TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LAURI NISKANEN

MULTI-LABEL SPEAKER RECOGNITION USING RECURRENT NEURAL NETWORKS

Master of Science thesis

Examiners: Professor Tuomas Virtanen and Professor Hannu-Matti Järvinen

The examiners and topic of the thesis were approved on 9 August 2017

Contents

1 Introduction

For a long time computers have been better than any human in doing calculations or simple repetitive tasks. However, tasks that require deeper understanding, creativity, or imagination have been very hard for computers to do. Only recently computers have began to conquer many of these problems. For example, research topics like computer vision, robotics, natural language processing, and automated medical diagnosis have been greatly advanced with the help of modern machine learning [11, 17, 31, 46].

These hard problems are typically so complex that it becomes impossible for a programmer to manage all possible cases in a systematic way. The solution is to use data-driven statistical methods to reduce the dimensionality of the task. Machine learning is the study of algorithms that make predictions from collected data. In contrast to classical computer algorithms, in machine learning the algorithm itself is typically somewhat general, and data is in a very important role. The quality, quantity, and representation of the data can have huge impact on the predictions. The recent success of machine learning is a combination of theoretical advances, more and more extensive data collection, and the availability of high performance graphics processing units (GPU). [44]

One important problem area where computer systems have become better with machine learning is processing human speech. Speech recognition is an especially popular research topic in which the textual message of speech is analyzed. However, human speech also contains information about the age, gender, emotion, and identity of the speaker. In this thesis we focus on speaker recognition and study how the identity of a speaker correlates with audible features. [45, 49]

Speaker recognition has a wide range of possible applications. One of them is annotating who speaks when in meeting recordings or other audio tracks. It could also be used for improving speech controlled home automation systems with multiple users. Voice commands could select their default parameters and preferences based on who gave the command. For example, a specific personal calendar could be selected when a new event is being created. Speaker recognition has also been used as a biometrical component for customer verification in financial services and in criminal investigations [39, 47].

We implement a machine learning system that can identify speakers from given audio samples based on how the voices of individual speakers sound different. The system is applied to a dataset containing recordings of meetings. Each meeting has typically four attendees having natural conversations. Our system analyzes the audio

recording at each time point and tries to determine who is speaking. People mostly speak in turns, but it is not uncommon that two or more people are speaking over each other. Meetings also have brief parts where nobody is speaking. To accomodate these situations, our system does not simply name a single speaker per time point, but instead outputs a list of active speakers. Since this list of speakers is given at each point in time the output for the whole recording is two dimensional as shown in figure 1.

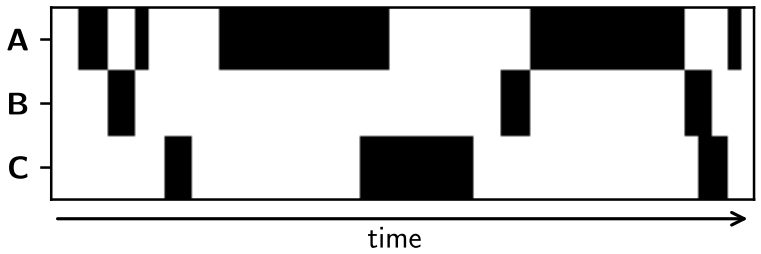

Figure 1: A simplified example of the program output, where A, B, and C are different speakers. The filled segments represent times where each speaker is active during the audio track.

Our system is based on a recurrent neural network classifier. Recurrent neural networks have the ability to remember past information to aid future predictions. This way the system can learn that one speaker is often active for some time before the speaker changes. During difficult points in the audio track the system can support its decisions by the understanding of the previous moments. We compare different neural network models and show how their hyperparameters affect the predictions.

Chapter 2 has an introduction to speaker recognition. Chapter 3 explains how machine learning and neural networks can be used to achieve our goal. Chapter 4 presents our speaker recognition implementation and chapter 5 analyzes the performance of our methods. Chapter 6 has concluding discussion and proposed topics for future research.

