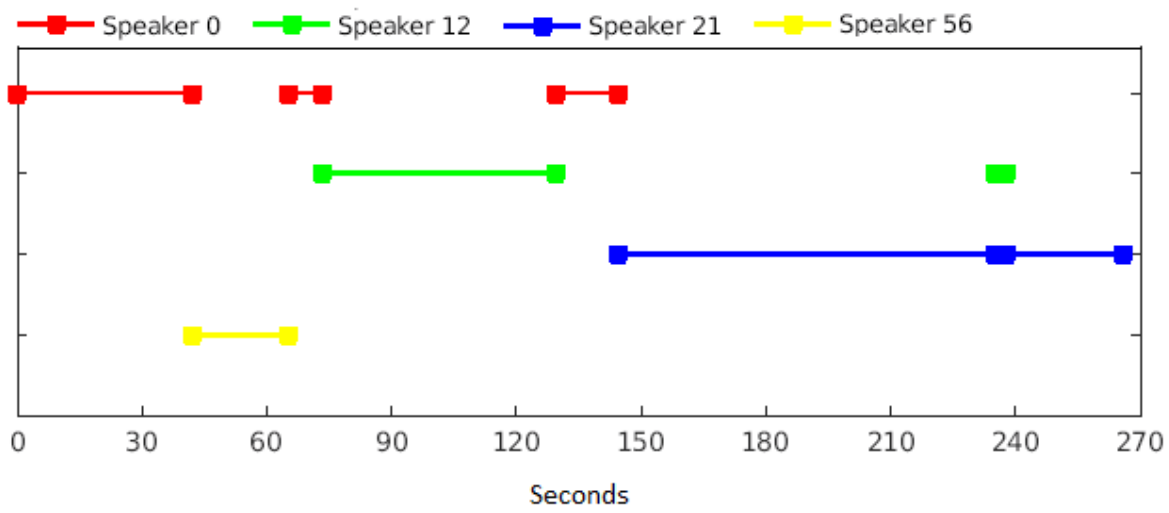


Figure 2.2: A visualization of an annotation

pyannote.core.Annotation class. That class has the following features:

- The segments are sorted by start time or, if tied, by end time.
- It's impossible to add the same track twice.
- Only non-empty tracks can be inserted.

A track is a combination of *support* and *name*, where support is the speech segment represented by start time and end time) and name is the speaker's id. Note that many different speaker ids may correspond to the same support in cases of overlapping speech segments. This is an example of a partial reference annotation file:

```
[ 00:00:00.000 – 00:00:41.847 ] - spk0
[ 00:00:41.847 – 00:01:05.100 ] - spk56
[ 00:01:05.100 – 00:01:13.041 ] - spk0
[ 00:01:13.041 – 00:02:09.172 ] - spk12
[ 00:02:09.172 – 00:02:24.267 ] - spk0
[ 00:02:24.267 – 00:03:54.692 ] - spk21
[ 00:03:54.692 – 00:03:57.578 ] 0 spk21
[ 00:03:54.692 – 00:03:57.578 ] 1 spk12
[ 00:03:57.578 – 00:04:25.656 ] - spk21
```

The reference annotation enables us to evaluate the system's performance by comparing it to the hypothesis annotation. The hypothesis annotation is produced by our diarization

system. Both annotations were inserted into a metrics tool in order to measure the effectiveness of our system. In this thesis, the Diarization Error Rate (DER) metric, as computed by the 'pyannote metrics' python library, was used.

Chapter 3

Audio Speaker Diarization

This chapter introduces our first diarization solution, which is only based on audio data. Our method uses the LIUM speaker diarization toolkit, with additional pre-processing to address the particulars of the AMI Meeting Corpus dataset.

We begin with a high-level description of the toolkit and the specific problem it solves. This is followed by an enumeration of the steps in the LIUM toolkit pipeline, focusing mainly on the extraction of audio features. The later stages of the framework are presented in summary. Complete mathematical expressions can be found in the literature [1, 11].

Moreover, we cover the creation of the audio-specific dataset from the AMI Corpus. Specifically, we include details on correcting the 3-way mismatch between audio clips, video clips and manual transcriptions. Finally, we provide a description of the pre-processing stage. This step is necessary, as the framework's machine learning models have been trained on a broadcast news dataset, whereas we use them in the context of meetings.

3.1 LIUM Speaker Diarization Toolkit

LIUM SpkDiarization ([1], with later improvements in [11]) is a framework for speaker diarization developed by the Laboratoire d'Informatique de l'Université du Mans. It is free, open-source and written in the Java programming language. In addition to uncovering the identities of the speakers and the duration of their speech, the toolkit can also find

the speaker's gender, the channel type (narrow or wide bandwidth), as well as detect the presence of music. Since the default configuration is tuned to discover a single speaker in a single channel, we edited our dataset to match that description (Section 3.4.3).

At the time of its publication, LIUM SpkDiarization possessed state-of-the-art results for the diarization task in the domain of broadcast news, as evidenced in the ESTER2 evaluation campaign [12]. However, the authors of the framework caution that while it is optimized for television and radio shows, its performance degrades when evaluated on phone conversations and meetings. This is in agreement with our findings when we adopted LIUM SpkDiarization in the context of meetings for this thesis. We discovered that additional pre-processing (Section 3.4.6) was required in order to maintain acceptable error rates. We should also note that SpkDiarization was trained by minimizing the Cross-Likelihood Ratio (CLR) and normalized Cross-Likelihood Ratio (NCLR) metrics [13, 14], whereas we evaluated our solution with the DER metric.

3.2 MFCC Features Extraction

3.2.1 Sphinx 4 MFCC module

The first stage of LIUM's diarization pipeline computes feature vectors from the audio waveform. To that end, it employs the open-source Sphinx 4 tools, which were developed by Carnegie Mellon University. This is acknowledged in the overview of the LIUM diarization effort after the ESTER2 campaign [15]. LIUM SpkDiarization uses Mel Frequency Cepstral Coefficients (MFCCs) as audio features, a common input for diarization and general audio processing work [16, 17, 18].

3.2.2 Human voice properties

As the name implies, the MFCC uses the Mel scale to divide the frequency spectrum into spectral sub-bands. The Mel scale is optimized for the way frequencies are perceived by the human ear. While humans can hear from 20Hz to about 20kHz, depending on their age, their ability to distinguish between close frequencies is not linear. It is easier to distinguish

between lower frequency tones (e.g. 150Hz and 300Hz), than higher frequency tones (e.g. 16kHz and 16.1kHz). The adult human voice fundamental tones are in the range of 85Hz to 255Hz (85Hz to 180Hz for male and 165Hz to 255Hz for female). Harmonics are integral multiples of the fundamental frequency. For example, a fundamental frequency of 150Hz has a 2nd harmonic at 300Hz and a 3rd harmonic at 450Hz. In 1937, Stevens, Volkmann and Newmann proposed the Mel scale, featuring equidistant pitches as interpreted by subjective listening tests.

3.2.3 MFCC computation steps

MFCC feature extraction can be subdivided in the following steps:

- **Audio Framing**: Audio is a non-stationary process, however we can assume that the audio signal is stationary for relatively small durations. We split the signal into small segments, known as frames. Audio frames overlap at the edges. This counteracts the effect of windowing, which can lose information. We set the frame window to 25ms, the hop size to 10ms and the overlap to 15ms for the Sphinx4 toolkit at a sample rate of 16kHz, per LIUM's recommendations. Therefore, window length can be calculated as 0.025 * 16000 = 400 samples.

- **Discrete Fourier Transform (DFT)**: Next, we convert each frame from the time domain to the frequency domain using the DFT. In practice, we prefer the Fast Fourier Transform (FFT) algorithm, which computes the DFT more efficiently ($O(N \log_2 N)$ calculations) than the naive approach ($O(N^2)$ calculations). We apply a Hamming window function to each frame to prevent high frequency distortions. This ensures that the start and end of the frame are near zero before the FFT. Note that shorter frames produce fewer frequency bands in the FFT and lower ability to discriminate between close frequencies, while in longer frames the frequency composition varies too much within the window.

- **Mel filterbank**: Afterwards, we apply the Mel filterbank to the frequency spectra. The FFT bins inside a region are isolated with a triangular filter and summed to calculate the energy of that frequency range. The filters are wider at higher frequencies and their center frequencies are spaced according to the Mel scale.

This reflects the human auditory system's inability to distinguish between higher frequencies that are closely spaced.

- **Logarithm of filterbank energies**: In the next step, we calculate the logarithm of the energy in each filterbank region. As with region spacing, this is motivated by the human auditory system: loudness is not perceived linearly, but in a logarithmic fashion. Large variations in energy could lead to small perceptual changes. While using logarithms enables us to utilize Cepstral Mean Subtraction (CMS) in a normalization step, we do not employ it in this project.

- **Discrete Cosine Transform (DCT)**: Finally, we calculate the DCT of the log-filterbank energies. The energies of the filterbanks are correlated, since they are overlapping. Including a DCT step can help decorrelate them. Diagonal covariance matrices can be used to model the features in an HMM classifier. Another use of the DCT is feature dimensionality reduction by dropping its higher coefficients (faster changes). However, no such step happens in this project.

- **Deltas calculation**: Deltas, also known as velocity coefficients, are the first derivatives of the MFCCs. Accordingly, delta-deltas, or acceleration coefficients, are the second derivatives. As energies in the power spectrum of each frame change over time, it is expected that this change will reflect additional information hidden in the signal. If they are used, computed deltas and delta-deltas are appended to the MFCCs to create a complete audio discriminating feature vector.

The first 3 stages of the LIUM toolkit (segmentation based on the Bayesian Information Criterion (BIC) [19], BIC clustering and segmentation based on Viterbi decoding) use 13 MFCCs as features. The first coefficient, C_0, is acting as energy. No features are normalized via CMS or warping. Additionally, neither first nor second derivatives are used for these stages.

3.3 Main Components of the LIUM Toolkit

With the exception of feature extraction as described in the previous section, we only give an outline of the system introduced in [1]. The stages involved are:

- **Generalized Likelihood Ratio (GLR) Segmentation**: The audio is split into separate segments, each assumed to have exactly one speaker. The first segmentation pass is based on GLR with full covariance Gaussians calculated over a 5-second sliding window. GLR measures the distance between adjacent speech segments. When it peaks, the algorithm splits the audio into two segments at the middle of the window.

- **BIC Segmentation**: A second segmentation pass fuses consecutive segments with the same speaker. It is based on BIC distances [19] between speakers. As in the first step, full covariance Gaussians are employed. Unlike [19], LIUM SpkDiarization only uses the lengths n_i and n_j of the candidate clusters to merge as a penalty factor. Experimentally [20, 21], this change produces better results than penalizing the entire data length N.

- **Hierarchical Agglomerative Clustering**: It is evident that speech segments can belong to the same speaker without being consecutive. This step models speakers with full covariance Gaussians. Initially, each segment is assigned to a different cluster. At each iteration, the two closest clusters, i and j, are merged. This is repeated until $\Delta BIC_{i,j}$ [22] is positive.

- **Viterbi Decoding**: Next, a Viterbi decoding step produces a new segmentation. Note that the features used are 12 of the MFCCs with deltas. Energy C_0 is dropped. Each cluster is modeled by a Hidden Markov Model (HMM) with one hidden state and the HMM log-penalty set in an experimental manner. The state is in turn modeled by a Gaussian Mixture Model (GMM) featuring 8 components and diagonal covariance. The GMM is trained using the Expectation-Maximization algorithm (EM), trained on the output of the previous step.

- **Speech/Music/Silence Segmentation and Filtering**: As before, C_0 is removed. Only 12 MFCCs and their first derivatives are employed. A second Viterbi decoding splits segments into two distinct classes, speech/non-speech. This step utilizes 8 HMMs: 2 silence models (narrowband and wideband), 3 models featuring wideband speech (clean, speech with noise, speech over music), 1 narrowband speech model, 1 model of jingles and 1 music model. Jingle and music models were developed to address radio and television shows. They don't aid in our meeting scenarios. Each

HMM has 1 hidden state, modeled by a GMM with 64 Gaussians. All Gaussians have a diagonal covariance matrix. Another point of note: LIUM SpkDiarization's authors moved this step after speaker segmentation after they noticed that speaker segmentation was negatively affected by the processing. Finally, segments without speech are removed (segmentation filtering). The segment padding and the minimum length for silence segments are both set equal to 1 MFCC window (25 ms). The minimum speech segment length is set to 150 ms.

- **Gender and Bandwidth Detection**: This stage also uses 12 MFCCs alongside deltas and no C_0. It predicts gender (male or female) and bandwidth (narrowband or wideband) for a total of 4 combinations. Therefore, a GMM is employed to predict each corresponding class (4 GMMs total). Each GMM is pre-trained with the ESTER dataset [12] and features a diagonal covariance and 128 Gaussians. A label is predicted for each cluster such that the likelihood of the cluster's features is maximized. Additionally, features are warped with a 3-second window to fix variations, as recommended in [23] and normalized through centering and reduction.

- **Integer Linear Programming Clustering**: The last stage uses 12 MFCCs with first order derivatives and no energy. These features are augmented with information about speaker gender and speech bandwidth from the previous stage. The clustering method of [24] is adopted to frame the clustering as an Integer Linear Programming problem, which is then solved. In contrast, the first version of SpkDiarization [1] used a Universal Background Model (UBM), with clustering based on the Cross Likelihood Ratio (CLR) and the Cross Entropy in addition to Normalized CLR (CE/NCLR). In ILP clustering, i-vectors [25], which are low-dimensional representations of the features that retain critical details, are utilized as a measure of cluster similarity. In the original task that LIUM SpkDiarization addresses (radio and broadcast shows) the number of speakers is generally unknown. After a first speaker segmentation, each cluster is converted to an i-vector using 12 MFCCs without energy or delta information and a GMM-UBM with 1024 components. In the following step, the N i-vectors are normalized via an iterative process [26]. A final clustering step attempts to simultaneously minimize the number K of resulting clusters and the intra-cluster distribution. A complete mathematical derivation of the ILP equations

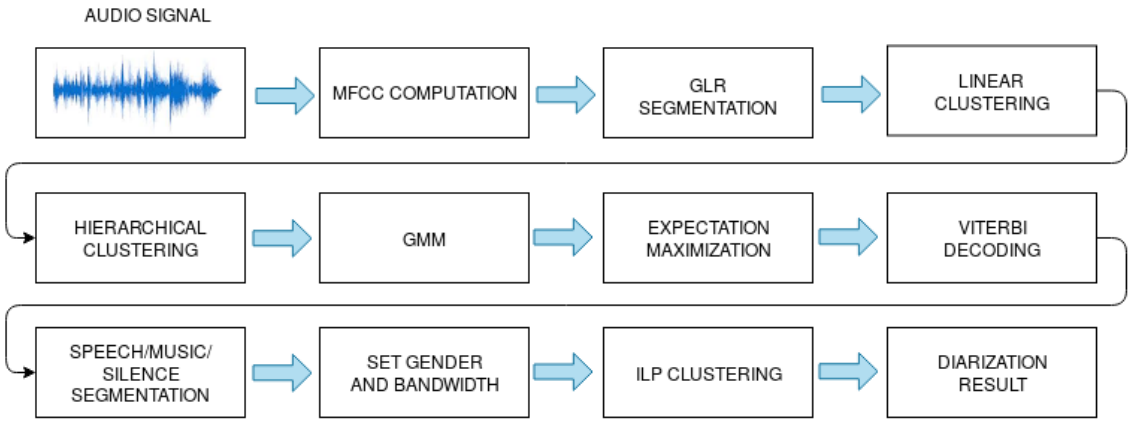


Figure 3.1: The complete audio diarization method of LIUM

can be found in [11].

The complete diarization process of LIUM SpkDiarization is illustrated in Figure 3.1.

3.4 Audio Dataset

3.4.1 File formats

The original dataset included multiple data streams of 4-person meetings. The audio dataset was created from AVI files with each individual speaker. The properties of the original video file type in the AMI Meeting Corpus were the following:

- **Container** - AVI (Audio Video Interleave)
- **Dimensions** - 352 × 288 pixels
- **Codec** - DivX MPEG-4 Version 4
- **Framerate** - 25 frames per second
- **Bitrate** - 155 kbps

The video files were combined with audio files to create video files with audio. The original audio files in the AMI Meeting Corpus had the following properties:

- **Container** - WAV (Waveform Audio File)
- **Codec** - WAV
- **Channels** - Mono
- **Sample rate** - 16000 Hz
- **Bitrate** - 256 kbps

3.4.2 Audio stream selection

Our original system arranged the 4 speakers in 4 windows, one at each corner of the screen. This was an attempt to recreate the configuration of an earlier successful broadcast news diarization project [27]. The AMI Meeting Corpus dataset included an array of 8 microphones placed at the center of the meeting room. The orientations of the microphones were equally spaced (0 degrees, 45 degrees, 90 degrees and so on). The audio recordings utilized for this stage were captured from the first microphone from the array. We speculated that one microphone would capture all speakers in the room and treated this stream as a room microphone.

Unfortunately, while the room microphone captured all speakers adequately, we discovered that the overlap in speech resulted in inconclusive diarization output. Moreover, the microphone captured a lot of ambient noise in the room, which further muddied input to the LIUM algorithm. Another possible reason for the unfavorable results is that the LIUM framework was trained on broadcast news datasets. It is not expected to generalize to realistic meeting scenarios. Finally, the microphone is directional. Quality from speakers behind the microphone's direction is lower, since it can only capture reflected sound as it bounces around the room, instead of the original audio. We postulate that this mostly affects higher frequencies, that are absorbed more than lower frequencies, in addition to being reflected. Those higher frequencies are critical when it comes to speech recognition.

Note that only 1 out of 8 streams was used as a room microphone. Our experiments showed that which microphone gets chosen doesn't drastically change the results. Therefore, the first microphone was chosen for simplicity. An interesting idea was mixing the streams (linear addition) or combining them in a similar manner. This prospect was abandoned,

as the distance from each of the 8 microphones to each of the 4 speakers is different. Even if we consider the room's frequency response as linear (an assumption that isn't far from the truth), we would still have the nontrivial problem of lining up the $4 \times 8 = 32$ different delays. However, this very echolocation provides an interesting future research area.

3.4.3 Individual speaker videos

The lack of results from using room microphone audio streams indicated the need for a new approach. We observed that the dataset had different video and audio streams for each speaker (Section 2.1.3). Consequently, these streams could be employed to conduct diarization separately for each speaker. The 4 resulting diarizations could then be combined in a late fusion stage to produce the complete diarization output of the system.

This increase in pipeline modularity offers numerous benefits. Firstly, the issue of having 8 room microphones with no easy way to combine them due to speaker delays is resolved gracefully. This comes from the insight that we can use different audio files from the dataset that more closely match the nature of 4 close-up videos of speakers. Audio from the personal microphone of each speaker is used instead (Section 2.1.5). Secondly, the change permits piecewise pre-processing and post-processing of individual clips, both in the video and in the audio modalities. This adds flexibility to the development of the final method. Finally, the problem involving ambient room noise disappears for the most part via a careful selection of microphones.

We shall expand further about this selection. The dataset included 2 candidate microphone positions:

- **Headset** - Audio from a headset worn by a speaker
- **Lapel** - Audio from a lapel microphone worn by a speaker

We chose the first of those 2, speculating that headset microphones produce less noise. A lapel microphone is more likely to pick up rustling sounds as the speaker moves, whereas a headset microphone is closer to the speaker's mouth and has a higher signal-to-noise ratio. We noted that the lapel microphones might be utilized in addition or instead of the headset microphones, with proper pre-processing to remove noise, ambient or otherwise.

Our first efforts focused on producing a mix of the 4 resulting headset microphones and evaluating the performance of LIUM diarization on it. On one hand, the amounts of ambient room noise that close microphones picked up was insignificant. On the other hand, this did not resolve the issue of speech overlap. In particular, the headset microphones were sensitive enough to pick up the other speakers in the room. This resulted in a muddy headset mix of $4 \times 4 = 16$ speakers from 4 microphones. LIUM produced mediocre diarizations, failing to reconstruct the speakers' identities (or even their number) correctly.

Therefore, a decision was made to utilize the microphone of each speaker separately. Other speakers in the recording are removed via pre-processing the file with an audio gate (Section 3.4.6). However, this approach is not without its disadvantages. Chief among them is an increase in the complexity of the resulting pipeline. There are numerous files to process and many parameters to fine-tune. A careful combination of manual and automatic hyperparameter search was required to produce a working audio method.

3.4.4 Audio, video and transcription match

In order to suitably compare the audio and visual methods, the duration of the video clips and the corresponding audio clips should match. The same is required for early fusion of the 2 methods into an audiovisual method. An exact match isn't required. Indeed, either file can be longer by a few milliseconds. If the audio file is longer, then it can be trimmed. If it is shorter, it can be padded with silence to compensate. Thus, our method ignores such length differences.