2 Speaker recognition

Speaker recognition aims to detect on which parts of an audio track someone is speaking and to identify who the speakers are on each of those parts. There are many ways to recognize speakers. One approach is to use multiple microphones with known locations in relation to the speakers [56]. It is also possible to do speech recogntion on the text of the speech and then recognize one or more specific phrases, for example have the speakers say their name or a password. However, in this thesis we are studing text-independent methods, where we only use acoustic characteristics of speech irrespective of what is being said, and without using the location of the microphones. [24]

This kind of recognition is possible, because people have individual voices. The differences in the voices are mainly caused by anatomical differences in the vocal tract. The shape of the vocal tract produces different resonances in the voice, also called formants. Other affecting factors include the anatomy of the lungs and the trachea. These differences in the voices can be analyzed using the frequency spectrum of the audio. [7]

Traditionally speaker recognition has been split to two phases: speaker diarization and speaker identification. The purpose of speaker diarization is to segment an audio track into runs with only one speaker in each and also separate parts where nobody is speaking [59]. The goal of speaker identification is to then detect the identity of the speaker in each segment [49].

However, the traditional approach has some limitations. First, diarization systems often need to process a whole file at a time and speaker identification is typically done only after segmentation. This means that speaker recognition cannot be done live on an audio stream. Second, people often interrupt each other or talk simultaneously when trying to take the floor. This cannot be accurately represented with one-speaker segments.

To resolve these issues, it is possible to do diarization and identification jointly in one pass. Instead of telling who is speaking on each segment, we recognize the speaker at each time frame. The idea of live recognition has been researched by Vinyals and Friedland [61]. As we avoid needing the one-speaker segments we can take a step further and do multi-label speaker recognition by giving a list of simultaneous speakers for each time frame.

Speaker recognition systems contain two main components: feature extraction and feature matching. The purpose of feature extraction is to represent the audio signal

of the examined speech in a compressed form where unuseful information is filtered out. There is a wide range of speech feature extraction methods such as linear prediction coding (LPC) [42] and Mel-frequency cepstral coefficients (MFCC) [12]. The purpose of feature matching is to connect the extracted speech features to the speaker identity or otherwise classify or cluster it. Feature matching techniques that are used with speech include dynamic time warping (DTW) [54], hidden Markov models (HMM) [2, 48], Gaussian mixture models (GMM) [52], vector quantization (VQ) [13, 41], and artificial neural networks (ANN) [15, 34, 35, 43]. [1, 24, 45, 49]

This thesis presents and analyzes a live multi-label text-independent speaker recognition method with Mel-frequency cepstral coefficients (MFCC) as the speech feature extraction method and recurrent neural network (RNN) as the feature matching method. Next, we introduce the theory behind the vocal feature extraction methods that are needed for our implementation. The neural network components are covered in chapter 3.

2.1 Frequency spectrum

Digital audio streams are typically represented with pulse-code modulation (PCM) [4]. It is a time domain representation where the amplitude of the audio signal is sampled with regular intervals. High quality audio signals are typically stored with 44.1 kHz or 48 kHz sampling rate.

However, in speaker analysis we are interested in high-level features of the audio signal. Individual amplitude samples of the one-dimensional PCM audio signal do not directly correlate with anything that would be useful for speaker recognition. The solution is to transform the signal to a more useful format. Practically all vocal feature extraction methods used in speaker recognition are based on distinguishing the individual frequency modes, or formants, using the frequency spectrum of the speech sample [14].

The spectrum can be calculated using discrete Fourier transform (DFT) [23], which converts the audio samples from time domain to frequency domain. In time domain we can see how the amplitude changes over time, but in frequency domain we can analyze how the audio frequencies of the signal behave. Discrete Fourier transform is defined by

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N},$$

where $x_0, x_1, \ldots, x_{N-1}$ represents uniformly spaced time domain samples and $X_0, X_1, \ldots, X_{N-1}$ is a sequence of complex numbers containing information about the amplitude and phase of the frequencies in the signal.