Videos without audio in mp4 format were combined with headset audio in WAV format using the avidemux editor. The properties of those files were defined in Section 3.4.1. This resulted in mp4 files with the following properties:

- **Container** - Quicktime
- **Video Dimensions** - 1280 × 720 pixels
- **Video Codec** - H.264 (High Profile)
- **Framerate** - 30 frames per second
- **Video Bitrate** - 13758 kbps

- **Audio Codec** - MPEG-4 AAC
- **Channels** - Stereo
- **Sample rate** - 48000 Hz
- **Audio Bitrate** - 192 kbps

Subsequently, the videos with audio were cut to the correct duration with the Openshot editor to create video clips with audio in mkv format. This resulted in the files with the following properties:

- **Container** - Matroska
- **Video Dimensions** - 1280 × 720 pixels
- **Video Codec** - H.264 (High Profile)
- **Framerate** - 30 frames per second
- **Video Bitrate** - N/A
- **Audio Codec** - MPEG-4 AAC
- **Channels** - Stereo
- **Sample rate** - 48000 Hz
- **Audio Bitrate** - N/A

The audio from these clips was extracted using the VLC player as an offline converter. The resulting files from this process had the following properties:

- **Container** - WAV (Waveform Audio File)
- **Codec** - WAV
- **Channels** - Stereo
- **Sample rate** - 44100 Hz
- **Bitrate** - 1411 kbps

After the previous steps, the audio and video clips were a temporal match. However, careful manual observation of the resulting files revealed a different type of mismatch. The

audio, video and transcription files did not always correspond to the speakers indicated by their names. This 3-way mismatch was a dataset error, not part of the problem's description. Thus, it was resolved by manual inspection of each clip and transcription and manual renaming of the corresponding files.

Specifically, these mismatches were corrected:

- **Video ES2002a, clip 1** - Videos 1-2-3-4 → Headsets 1-3-4-2 → Transcripts 1-3-4-2
- **Video ES2002c, clip 1** - Videos 1-2-3-4 → Headsets 1-3-4-2 → Transcripts 1-3-4-2
- **Video ES2002c, clip 2** - Videos 1-2-3-4 → Headsets 3-1-4-2 → Transcripts 3-1-4-2

Note that in all cases of mismatch, the transcriptions match the headset audio files. Both of them were changed to match the speakers in the video clips, which were treated as a reference. Another point of interest is that the speakers change their positions between clips 1 and 2 of video ES2002c. The headset audio files and the transcriptions files don't change, resulting in different, yet also mismatching, corresponding video clips. This is not noticeable above, since the speakers, as they appear in the video clips, are considered the single source of truth. Therefore, it appears as if the headset audio clips and the transcriptions change.

3.4.5 Transcription parsing script

Ground truth files are needed for the evaluation of our pipeline. The manual transcriptions in the AMI Meeting Corpus were used for this purpose. A special script was created to parse the transcriptions. The files containing the speakers' words had 3 different kinds of interjections:

- **vocalsound**
- **disfmarker**
- **gap**

Examples of vocalsound interjections include the speaker laughing or coughing. Vocalsound interjections could appear in a segment with a single word. In addition, they could be in the middle of a speech segment, but without a duration of their own, thus not affecting the

durations in the transcription. Zero-duration vocalsound interjections could also appear at the beginning of a speech segment, also not affecting the transcriptions. Finally, 2 vocalsound interjections could appear in a row. As an example, 2 vocalsound interjections at the start of a speech segment had the unfortunate consequence of classifying the empty space between them as speech.

A disfmarker (discontinuity marker) interjection indicates that the utterance wasn't completed. It can also be used to identify a grammatically correct sentence when the speaker's intonation suggests that there might have been more to follow. 2 disfmarker interjections could appear in a sequence, but never belonging to the same speaker. Words with the disfmarker characterization had no special meaning, so they were ignored by the parsing script. Gap interjections were also removed, since they appeared at the middle of speech and they didn't affect its duration. Vocalsound interjections were too complicated to treat in an automatic fashion. Therefore, they were manually changed after careful inspection to closely match the videos.

At this point, we should note that the topic files don't include the speech segments in chronological order. Furthermore, the topics include sub-topics recursively, according to the hierarchical nature of XML files. Our transcription parser handles these situations gracefully.

3.4.6 Pre-processing via audio gate

As alluded to in Section 3.4.3, audio clips from headset microphones are chosen because they contain clear audio by the speaker. However, a new problem arises from this choice. The headset microphone that belongs to a speaker picks up speech from other speakers in the room.

This is analogous to crossfeed in a stereo audio processing system. In crossfeed, audio from the left output is mixed back into the right channel, and audio from the right output is mixed back into the left channel. Crossfeed in the meeting headset microphones is even worse, since speech from each of the 4 speakers is mixed into each of the 4 microphones. This results in a total of $4 \times 4 = 16$ speakers in the 4 audio clips.

In order to resolve this issue, we added a separate pre-processing module to suppress

crossfeed. The module consists of an audio gate. Specifically, we selected ReaGate [28], a freely available audio processing effect plug-in. Audio gates silence audio when it falls below a predefined threshold. We speculated that the crossfeed audio would be at a lower volume than that of a speaker talking into the own headset microphone. Therefore, by setting the threshold appropriately, we could remove the other speakers in the room from the audio clip without affecting the main speaker associated with that microphone. We experimented with 3 parameters in the gates we used:

- **threshold** - When the input's amplitude falls below this level, the output drops to 0.
- **attack** - The time it takes from the audio falling below the threshold to the gate silencing the audio. Note that the audio starts diminishing in amplitude immediately after it crosses the threshold, not after the attack time is over.
- **release** - The time it takes for the output to recover after the input rises above the threshold again.

After manually experimenting with various combinations of these parameters, it became obvious that attack time can be too fast for effective diarization. In effect, an attack time of **3 ms** produces an audible distortion, as the audio goes silent almost instantly. Accordingly, the speaker segmentation module of LIUM diarization fails to categorize the speakers accurately. Our manual experiments included 4 other attack times:

- **10 ms**
- **15 ms**
- **20 ms**
- **25 ms**

The first 2 of the above produced equally satisfactory outcomes, with the others trailing in accuracy. We selected **10 ms** for this hyperparameter.

The release hyperparameter was left to the default setting of **100 ms**. According to manual tests, it was not as critical as attack time for the speaker segmentation step.

The threshold hyperparameter proved more challenging to configure. Among the main

concerns was the major difference in audio levels between videos. In particular, clips that were created from the following videos had much lower loudness (and average amplitude) levels than the other audio clips:

- **ES2002a**
- **ES2002c**

While lower audio levels are not by themselves troubling for the LIUM pipeline, they make it impossible to select a common threshold for the gates that would work for all audio clips. On one hand, setting the threshold too high would mute all audio in the quiet clips. On the other hand, setting it too low would not prevent crossfeed in the loud clips.

We wanted our method to be effective without any assumptions about audio clip levels. In order to achieve that, we prepended the gate module with a step that automatically determines the ideal gate threshold for each audio clip separately. This step utilizes polynomial regression to predict the thresholds for the gates that would result in lower diarization error rates. Linear regression was also attempted, but proved insufficient for robust threshold prediction. The parameters used for the polynomial regression are:

- **Integrated loudness**
- **Loudness range**
- **True peak**
- **Maximum short-term loudness**
- **Maximum momentary loudness**

Naturally, these predictors are not independent. Indeed, we should expect them not to be. Clips can be quiet in the short-term as well as in their entirety. Conversely, they can be loud both short-term and long-term. In other words, there is correlation between the features. If required, this can be fixed with an orthogonalization step. Our tests indicated that such a measure was not needed.

An interesting subject is whether the polynomial regression method for automated hyperparameter search is expected to generalize for the diarization task in other settings. We posit that it will produce accurate results, especially for other meetings in a company

setting. However, further research is required in order to evaluate it in other contexts, such as broadcast news.

3.4.7 Audio diarization testing

For each audio clip in the dataset, we applied gate pre-processing with the predicted threshold (Section 3.4.6) to remove other speakers. Then, we used LIUM diarization to find speech (or lack thereof) for each speaker individually. The 4 predictions for each video clip (one per speaker) were concatenated into a complete hypothesis annotation for the clip. Finally, this annotation was compared to the manual transcription in the dataset, which was acting as a reference annotation, to measure the diarization error rate (Section 6.2).

Chapter 4

Visual Speaker Diarization

This chapter introduces our second diarization solution, which is based solely on video data. First, we cover the theoretical background of the dlib library and explain how it enables face recognition. Then, we describe our experimentally derived technique for detecting the mouth area. Afterwards, we focus on scaling the mouth area for consistency across frames.

In addition, we present in full detail the mathematical formulation for calculating optical flow features from two consecutive mouth areas. This is followed by a summary of feature post-processing. Finally, we focus on training and testing a classifier based on Support Vector Machines.

4.1 Preliminaries

4.1.1 Convolutional neural networks

Convolutional Neural Networks (CNNs, or ConvNets) are a special class of *Deep Neural Networks* (DNNs). They are dominant in the field of computer vision, with common applications including image and video recognition, image classification and image analysis. CNNs take their inspiration from biological processes, where the visual cortex of animals has layers of connected neurons that only respond to a restricted area of visual stimuli. Similarly, convolutional networks can learn the spatial hierarchies in image data through

backpropagation by using multiple simple transformation components. These building blocks include:

1. Convolution layer.
2. Activation function.
3. Pooling layer.
4. Fully-Connected (Dense) layer.

A typical Convolutional Neural Network along with its components is shown in Figure 4.1.

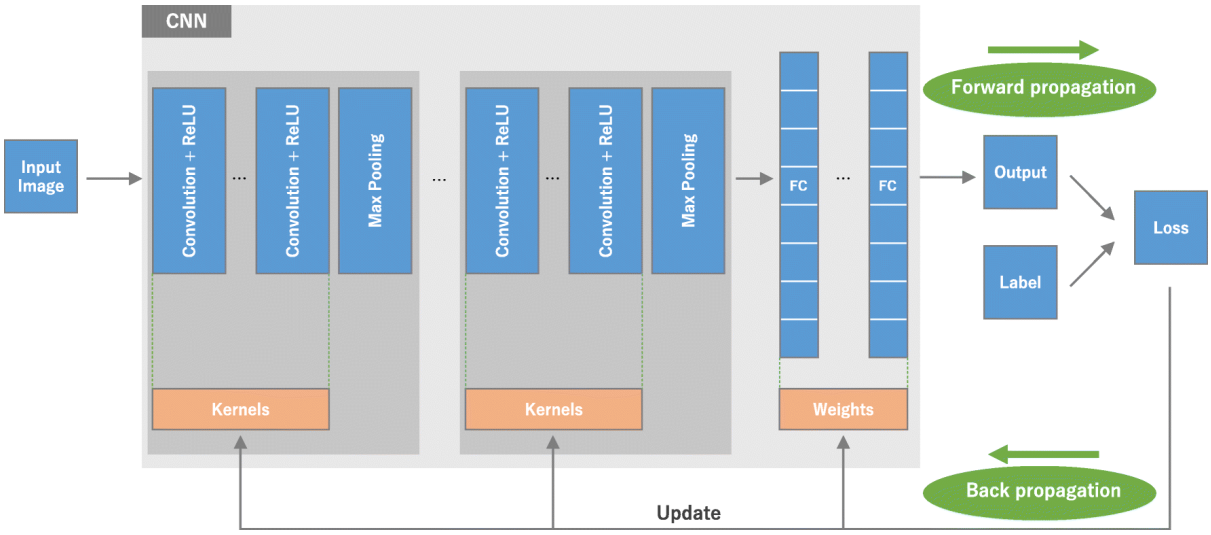


Figure 4.1: A typical Convolutional Neural Network architecture (Figure from [3])

The convolution layer of a CNN (Figure 4.2) includes a filter, known as the convolutional kernel, that is usually much smaller spatially than the input. An element-wise product between each element of the kernel and the input image is calculated at each image location and summed to obtain the final value in the corresponding position of the output, which is also called a feature map. Multiple kernels can be used instead to increase the depth of the feature map.