2.2 Mel-frequency cepstrum

One step further to make the signal more compact and better suited for speaker classification is to calculate the Mel-frequency cepstrum [12]. The Mel-frequency cepstral coefficients (MFCC) are widely used as a feature vector in the field of automated speech analysis [36, 50, 64, 65]. The Mel scale [58] is a scale of audio pitches derived from listening experiments with the purpose of mimicing how humans perceive audio signals [36]. A frequency f given in hertz can be converted to mels using the formula [53]

$$f_m = 1127 \ln\left(1 + \frac{f}{700\ \text{Hz}}\right) \text{mel}$$

Cepstrum of signal $\overline{\mathbf{x}}$ can be defined as

$$C = DCT\{\log|DFT[\overline{\mathbf{x}}]|^2\},$$

where DFT is the discrete Fourier transform and DCT is the discrete cosine transform [5]. In Mel-frequency cepstrum the frequency bands are spaced based on the Mel scale [36]. Mel-frequency cepstrum is calculated by applying the Mel-weighting function w_m before the discrete cosine transform [64]:

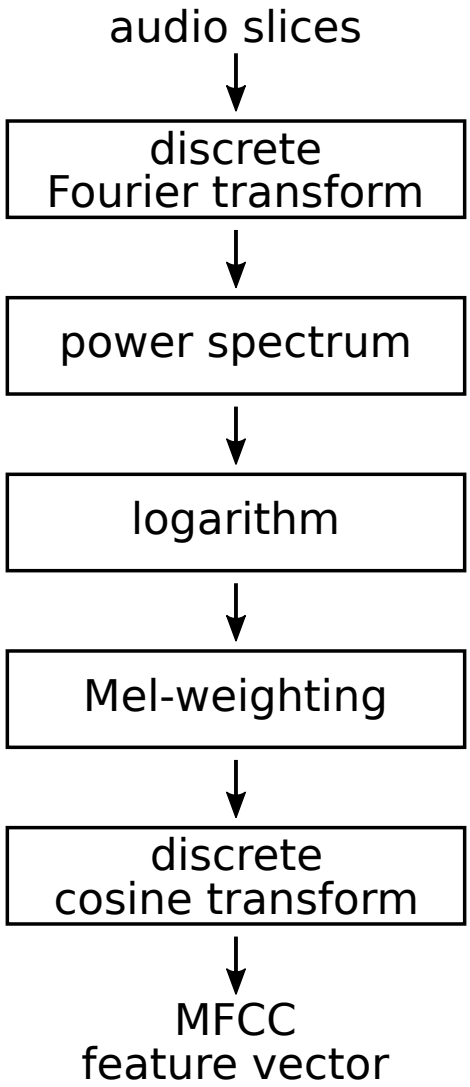

Figure 2: Pipeline for MFCC feature vector calculation.

$$C_m = DCT\{w_m(\log|DFT[\overline{\mathbf{x}}]|^2)\}.$$

The computation flow for the Mel cepstrum is illustrated in figure 2. The discrete Fourier transform is used to calculate the power spectrum of the audio signal. Phase

information can be discarded because it has been shown to be not as important. The logarithm of the spectrum is taken, because it approximately matches the perceived loudness of the signal. Finally, after scaling the signal to the Mel scale, discrete cosine transform is taken to reduce the number of parameters. This is useful because the calculated Mel-spectral vectors consists of highly correlated components. Karhunen–Loève transform [29] would be more precise, but with speech signals discrete cosine transform is commonly used to approximate it. [36]

3 Machine learning

Machine learning can be applied to a wide range of problems, and there are multiple ways to use the collected data to solve the machine learning problems. In this thesis, we are mainly interested in classification, where the goal is to assign a category for each given item. Next, we will explain how classification can be implemented with supervised learning using artificial neural networks.

3.1 Supervised learning

In classification, the goal is to train a prediction model, a classifier, that tells to which category a sample belongs based on its features. The classifier can be trained using example data. Each example has a vector of features and a target label. Features are the input attributes that describe the sample. Labels are the categories to which each sample belongs. [44]

The data must be split to three sets: training samples, validation samples, and test samples. Training samples are used for training the classifier model. Validation samples are used to compare different methods and to adjust the model parameters. Test samples are used to test the performance of the trained classifier. It is important that test samples are not available for the algorithm during the learning stage. The model can be tested by predicting where the test samples should belong based on its features and comparing the result to the sample label that represents the truth. [3]

In supervised learning the algorithm is given access to both the training features and training labels. In contrast, in unsupervised learning the algorithm can only see unlabeled features. There are also other scenarios that differ in how the data is available to the algorithm. [44]

There are many algorithms and models that can be trained to make classification predictions [3]. Next, we will present one of them: artificial neural networks.

3.2 Artificial neural networks

Neural networks were first researched as a way to represent biological information processing with mathematics by McCulloch in 1943 [43]. The term has since been associated with numerous different models, most of which are only remotely related to biology if at all. Neural networks consists of a fixed number of interconnected

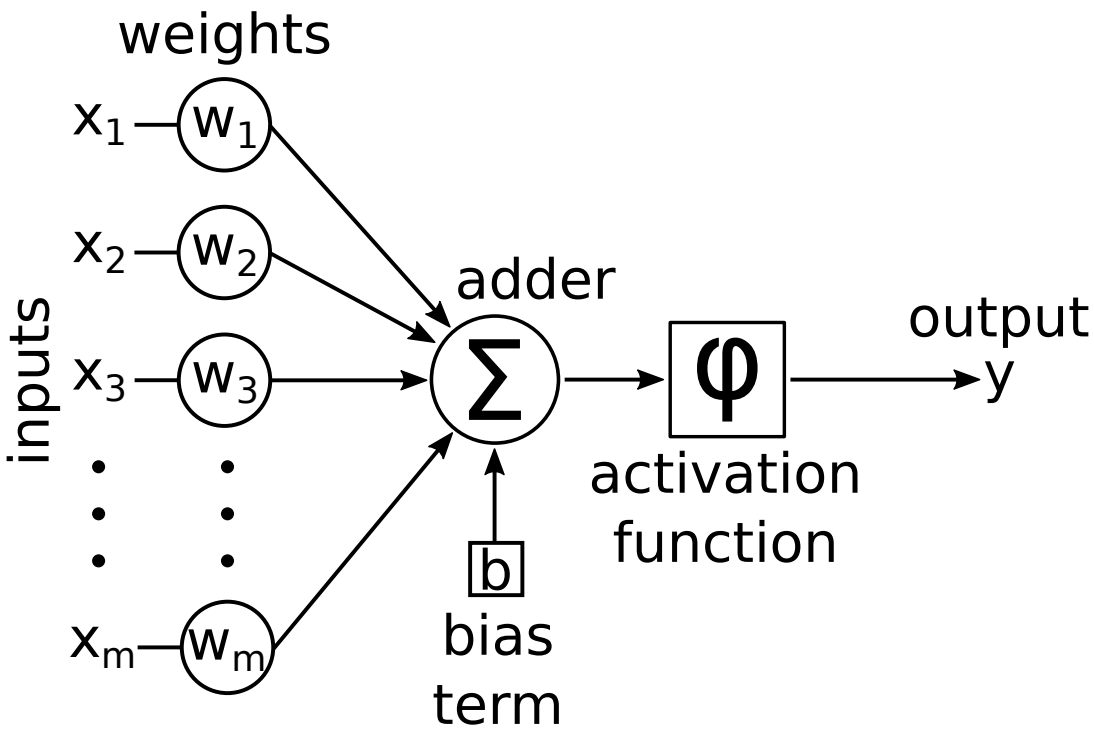

Figure 3: The components of an artificial neuron.

parametric units, or artificial neurons, whose parameters can be adjusted in training phase so that they and the network as a whole produce the desired output. [3]

There are multiple models for artificial neurons, but in the common basic case a neuron can be defined by the parts shown in figure 3. Each neuron has a set of connecting links to predecessor neurons with weights associated to each link. The adder component takes the output values from the connected predecessor neurons, multiplies them by the link weights, and calculates the sum of these values. A bias term can also be added to the sum. Activation function is a function that takes the calculated sum as an input and produces an output value for the neuron. Some commonly used activation functions are listed in table 1. [25]

rectified linear unit	$\begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
logistic sigmoid	$\frac{1}{1+e^{-x}}$
hyperbolic tangent	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
softplus	$\ln(1+e^x)$