Many convolution layers can be cascaded. Each linear convolution is passed through a non-linear activation function (Figure 4.3). Some examples of activation functions commonly employed in artificial neural networks are:

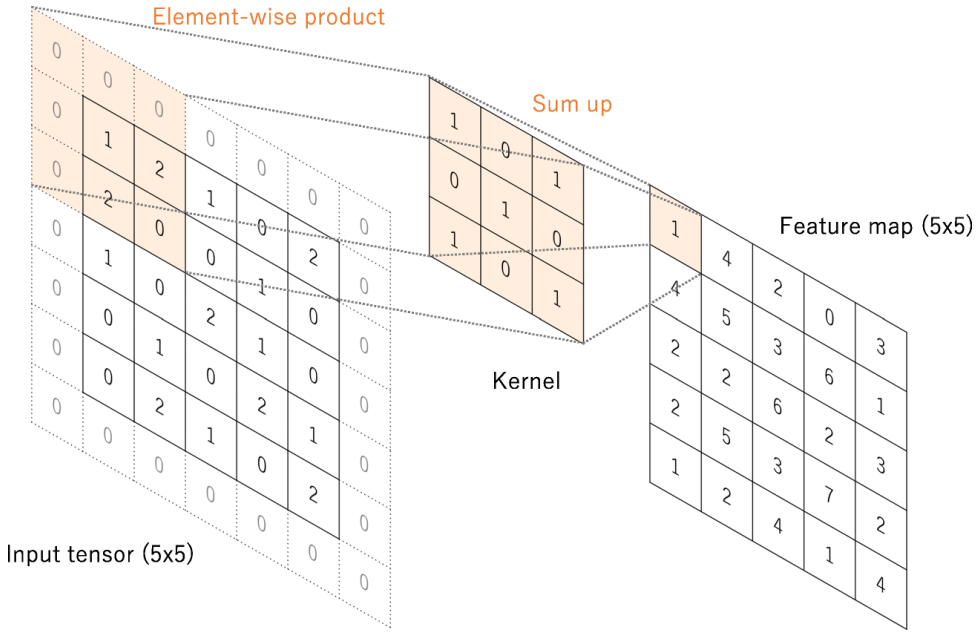


Figure 4.2: Convolution operation (Figure from [3])

1. Binary step

$$y = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{4.1}$$

2. Sigmoid

$$y = \frac{1}{1 + e^{-x}} \tag{4.2}$$

3. Identity

$$y = x \tag{4.3}$$

4. ReLU

$$y = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{4.4}$$

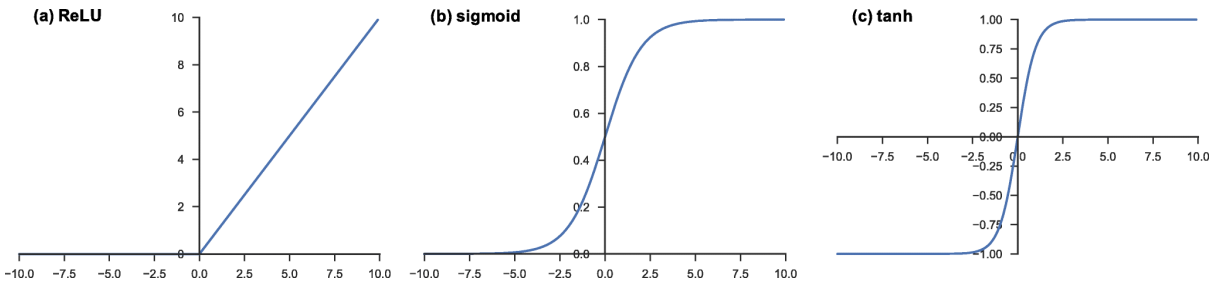


Figure 4.3: Examples of activation functions (Figure from [3])

A pooling layer (Figure 4.4) reduces the dimensionality of the feature map through downsampling. It simultaneously decreases the computational cost of the network and diminishes the possibility of overfitting. Common variants of functions in this layer include *Max Pooling* and *Average Pooling*. Note that pooling layers do not contain any learnable parameters.

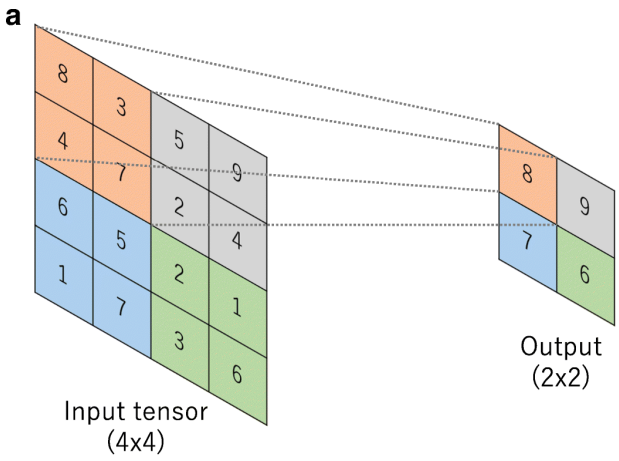


Figure 4.4: Max pooling operation (Figure from [3])

Typically, the output of the last convolution or pooling layer is flattened to a 1-dimensional vector. Then it is connected to 1 or more cascaded fully connected layers, also known as dense layers. In a fully connected layer, every input is connected to every output via a learnable weight. Each dense layer is followed by a non-linear activation function, as with convolution layers.

4.2 Dlib Face Detection

4.2.1 Dlib overview

Dlib is a C++ framework with machine learning implementations to tackle real world problems. It is popular in both industrial and academic contexts. Dlib has many features that make it attractive for our application. It is free and open source, capable of accurate results and features a convenient high-level python API. An additional benefit is the ease of finding examples using dlib on github, due to the large community around it.

4.2.2 HOG face detection

Our first attempt was based on a dlib face detector that was pre-trained via Histogram of Oriented Gradients (HOG) [29]. The detector also employed a latent SVM classifier (LSVM) trained via gradient descent, an image pyramid (Figure 4.5) and detection during a sliding window. The algorithm used by dlib is an improved version [30] of the original. It features a convolutional kernel that is applied at multiple positions and scales of the input image to compute gradients. In addition to the "root" shape detection filter, "part" filters can detect parts of the original shape. As an example, for a bicycle detector, each filter might detect a wheel. The dlib solution tries to independently match each part to the parts of the input image, then fixes the root location and finalizes the locations of the parts relative to their root. The output of the detector is a bounding box for every face in the image.

The HOG detector was initially chosen for its simplicity and fast detection. However, early testing showed that this method was insufficient for detecting faces in the dataset. Indeed, the face detection error rate was over 80% for a clip with the participant in all 10500 frames. In other words, the algorithm consistently missed a face that would be obvious for a human to detect. This can be explained by the challenging nature of the dataset. The realistic meeting scenarios in the AMI Corpus feature speakers that do not face the camera directly, rotate their heads to address other speakers or follow the conversation and, in some cases, cover part of their face with their hands. At this point, it was evident that a different face recognition method was required in order to proceed with region of interest (ROI) extraction.

4.2.3 CNN Face Detection

Our search for a more robust face recognition module led us to dlib's max-margin human detector, which is based on Max-Margin Object Detection (MMOD) [31]. It should be noted that this iteration of MMOD utilizes CNNs instead of the original dlib MMOD implementation with HOG followed by a linear filter. The benefits of this new method are higher generality (no need to train multiple detectors for different poses), lower amount of

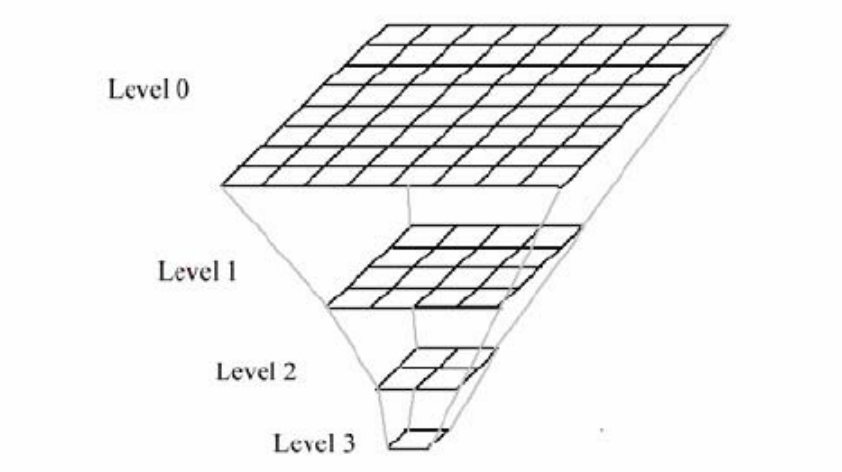


Figure 4.5: An image pyramid (Figure from [4])

required training data and higher recall than other methods (at the time of publication) such as Face Detection with the Faster R-CNN [32].

Unlike earlier methods that trained on a subset of image windows (sub-sampling) and thus could not optimize the entire behavior of an object recognition system, MMOD runs over all detection windows. Moreover, it jointly minimizes missed detections and false alarms, which leads to higher recall and higher accuracy respectively. Additionally, it offers an elegant solution to partially overlapping object detections, which are neither completely true detections or false alarms. Finally, MMOD is flexible enough to work with HOG, bag-of-visual-word models or, in our case, CNNs.

Non-maximum suppression is a common practice in the object detection domain to prevent labeling of overlapping items. MMOD defines rectangles r_1 and r_2 as not overlapping when:

$$\frac{Area(r_1 \cap r_2)}{Area(r_1 \cup r_2)} < 0.5. \tag{4.5}$$

where

$$Area(r_1 \cap r_2) \tag{4.6}$$

is the overlapping area of r_1 and r_2 and

$$Area(r_1 \cup r_2) \tag{4.7}$$

is the total area. Therefore, MMOD uses an intersection-over-union metric.