Table 1: Common activation functions. [20]

This basic neuron model can be described mathematically as the following equation:

$$y = \varphi\left(b + \sum_{j=1}^{m} w_j x_j\right),$$

where y is the output value of the neuron, φ is the activation function, b is the bias term, $w_1, w_2, \ldots, w_m$ are the link weights, and $x_1, x_2, \ldots, x_m$ are the connected predecessor neuron values. [25]

A feedforward neural network is composed of individual neurons arranged in layers as shown in figure 4. The links between the neurons are defined so that the predecessor of a neuron is in the preceding layer. Some of the neurons are selected to be in the output layer of the network. Their value is visible as the output vector of the whole network. Similarly, some of the neurons are used as the input to the network. The input neurons do not have any predecessors neurons, but instead their value is given from outside of the network. Neurons that are neither input nor output units are called hidden units as they are not directly exposed to the outside. [3]

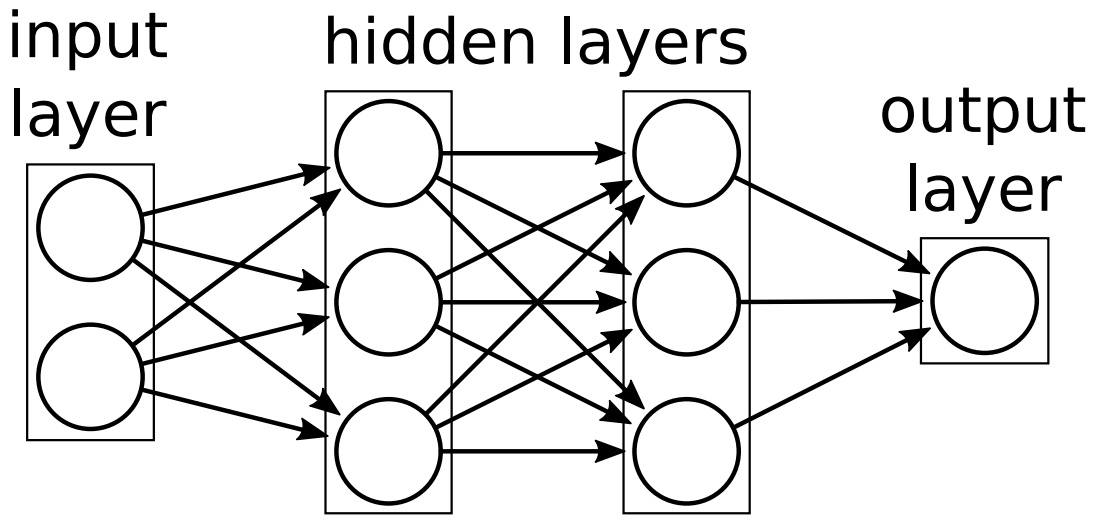

Figure 4: A simplified feedforward network composed of neurons arranged in layers.

Typically the network structure is expressed by defining the number of hidden layers in the network and the number of units in each layer, also called the size of the layer [3]. Different layers may have a different sizes. The input and output layer sizes are usually defined by the intended use of the network. The number and sizes of hidden layers can be adjusted based on the desired network complexity. Different layers may also use different activation functions or extended neuron models, some of which we will examine later. These parameters related to the structure of the network are called the hyperparameters of the network. They are typically not changed during the training process. However, next we see how other parameters, especially the

neuron link weights, are not fixed in place but instead being adjusted dynamically during neural network training.

3.3 Backpropagation

Training a neural network is a process where the neuron link weights, and sometimes other parameters, are adjusted so that the network can produce the desired output values for given input values [25]. In other words, the internal parameters of a neural network are modified so that the network can approximate a given function. Backpropagation [62] is a popular and effective algorithm for training neural networks.