Let f be a window scoring function, x an image, y a labeling and $\mathcal{Y}$ the set of all valid labelings. The object detection procedure is defined as:

$$y^* = \arg\max_{y \in \mathcal{Y}} \sum_{r \in y} f(x, r). \tag{4.8}$$

While an algorithm for finding the f that optimally jointly minimizes the number of false alarms and missed detections is computationally intractable, MMOD features an optimized version of the greedy peak sorting algorithm that works well in practice.

The original Max-Margin Object Detection [31] used only linear mappings for window scoring:

$$f(x, r) = \langle w, \phi(x, r) \rangle \tag{4.9}$$

where ϕ extracts a feature vector from the sliding window rectangular area r in image x, and w is a vector of parameters. We call $F(x, y)$ the sum of scores for a set y of rectangles. Therefore, the detection procedure becomes:

$$y^* = \arg\max_{y \in \mathcal{Y}} F(x, y) = \arg\max_{y \in \mathcal{Y}} \sum_{r \in y} \langle w, \phi(x, r) \rangle. \tag{4.10}$$

We're trying to train a parameter vector w to minimize detection errors (false positives and false negatives). Note that in the dlib detector we use for this thesis, a Convolutional Neural Network is trained instead of w. For every image x_i with label pair $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ we want the correct labeling to have a higher score than all incorrect labelings. In mathematical terms, we are trying to maximize the occurrences of:

$$F(x_i, y_i) > \max_{y \neq y_i} F(x_i, y) \tag{4.11}$$

For a set of images $\{x_1, x_2, \ldots, x_n\} \in \mathcal{X}$ with respective labels $\{y_1, y_2, \ldots, y_n\} \in \mathcal{Y}$, the max-margin [33] algorithm attempts to find the parameters w that make correct predictions when presented with the training samples. The Max-Margin Objection Detection method defines a convex optimization problem:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & F(x_i, y_i) \geq \max_{y \in \mathcal{Y}} [F(x_i, y) + \triangle(y, y_i)], \quad \forall i \end{aligned} \tag{4.12}$$

with $\triangle(y, y_i)$ signifying the cost of a false labeling of y instead of the correct y_i. This loss is defined as:

$$\triangle(y, y_i) = L_{miss} \cdot (\#\text{ of missed detections}) + L_{fa} \cdot (\#\text{ of false alarms}) \tag{4.13}$$

where L_{miss} and L_{fa} are the relative weights to control for high recall versus high precision respectively.

For data in the real world that is noisy, not perfectly separable or contains outliers, we prefer a soft-margin approach to the hard-margin problem formulation of (4.12). Thus, the soft-margin formulation for Max-Margin Object Detection (MMOD) becomes:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & F(x_i, y_i) \geq \max_{y\in\mathcal{Y}}[F(x_i, y) + \triangle(y, y_i)] - \xi_i, \quad \forall i, \\ & \xi_i \geq 0, \quad \forall i \end{aligned} \tag{4.14}$$

where C controls the tradeoff between fitting the training data and maximizing the margin, as in SVMs. Each ξ_i is an upper limit to the loss caused by training with the pair (x_i, y_i) of example and true label. To prove this (let $g(x) = \max_{y\in\mathcal{Y}} F(x, y)$)

$$\xi_i \geq \max_{y\in\mathcal{Y}}[F(x_i, y) + \triangle(y, y_i)] - F(x_i, y_i) \tag{4.15}$$

$$\xi_i \geq [F(x_i, g(x_i)) + \triangle(g(x_i), y_i)] - F(x_i, y_i) \tag{4.16}$$

$$\xi_i \geq \triangle(g(x_i), y_i) \tag{4.17}$$

In the last substitution, $g(x_i)$ is defined as the element of set $\mathcal{Y}$ that maximizes $F(x_i, \cdot)$ and therefore $F(x_i, g(x_i)) - F(x_i, y_i) \geq 0$.

This makes the soft-margin MMOD objective function (4.14) a convex upper bound on the average loss per training image

$$\frac{C}{n}\sum_{i=1}^{n}\triangle(\arg\max_{y\in\mathcal{Y}} F(x_i, y), y_i). \tag{4.18}$$

When ξ_i is zero, the detector produces the correct result for the training example (x_i, y_i).

The max-margin method was not originally introduced in MMOD. It had been successfully used in other domains [33, 34].

As mentioned in Section 4.2.2, a HOG system using MMOD as a loss function did not achieve sufficient accuracy for our intended purpose. Since we ended up using an improved implementation of MMOD with the HOG feature extraction replaced with a CNN [35], we will not reproduce the mathematical expressions for solving the original MMOD optimization problem. They can be found in [31].

4.2.4 CNN face detection accuracy

At this junction, we will examine the accuracy of the dlib MMOD implementation with a CNN feature extractor in our test data. Empirically, it produces excellent results (Figure 4.6a). In 1 of the 4 video clips we tested, the face recognition algorithm had only 7 missed detections in 10500 frames. Upon closer examination, it was revealed that all of those cases involved the speaker hiding most of their face with their hand (Figure 4.6c). Surprisingly, the face detector worked in slightly less pathological cases where the speaker was still covering part of their face with their hand (Figure 4.6b).

In another clip, the algorithm only missed 1 out of 10500 frames. Finally, tests for the last 2 speakers exhibited perfect recall.

(a) Normal detection **(b)** Detection with occlusion **(c)** Missed detection

Figure 4.6: Example frames with CNN face recognition

Evidently, the face detector was excellent for our needs. However, before we could proceed to feature pre-processing we had to overcome the extreme performance demands of this algorithm in practice.

4.2.5 CNN face detection CPU performance

While the CNN method was very accurate (Section 4.2.4), it was also rather demanding in computing resources. The initial tests were performed using the CPU of a standard consumer desktop PC that was relatively old.

According to our tests, one frame could be processed in approximately 7 seconds. The video clip framerate was 30 frames per second. Therefore, it would take $30 * 7 = 210$ seconds = 3 minutes and 30 seconds to process a single second of video. The duration of each video clip tested is 3 minutes and 20 seconds = 200 seconds. Simple calculations reveal that it would take $200 * 210 = 42000$ seconds = 700 minutes = 11.667 hours of computation to process only one video clip. Hence, a different course of action was required.

4.2.6 Cloud deployment

In the previous section we calculated the time to process a single video clip. It was evident that an iteration time of over 11.5 hours would have been untenable for developing the face recognition part of our solution. An additional requirement was being able to process multiple video clips, not exclusively the one we tested on. Thus, we began exploring alternatives to using our desktop CPU.

One option would have been looking into workstation-grade GPUs. We decided not to pursue that avenue because it had different tradeoffs than our needs. While buying a desktop GPU would enable faster computation time, it was difficult to justify the cost for a one-off project. On the other hand, using university hardware meant working with reservations around the timeline of other experiments, sacrificing flexibility.

The solution we arrived at when looking for higher hardware performance as well as flexibility was deploying our algorithm to a cloud computing platform. In particular, we chose Google Colaboratory [36] (Google Colab in short), a platform for students, data scientists and AI researchers. Importantly, it offered a free tier with GPUs that were sufficiently performant for our needs. Another convenience was that many libraries, including all of the ones we were using, were already installed in Colaboratory by default.

This enabled us to take our existing python code and run it directly on the cloud platform without changing anything.

Since our training and testing data involved videos, we had to find a way to load them on the cloud platform. To that end, we transferred the local data to a Google Drive account. Then, using the Google Drive integration in Google Colab, we imported the data so that our scripts could use it.

The result was a marked improvement in speed. The 10500-frame video clip that would take over 11.5 hours to process locally could now processed in the cloud in only 7 minutes. Effectively, this represented a 100-fold speedup. With this new iteration time, we could proceed to feature pre-processing.

From the above, we conclude that older GPU hardware or even CPU hardware can be used to run simple machine learning models. On the other hand, state-of-the-art face detection models that depend on deep neural networks can achieve dramatic speedups by moving to modern hardware designed for them. Finally, we note that while it was possible to reserve even more computing resources, it would have exceeded the capabilities of the free tier in Google Colab. A research team should be mindful of costs and evaluate all options carefully.

4.3 ROI Selection

The dlib CNN face detector (Section 4.2.3) finds human faces in a frame and returns bounding boxes for each face. In our meeting dataset, each participant has their own face cameras, so the detector finds at most one person per frame. This represents an advantage that might not be immediately obvious. Normally, in an image containing multiple faces, we would have to implement a face tracking algorithm to ensure that the face id of the same person does not switch from one frame to the next. Conversely, we can assume one face id per video clip in our meetings.

We chose to extract the useful information about speech from an area (known as ROI, or region of interest) around the mouth of each face in every frame. This is a common practice in the field of visual speaker diarization [18].

Note that dlib includes a pose estimator based on the Ensemble of Regression Trees methodology [37] and trained on the iBUG 300 face landmark dataset [38]. This estimator matches the pose of a face in a bounding box to 68 facial landmarks. However, we did not utilize this method to locate the mouth area, as it was designed to work with the HOG face detector (Section 4.2.2) and not the CNN face detector (Section 4.2.3) we employed in the preceding stage.

In order to find the mouth of each face, we used manual placement. The number of pixels selected is potentially different in every frame, because it is a function of the size of the bounding box in each direction (vertically and horizontally). To alleviate this, the ROI is rescaled to the same size (32×32 pixels) and normalized. The custom orthogonal mouth areas are designed to be larger vertically than horizontally, since we intuitively expect vertical mouth motion to have higher correlation with speech than other random movement. According to our experiments, this assumption was proven to be correct.

4.4 Optical Flow

After extracting the mouth area (ROI), we needed to find a way to measure movement within it, which would reveal the presence of speech. A natural solution arrived in the form of optical flow [39] techniques. These enable estimating the movement of an object from its pixel displacement in two consecutive frames. Crucially, no additional spatiotemporal information is required.

The neighborhood of a pixel is approximated via polynomial expansion. The local signal model in the form of a quadratic polynomial, expressed in a local coordinate system, is:

$$f(\mathbf{x}) \sim \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c, \tag{4.19}$$

where $\mathbf{A}$ is a matrix, $\mathbf{b}$ is a vector, and c is scalar. The coefficients of the model are estimated by fitting a least squares model to the neighborhood signal. That model features two weighting parameters, certainty and applicability, which are the same as in normalized convolution [40]. Certainty refers to signals in the neighborhood and should be set to zero outside the image. Applicability selects the relative importance of pixels based on where

they are in the neighborhood. Generally, we want to bias in favor of the center pixel and have the weights decrease as the radius grows, with the width of the applicability controlling the size of the receptive field.

We assume the function in a neighborhood is approximated by a polynomial and examine what happens when it undergoes a displacement $\mathbf{d}$. Afterwards, we can solve for the global displacement $\mathbf{d}$. However, this global translation is an unrealistic scenario when applying the algorithm to real signals. Therefore, we replace the global polynomial with a local variant. Then, we attempt to solve a polynomial expansion of two images, considering the approximation:

$$\mathbf{A}(\mathbf{x}) = \frac{\mathbf{A}_1(\mathbf{x}) + \mathbf{A}_2(\mathbf{x})}{2}, \tag{4.20}$$

where $\mathbf{A}_1(\mathbf{x})$ refers to the matrix of the polynomial of the first image and $\mathbf{A}_2(\mathbf{x})$ to the matrix of the polynomial of the second image. Additionally, we can set:

$$\boldsymbol{\Delta}\mathbf{b}(\mathbf{x}) = -\frac{1}{2}(\mathbf{b}_2(\mathbf{x}) - \mathbf{b}_1(\mathbf{x})) \tag{4.21}$$

to obtain the primary constraint:

$$\mathbf{A}(\mathbf{x})\mathbf{d}(\mathbf{x}) = \boldsymbol{\Delta}\mathbf{b}(\mathbf{x}). \tag{4.22}$$

Note that the global displacement is also replaced by a spatially varying displacement field.

The error due to noise from computing the quadratic polynomial in a pointwise manner is too large. To fix that, we assume that in every neighborhood the displacement field is only slowly varying. This means that we can integrate the information about each pixel over a neighborhood. Then, we minimize the weighted square error.

We can increase robustness through parameterizing the displacement field according to

the eight-parameter motion model in two dimensions:

$$\begin{aligned} d_x(x,y) &= a_1 + a_2 x + a_3 y + a_7 x^2 + a_8 xy, \\ d_y(x,y) &= a_4 + a_5 x + a_6 y + a_7 xy + a_8 y^2. \end{aligned} \tag{4.23}$$

Or, in matrix notation:

$$\mathbf{d} = \mathbf{S}\mathbf{p}, \tag{4.24}$$

$$\mathbf{S} = \begin{pmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{pmatrix}, \tag{4.25}$$

$$\mathbf{p} = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \end{pmatrix}^T. \tag{4.26}$$

The weighted square error is:

$$\sum_i w_i \left\| \mathbf{A}_i \mathbf{S}_i \mathbf{p} - \Delta \mathbf{b}_i \right\|^2, \tag{4.27}$$

where i iterates over the neighborhood pixels. Minimization of that error leads to the solution:

$$\mathbf{p} = (\sum_i w_i \mathbf{S}_i^T \mathbf{A}_i^T \mathbf{A}_i \mathbf{S}_i)^{-1} \sum_i w_i \mathbf{S}_i^T \mathbf{A}_i^T \Delta \mathbf{b}_i. \tag{4.28}$$

Both $\mathbf{S}^T\mathbf{A}^T\mathbf{A}\mathbf{S}$ and $\mathbf{S}^T\mathbf{A}^T\Delta\mathbf{b}_i$ can be computed at each point and then averaged according to the weights w.

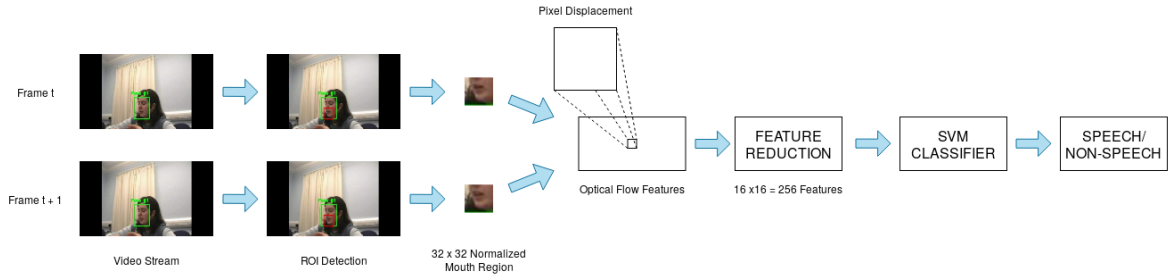


Figure 4.7: The optical flow with SVM visual diarization method

4.5 SVM Visual Diarization

4.5.1 Speech detection features

We begin by detecting the faces of each participant in each frame and selecting the mouth area via manual placement (Section 4.3) of the region of interest (ROI). As already mentioned, these mouth regions have been resized and normalized into a 32×32 image for consistency. Then, we calculate the displacement of every pixel in the mouth region (ROI). The displacement of each pixel is a complex number, having both magnitude and orientation. All the displacements together constitute the optical flow features used as input for the algorithm. Finally, we include a dimensionality reduction step to downsample the optical flow features down to $16 \times 16 = 256$. This is achieved via an average pooling layer that calculates the average of four pixels in every 2×2 square neighborhood.

The complete optical flow with SVM diarization method is illustrated in Figure 4.7.

4.5.2 Speech detection SVM training

These optical flow features were used to train an SVM classifier. In total, about 6000 reduced feature vectors from face cameras with conversation in meetings were used to train the classifier. In particular, each of the following 3 videos contributed about 2000 training examples to the training procedure:

- **ES2002c, 2nd clip (5 minutes)** - 2 speaker videos
- **IS1009b, 1st clip (6 minutes)** - 1 speaker video

In this thesis, the Radial-Basis Function network (RBF) kernel was employed to train the SVM with python's sklearn library. The target labels for each frame were:

$$\begin{cases} 0, & \text{if not speaking} \\ 1, & \text{if speaking} \end{cases}$$

4.5.3 Speech detection SVM testing

There is one potential speaker in each video. For each person and the corresponding selected ROI (Section 4.3) in every frame, we used the SVM model to reach a binary prediction between speech/non-speech. Afterwards, the predictions for each person separately were summed within a window of N frames. The person with the maximum value in a window was designated the speaker of that window. The target labels were chosen to be the same that were used during the training phase. The following 4 videos (one for each individual speaker) were used for testing our visual diarization method:

- **ES2002c, 1st clip (3 minutes)** - 4 speaker videos

We ensured that every participant of the meeting speaks in the test clip. Then, we created the hypothesis annotations with binary labels from the clips. The hypothesis annotations were then compared to the reference annotation from the dataset (see Section 3.4.5) to measure the diarization error rate (DER). The results appear in Section 6.3.

Chapter 5

Audio-Visual Speaker Diarization

Our audio-visual solution to the diarization task is chiefly based on the visual only mode and relies on improving it though the addition of audio information. In this chapter, we begin by covering the motivation for audio-visual fusion. Afterwards, we introduce the framework we employed for audio analysis and expand on the reasons for choosing it. We continue by explaining audio feature extraction in detail. In particular, we focus on the problem of synchronization between the audio and video streams, which run at different rates. Finally, we elucidate training and testing the proposed classifier with the augmented information.

5.1 Motivation for Audio-Visual Fusion

Throughout our measurements, we observed that the visual modality achieved good results. However, it missed shorter speech segments, including interjections. In contrast, the audio method could locate speakers even when face detection failed. On the other hand, it was incapable of discovering the correct number of speakers in some extreme cases. In order to combine the strengths of both the modalities and address their respective weaknesses, we opted to add audio information to the visual implementation.

5.2 Custom MFCC Feature Extraction

5.2.1 Reasoning for custom MFCC computation

In theory, we could have utilized the MFCC computation stage of LIUM SpkDiarization that employs the open-source Sphinx 4 toolkit (see Section 3.2.1). SpkDiarization itself is open-source and designed in a modular fashion to encourage exploration of different diarization architectures. Unfortunately, it is implemented in the Java programming language. In order to keep our code base consistent, as well as have complete control of the diarization pipeline, we decided to write a custom python script that extracts the MFCC features (Section 3.2.3) from the audio dataset. To that end, we employed the widely used librosa python package for music and audio analysis.

5.2.2 Librosa

In our search for MFCC extraction libraries, we came across librosa, a python collection of submodules targeting a variety of interrelated audio tasks:

- **Tempo and beat detection**
- **Harmonic-percussive source separation (HPSS) and generic spectrogram decomposition**
- **Visualization**
- **Audio processing effects (e.g. pitch shifting, time stretching)**
- **Feature extraction (including MFCC)**
- **Filter-bank generation**
- **Onset detection and onset strength computation**
- **Structural segmentation**
- **Sequential modeling**
- **Additional utilities (e.g. normalization, padding)**

Librosa presents numerous advantages for our use case. These include a low barrier to entry for scientific researchers, standardized parameters and interfaces, backwards compatibility with reference implementations, subscription to modern development practices around testing and documentation as well as modularity [41]. The last feature was especially important for us, since we wanted to use only parts of the framework. In more practical considerations, librosa is also free and open-source.

For the most part, we employed the default MFCC extractor configuration. We had to tune the *hop_length* parameter during the synchronization of the audio and video streams (Section 5.3). In the name of reproducibility, we should also mention that version 0.8.0 [42] of librosa was used in our python script.

5.3 Audio and Video Synchronization

After extracting the MFCC audio features from the dataset, we had to select a process to combine them with the existing visual features. In the literature, we found an algorithm that duplicated video features to match every audio window inside a longer video window [43]. Following that, both the audio and video models of a classifier are jointly optimized. Another paper [44] included two ways of performing the audio-visual fusion: a late fusion approach that combined the results of two separately trained models and an early fusion method that combined the parameter training to maximize the global performance of the system.

In our system, we considered that MFCCs are extracted from a window of audio samples. As a result, there should be overlap between consecutive windows, with the features strongly correlating to the center of each window. We opted for matching an audio window to every video difference frame.

Note that we refer to the optical flow difference features (Section 4.5.1) between two consecutive video frames. Evidently, $N + 1$ actual video frames are required to produce N difference frames. This implementation detail doesn't affect the following calculations.