In supervised learning, the neural network is trained with a set of labelled training examples. These examples, or training samples, are pairs of input and output values. When the network is trained to do classification the input values are called features, which describe a given sample in some way. The output vector of the network is used for encoding the class of the sample. For example, in a binary classification case there are positive and negative samples, and the class of the sample can be encoded simply by using one neuron in the output layer and by differentiating the class by the value of the neuron (e.g. 0 or 1). [25]

Before starting the training process it not known what values the neuron link weights should have. They can be initialized randomly. The backpropagation algorithm works by iterating training samples making a forward pass and a backward pass for each of them. In the forward pass the values of the input units are set according to features of the sample. Then, the neuron functions are evaluated propagating the signal towards the output layer. This process produces values for the output neurons of the network. The output values are compared to the expected output vector, which in supervised learning is known for each sample. In the backward pass the neuron link weights are adjusted so that the difference between the expected output vector and the produced output vector decreases. The goal of the training process is to iteratively tweak the parameters in the network so that the response produced in the forward pass matches the desired one more and more closely. [25]

There are multiple ways to determine how the parameters should be adjusted. Mathematically speaking, we need to define a loss function that the training process is trying to minimize. For binary classification the loss function can be the logistic sigmoid function on an output neuron [3]. We will later examine the choice of the loss function and the encoding scheme for output neurons for various classification problems. The gradient of the loss function is useful in calculating the needed change

to the parameters. To decrease the loss, a small step can be taken in the direction of the negative gradient [3]:

$$a_{n+1} = a_n - \eta \nabla E(a_n),$$

where a_n is the value of the examined parameter before the adjustment and a_{n+1} after it, ∇E is the gradient of the loss function, and η is a parameter called the learning rate. Learning rate determines how great the adjustment steps are. When iterating the training samples randomly and evaluating the loss function for one sample at a time, this way of adjusting the network is known as stochastic gradient descent [51].

3.4 Multi-label classification

Basic single-label binary classification answers to the yes–no question "Does the sample match the criteria?" In practice the question can be for example "Is the object in the given image a cat?" In these cases the neural network output is typically encoded as a single neuron whose value is either one or zero. Values in between may represent varying levels of uncertainty. This encoding scheme can be extended to support multiple mutually exclusive classes, answering for example to the question "Is the animal in the image a cat, a dog, or a horse?" With n classes the output is often encoded with n neurons so that one neuron has the value one and the rest have value zero. Uncertainty can be encoded by assigning weighted values to the neurons so that the sum of the values of all output neurons is one. This is called multiclass or multinomial classification with one-hot encoding. [60]

Another way to extend the scheme is to allow multiple labels that are not mutually exclusive. The classification question could be "Which of the listed animals, if any, are present in the image?" Multi-label classification answers to n independent questions at the same time. Often these are binary questions and the output can be encoded with n neurons whose values are all independently between one and zero. It is also possible to further extend the output encoding to allow more complicated structures. [60]

We have argued that the structure of the output layer depends on the type of classification. In addition, the activation function of the output layer neurons and the classifier loss function must be chosen accordingly. In single-label binary classification the activation function is typically logistic sigmoid and the loss function is logistic loss. There are also many alternative activation and loss functions for this

case. In multiclass classification the natural choice is to use softmax activation and categorical cross entropy loss, which is an extension of logistic loss. Softmax is a function that squashes a vector of values in between zero and one so that their sum is one. It is mathematically defined as

$$\varphi_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \quad \text{for } i = 1, \ldots, n,$$

where $x_1, x_2, \ldots, x_n$ are the input values [3]. In binary multi-label classification logistic sigmoid can be used as the activation function for all the output neurons separately. The multi-label extension of logistic loss is called binary cross entropy loss. These activation and loss functions for different classification problems are listed in table 2. [21]

classification type	**output activation**	**loss function**
binary single-label	logistic sigmoid	logistic loss
multiclass	softmax	categorical cross entropy
binary multi-label	logistic sigmoid	binary cross entropy

Table 2: Common output layer activation functions and loss functions for different classification problems.