The audio files we were working with featured a sample rate of 44100 Hz, while the video frame rate was 30 frames per second. This is covered in detail in Section 3.4.4. We set

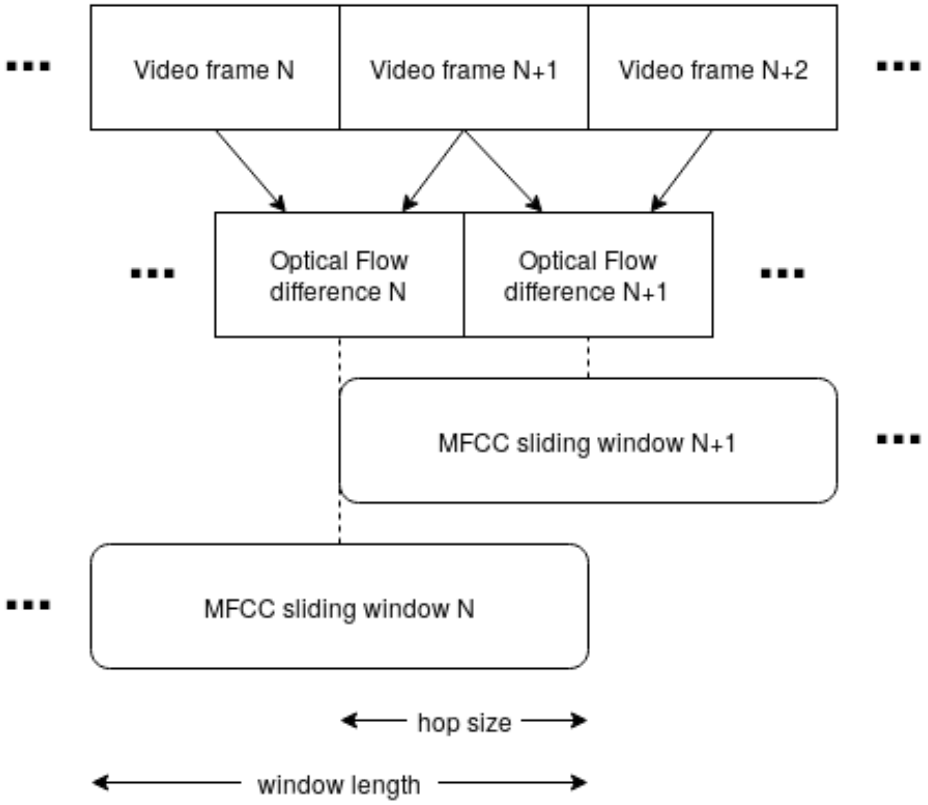


Figure 5.1: The synchronization process between audio and video streams

the *hop_size* parameter to $44100/(30 \times 2) = 735$ samples, exactly half the length of each MFCC sliding window, 1470 samples.

There is one final complication to address. There are frames where the detector doesn't find any faces. Since there are at most 7 missed detections in a test clip of multiple thousands of video frames (see Section 4.2.4 for more details), the error will not be majorly affected. We chose to fill any missing optical flow features via zero-padding.

The synchronization process can be seen in Figure 5.1.

The complete audio-visual diarization solution is illustrated in Figure 5.2.

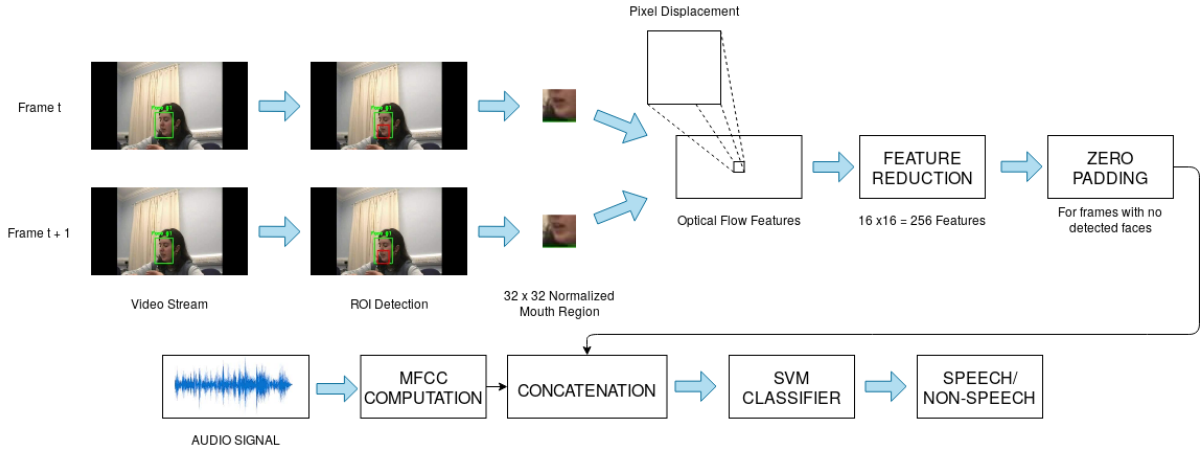


Figure 5.2: Audio-visual fusion using MFCC information

5.4 SVM Audio-Visual Diarization

5.4.1 Speech detection features

The first step in feature extraction is detecting the participant faces and locating the mouth of each person (see Section 4.3). Next, the displacement of each pixel in the resized 32×32 mouth region (ROI) is computed as magnitude and orientation. Following that, an average pooling stage reduces the dimensionality of the optical flow features to 16×16. Until this point, all steps have been equivalent to the visual method (Section 4.5.1). Now, we choose to add zero padding to the visual features which is key to the synchronization of the two different modalities (audio and video). Indeed, we cannot have detection gaps in only one of the streams. Finally, we augment the optical flow features by concatenating them with the MFCC features obtained via librosa.

5.4.2 Speech detection SVM training

These concatenated audio-visual vectors were used as the training examples for an SVM classifier. In total, about 42000 combined feature vectors from face cameras and headset microphones in company meetings were used to train the classifier. In particular, each of these 4 videos contributed about 10500 training examples to the training routine:

- **ES2002c, 2nd clip (5 minutes)** - 4 speaker videos

As in the visual-only case (Section 4.5.2), an RBF kernel was chosen to train the SVM with sklearn. The binary target labels were also the same.

5.4.3 Speech detection SVM testing

In order to test the complete audio-visual algorithm, we used the following 4 videos (one for each individual speaker):

- **ES2002c, 1st clip (3 minutes)** - 4 speaker videos

All participants of the meeting speak at least once during the test clip. In the audio-visual fusion stage, we augmented the video information with audio features (as seen in Figure 5.2). Afterwards, we used the classifier on the test clips, obtaining binary speech/non-speech labels as output. Next, we created hypothesis annotations from those predicted labels. Finally, we compared the hypothesis annotations to the reference (manually created) annotations from the AMI Meeting Corpus dataset. We observed that audio-visual fusion results in an improved DER, as well greater temporal accuracy (Section 6.4).

Chapter 6

Evaluation

6.1 Diarization Error Rate

The dataset derived from the AMI Meeting Corpus, as described in chapter 2, was used for the purpose of evaluating of the speaker diarization systems we developed. In order to measure the error rate of our methods, we chose the Diarization Error Rate (DER) [45] metric, a very common measure of diarization tool accuracy that is defined as follows:

$$\text{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}}, \tag{6.1}$$

where the equation's terms have the following definitions:

- **false alarm** - The duration when a person was labeled as speaker, but wasn't speaking in these frames.
- **missed detection** - The duration when speech exists but the diarization system labeled it as non-speech.
- **confusion** - The duration when the diarization system labeled a speaker incorrectly.

6.2 Audio-Only Evaluation

During the evaluation of our audio method featuring the LIUM SpkDiarization framework, we first calculated the DER of each speaker separately. Each audio file included one of the 4 speakers in the test clip, **the 1st clip of the ES2002c video**. Note that video and audio from this same span of the dataset was selected for all methods we developed. This enabled us to compare their accuracy in a fair manner.

In addition, we created a total hypothesis annotation by combining the audio hypothesis annotations for all 4 speakers (Section 3.4.7). Then, we compared the audio hypothesis annotations to the corresponding reference annotations, both for each speaker separately and in total. The python DiarizationErrorRate function was employed to calculate the DER metric. The following table 6.1 contains a summary of the results of the audio-only approach:

audio	DER (%)
speaker 1	37
speaker 2	25
speaker 3	71
speaker 4	70
total	33

Table 6.1: Results of audio-only diarization.

At first, the algorithm seems to mislabel the speech segments of the 3rd and 4th speakers. Upon further examination, we found that those speakers rarely speak (other than interjections) during the 3 minutes of the test clip. As a consequence, a single false alarm or missed detection can cause the duration-based error percentage to explode. Our methodology is relatively stable when it comes to the 1st and 2nd speaker, who each have greater duration speech segments. The total DER, which is duration-weighted and not a simple average, confirms this.

Still, performance of the LIUM method isn't perfect. This can be partly attributed to the challenging nature of the dataset and its realistic interruptions. Another reason for the mistakes is that the LIUM SpkDiarization machine learning models have been pre-trained on broadcast news. Considering that diarization in company meetings is not their intended use case, they perform reasonably in that task.

It must be pointed out that overlapping speech segments from different speakers should also be discovered by an ideal algorithm. In our case, however, LIUM SpkDiarization does not include that capability. As a result, the DER is expected to be at least equal to the duration of the overlapping segments. This can only be prevented by choosing an alternative method. In practice, meetings only include small amounts of overlap, mainly centered around interjections. As the typical duration of overlapping speech is a few seconds, it has little effect on overall results.

Specifically for our test clip, only one instance of nontrivial overlap exists, and that is arguably due to labelling error. In particular, 3 different 1-second interjections in 15 seconds of the manual reference annotation are denoted as overlapping speech for the entire 15-second segment. This manifests as increased error in our measurements.

6.3 Visual-Only Evaluation

In order to measure the Diarization Error Rate (DER), we first created 4 hypothesis annotations, one from each video, through our optical flow SVM solution. We should remark that every video featured a single speaker. Then, we concatenated the 4 hypothesis annotations into a total hypothesis annotation for the test video clip, **the 1st clip of the ES2002c video** (Section 4.5.3). Correspondingly, we wrote a python script to compile a 4-person reference annotation for the same clip utilising information from the meeting dataset (Section 2.4). Finally, we compared the total hypothesis annotation to the reference annotation for the clip.

We chose to evaluate the optical flow method featuring an SVM model with the DER metric, computed via the DiarizationErrorRate python function. For better evaluation of the system accuracy, we decided to measure the DER for five different window sizes. The size N of the windows, measured in frames, was 200, 250, 300, 350 and 400. These windows correspond to 6.66, 8.33, 10, 11.6 and 13.33 seconds respectively, at the clip's stated framerate of 30 frames per second.

In table 6.2 we can see the DER results for the 5 different window lengths.

We conclude that the 350-frame window outperforms the other four cases (N=200, N=250,

Window Length	4-speaker DER (%)
200	35.83
250	29.13
300	28.30
350	**25.21**
400	29.81

Table 6.2: Results of visual-only diarization.

N=300, N=400). This means that the optical flow method cannot easily distinguish between the rapid changes of active speaker in a realistic meeting environment. Indeed, even humans might find the task of detecting speech from image frames difficult.

In constrast, raising the window length allows the algorithm to correctly discover longer passages of contiguous speech (∼20 seconds) by a single speaker. As expected, this only works up to a point. The largest, 400-frame window leads to labeling failure, as it drives the algorithm to examine very large segments (∼13 seconds per segment). Therefore, it lacks the granularity required to correctly estimate speech durations when speakers change.

6.4 Audio-Visual Evaluation

In order to evaluate the complete audio-visual diarization solution, we measured the 4-speaker Diarization Error Rate. For comparison purposes, we used the same test clip as in the audio-only and visual-only approaches, **the 1st clip of the ES2002c video**. As before, we created 4 hypothesis annotations from the output of our system. Then, we assembled them into one hypothesis annotation for the entire clip. Lastly, we compared the total hypothesis annotation to the reference annotations from the transcriptions in the dataset.

The python DiarizationErrorRate function was employed to compute the DER metric. For a more accurate representation of the performance of our system, we chose to benchmark five different window sizes. These include 50, 100, 150, 200 and 250 frames. The five windows correspond to 1.66, 3.33, 5, 6.66 and 8.33 seconds respectively, at a clip framerate of 30 frames per second. Observe that some of the window sizes N in the audio-visual method are different than those employed for the visual method (Section 6.3).

This new method augmented the information from the visual-only algorithm with additional MFCC information. The following table 6.3 covers the DER results for the 5 different window lengths:

Window Length	4-speaker DER (%)
50	26.98
100	23.12
150	**23.07**
200	25.83
250	26.68

Table 6.3: Results of audio-visual diarization.

We can observe a marked improvement in two distinct directions. First, the Diarization Error Rate is better than in the visual-only evaluation, both overall and in the best possible case. Indeed, the worst error in the audio-visual solution is comparable to the *best* error in the visual method. Second, the diarization results happen over a smaller MFCC window. This implies an improvement in temporal accuracy, with the best case corresponding to classification of 5-second segments, instead of 11.6 in the visual case.

6.5 Performance Concerns

Evaluating the computational performance of all 3 approaches (audio, visual and audio-visual), none can be considered online (or realtime) algorithms. The first is based on the clustering sections of LIUM SpkDiarization, which require the entire audio clip to be present in order to function. The latter two have a bottleneck in the face detector that leverages the free tier in Google Colab (Section 4.2.6). The face detector runs at an approximate 0.5×realtime rate. We assume, but have not tested, that more modern hardware can achieve realtime detection.

Chapter 7

Conclusion and Future Work

In this thesis we developed a complete audio-visual speaker diarization solution for meetings with many participants.

We started by examining the effects of using each modality separately. In the audio-only approach, we employed the LIUM SpkDiarization toolkit. This features a complete audio diarization pipeline. Its stages include feature extraction, audio segmentation, speaker clustering and a final ILP clustering that enables diarization. In the video-only diarization system, we explored a different system, leveraging cloud resources for faster iteration. It utilizes the dlib face recognition library and selects the mouth area as region of interest. Then, it uses an optical flow technique to extract the features for an SVM classifier.

Afterwards, we combined the audio and visual modalities to create an audio-visual approach. The visual method was extended with MFCC features to aid the detection of small speech segments. That information was recovered from the audio via a custom script and then synchronized to the video frames, with special consideration for overlapping MFCC windows.

We measured the accuracy of the system via the diarization error rate (DER) metric. Comparison of our methods leads to the conclusion that the audio-visual fusion performs better than using audio or video information separately.

A possible future extension for this system is augmenting the diarization process with location information. This can be discovered from the microphone array streams already present in the AMI corpus. There is additional room for improvement by exploring recent

advances in neural network structures. In particular, Recurrent Neural Networks (RNNs) is an approach that has led to very good performance in the speaker diarization task in the literature, especially in combination with CNNs.

Another idea is taking speech overlap into consideration. Neither our system nor LIUM SpkDiarization have this capability. However, a more general diarization solution will be expected to deal with multiple concurrent speakers. Finally, we could explore matching the algorithm to the nature of the meeting diarization task. In particular, our current system struggles with some common events, particularly interjections. Special handling of these cases could potentially improve results.

Bibliography

[1] S. Meignier and T. Merlin, "LIUM spkdiarization: an open source toolkit for diarization," in *Proc. CMU SPUD Workshop*, 2010.

[2] The AMI corpus [online] available at: http://groups.inf.ed.ac.uk/ami/corpus/.

[3] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, 2018.

[4] V. J. Hodge, G. Hollier, J. Austin, and J. Eakins, "Identifying perceptual structures in trademark images," in *Proceedings of the Fifth IASTED International Conference on Signal Processing, Pattern Recognition and Applications*, ser. SPPRA '08. USA: ACTA Press, 2008, p. 81–86.

[5] C. Wooters and M. Huijbregts, "The ICSI RT07s speaker diarization system," in *Proc. Multimodal Technologies for Perception of Humans*, 2008, pp. 509–519.

[6] M. Bendris, D. Charlet, and G. Chollet, "Lip activity detection for talking faces classification in TV-content," in *Proc. of International Conference on Machine Vision*, 2010, pp. 187–190.

[7] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active shape models - their training and application," *Comput. Vis. Image Underst.*, vol. 61, no. 1, pp. 38–59, 1995.

[8] G. Garau, A. Dielmann, and H. Bourlard, "Audio-visual synchronisation for speaker diarisation," in *Proc. INTERSPEECH*, 2010.

[9] F. Vallet, S. Essid, and J. Carrive, "A multimodal approach to speaker diarization on TV talk-shows," *IEEE Transactions on Multimedia*, vol. 15, no. 3, pp. 509–520, 2013.

[10] H. Vajaria, T. Islam, S. Sarkar, R. Sankar, and R. Kasturi, "Audio segmentation and speaker localization in meeting videos," in *Proc. 18th International Conference on Pattern Recognition*, 2006, pp. 1150–1153.

[11] M. Rouvier, G. Dupuy, P. Gay, E. el Khoury, T. Merlin, and S. Meignier, "An open-source state-of-the-art toolbox for broadcast news diarization," in *Proc. INTERSPEECH*, 2013.

[12] S. Galliano, G. Gravier, and L. Chaubard, "The ESTER 2 evaluation campaign for the rich transcription of French radio broadcasts," in *Proc. INTERSPEECH*, 2009, pp. 2583–2586.

[13] D. Reynolds, E. Singer, B. Carlson, G. O'Leary, J. McLaughlin, and M. Zissman, "Blind clustering of speech utterances based on speaker and language characteristics." in *Proc. of the 5th International Conference on Spoken Language Processing*, 1998.

[14] V. B. Le, O. Mella, and D. Fohr, "Speaker diarization using normalized cross likelihood ratio," in *Proc. INTERSPEECH*, 2007, pp. 1869–1872.

[15] P. Deléglise, Y. Estève, S. Meignier, and T. Merlin, "Improvements to the LIUM French asr system based on CMU Sphinx: what helps to significantly reduce the word error rate?" in *Proc. INTERSPEECH*, 2009.

[16] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[17] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[18] G. Potamianos, E. Marcheret, Y. Mroueh, V. Goel, A. Koumbaroulis, A. Vartholomaios, and S. Thermos, "Audio and visual modality combination in speech processing applications," in *The Handbook of Multimodal-Multisensor Interfaces: Foundations, User Modeling, and Common Modality Combinations - Volume 1*, S. Oviatt, B. Schuller, P. R. Cohen, D. Sonntag, G. Potamianos, and A. Krüger, Eds. Association for Computing Machinery and Morgan & Claypool, 2017, pp. 489–543.

[19] S. S. Chen and P. S. Gopalakrishnan, "Speaker, environment and channel change detection and clustering via the Bayesian information criterion," in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 127–132.

[20] C. Barras, X. Zhu, S. Meignier, and J. L. Gauvain, "Multistage speaker diarization of broadcast news," *Trans. Audio, Speech and Lang. Proc.*, vol. 14, no. 5, pp. 1505–1512, 2006.

[21] S. Tranter and D. A. Reynolds, "Speaker diarisation for broadcast news," in *Proc. Odyssey Speaker and Language Recognition Workshop*, 2004, pp. 337–344.

[22] T. Stafylakis, G. Tzimiropoulos, V. Katsouros, and G. Carayannis, "A new penalty term for the BIC with respect to speaker diarization," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010, pp. 4978–4981.

[23] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *IEEE Odyssey: The Speaker and Language Recognition Workshop*, 2001, pp. 213–218.

[24] M. Rouvier and S. Meignier, "A global optimization framework for speaker diarization," in *Proc. Odyssey Workshop*, 2012.

[25] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[26] P.-M. Bousquet, M. Driss, and J.-F. Bonastre, "Intersession compensation and scoring methods in the i-vectors space for speaker recognition," in *Proc. International Conference on Speech Communication and Technology*, 2011, pp. 485–488.

[27] C. Vossos, "Audio visual speaker diarization in broadcast news," Diploma Thesis, University of Thessaly, Volos, Greece, 2019.

[28] REAPER [online] available at: https://www.reaper.fm/reaplugs/.

[29] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893.

[30] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection

with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010.

[31] D. E. King, "Max-margin object detection," *CoRR*, vol. abs/1502.00046, 2015. [Online]. Available: http://arxiv.org/abs/1502.00046

[32] H. Jiang and E. Learned-Miller, "Face detection with the faster R-CNN," in *Proc. 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, 2017, pp. 650–657.

[33] T. Joachims, T. Hofmann, Y. Yue, and C.-N. Yu, "Predicting structured objects with support vector machines," *Commun. ACM*, vol. 52, pp. 97–104, 11 2009.

[34] Y. Altun, I. Tsochantaridis, and T. Hofmann, "Hidden Markov support vector machines," in *Proc. Twentieth International Conference on Machine Learning*, vol. 1, 2003.

[35] DLib [online] available at: http://blog.dlib.net/2016/10/easily-create-high-quality-object.html.

[36] Google Colaboratory [online] available at: https://colab.research.google.com.

[37] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proc. of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867–1874.

[38] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic, "300 faces in-the-wild challenge: Database and results," *Image and Vision Computing (IMAVIS)*, 2016.

[39] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Proc. Image Analysis*, J. Bigun and T. Gustavsson, Eds. Berlin, Heidelberg: Springer, 2003, pp. 363–370.

[40] H. Knutsson and C.-F. Westin, "Normalized and differential convolution: methods for interpolation and filtering of incomplete and uncertain data," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1993.

[41] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric

Battenberg, and Oriol Nieto, "librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra, Eds., 2015, pp. 18 – 24.

[42] B. McFee, V. Lostanlen, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, J. Mason, D. Ellis, E. Battenberg, S. Seyfarth, R. Yamamoto, K. Choi, V. Morozov, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, D. Hereñú, F.-R. Stöter, P. Friesch, A. Weiss, M. Vollrath, and T. Kim, "librosa/librosa: 0.8.0," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3955228

[43] G. Friedland, C. Yeo, and H. Hung, "Dialocalization: Acoustic speaker diarization and visual localization as joint optimization problem," *ACM Trans. Multim. Comput. Commun. Appl.*, vol. 6, pp. 27:1–27:18, 2010.

[44] P. Ercolessi, C. Senac, and P. Joly, "Segmenting TV series into scenes using speaker diarization," in *Proc. International Workshop on Image Analysis for Multimedia Interactive Services*, 2011.

[45] H. Bredin, "pyannote.metrics: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems," in *Proc. Interspeech*, 2017, pp. 3587–3591.