Choosing the activation and loss functions correctly is important. For example, if softmax and categorical cross entropy loss were used in a multi-label classification problem where the labels are actually independent, naïvely interpreting the output values would produce somewhat correlated but as a matter of fact meaningless results. The other way around, if mutually exclusive multiclass classifier used independent logistic sigmoid activations instead of softmax, the produced probability distribution would be invalid because the total probability would not be exactly one.

3.5 Overfitting and regularization

A supervised model should be able to repeat predictions that it was given in the training phase. If the model is too simple it might fail to do this because it is not be able to capture all the details of the training samples. This phenomenon is called underfitting. However, it is also very important that the model can generalize and make good predictions about new samples. It is not enough to remember all the details of the training samples since the number of training samples is in often limited and the samples may contain noise. By noise we mean random details in the features that do not represent the underlying properties of the data. When the model learns too much of this noise instead of the intended structure it is said to be overfitting. An example of this behavior is shown in figure 5. [6]

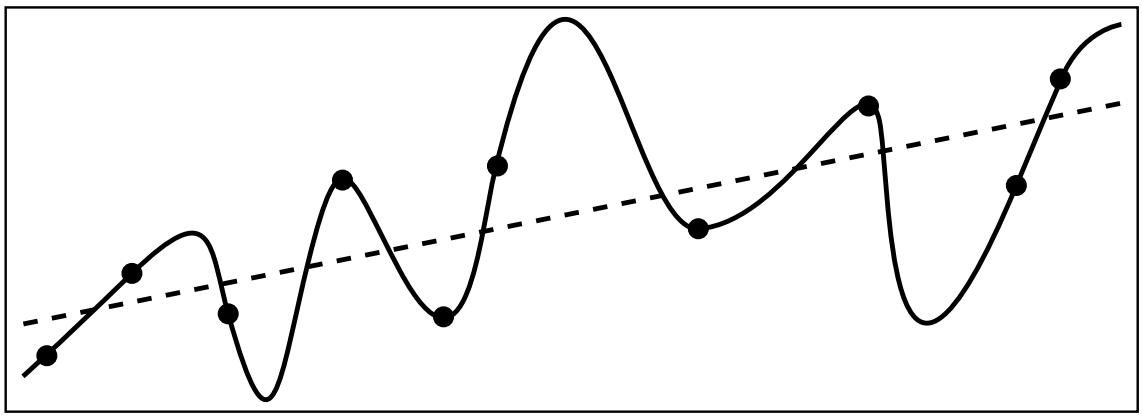

Figure 5: The solid line is a precise fit to the data points, but the dashed line may generalize better when making predictions.

It is beneficial to use a model structure that is so simple that it cannot learn all the noise in the data so it must instead approximate the samples it was given in order to minimize the loss function. This approximate often generalizes better producing good predictions for unknown samples. In neural network models a natural way to adjust the complexity and learning capacity is to choose the number of neurons accordingly. One way to prevent overfitting is to train multiple models and average the predictions. However, there are regularization techniques that are designed to prevent overfitting with less computation. [32]

Overfitting can be visualized using learning curves. These curves show validation loss as a function of training time. The validation loss of an ideal model would monotonically converge to a global minimum. If the model overfits the validation loss starts to increase after a point. Early stopping is a popular regularization method that leverages this behavior by stopping the training when the loss starts to go up. Figure 6 has learning curves for two example models. The dashed line converges and the solid line represents a model that overfits. [55]

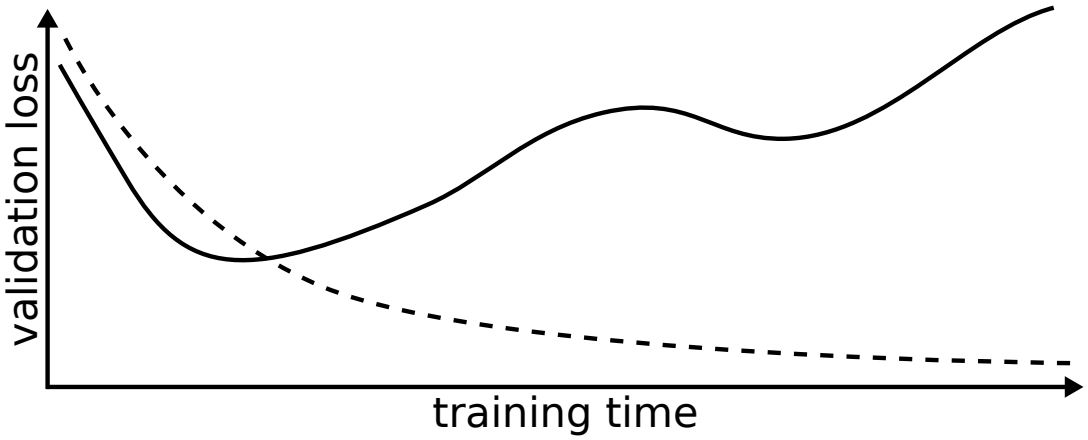

Figure 6: Learning curves for two models showing loss convergence and overfitting.

Dropout [57] is a commonly used regularization method for neural networks [18]. Dropout works by randomly dropping neurons from the network during training as illustrated in figure 7. This prevents the neurons from relying too much on each other and forces the network to create redundant paths. After the training phase all neurons are enabled. The output of single neurons and ultimately the output of the whole network is averaged over multiple paths making the results more robust against noise. [57]

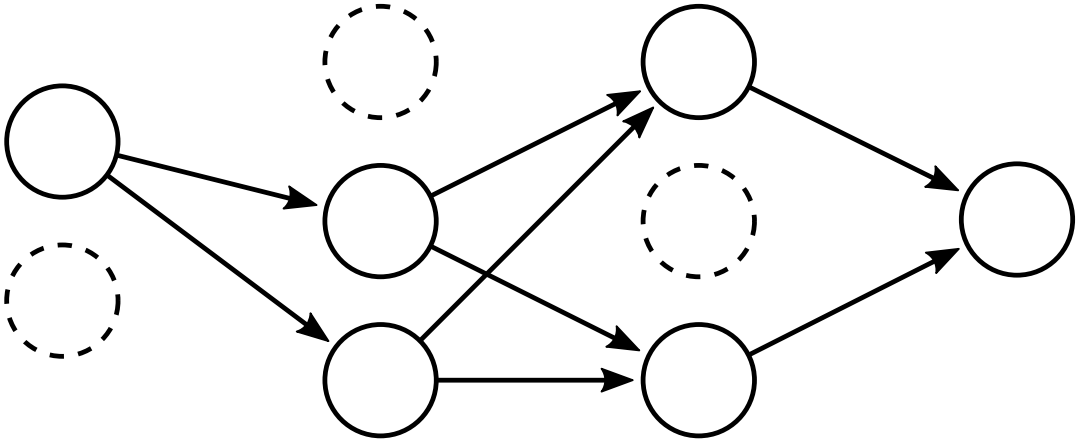

Figure 7: With dropout random neurons are disabled in the training phase.

3.6 Recurrent neural networks

Recurrent neural networks (RNN) [28] were first introduced by Hopfield in 1982. They gained popularity after Hochreiter and Schmidhuber discovered long short-term memory (LSTM) [26] networks in 1997. Long short-term memory networks outperform many other models in multiple fields including natural language text processing and speech recognition [16, 40]. Gated recurrent units (GRU) [9] have also been proved to have similar performance with LSTM networks.

Recurrent neural networks are applied to sequences like a stream of text, audio, or time series data points. In contrast, traditional neural networks must be applied to fixed-length vectors and cannot handle variable-length sequences. Recurrent neural networks are also able to output variable length sequences. Recurrent neural networks are based on recurrent neurons that extend the basic neuron model by adding connections to earlier neurons along the sequence. This effectively enables recurrent neurons to remember values over time. [22]

An LSTM neuron contains an input gate, a memory cell, a forget gate, and an output gate as shown in figure 8. The dashed lines represent connections along the sequence. The LSTM gates regulate how the information is stored and accessed from

the cell allowing the neuron to remember a value for a time it is useful but no longer than necessary. This structure tries to avoid the vanishing gradient problems that often limit the usefulness of recurrent neural network models [27]. An extension of the backpropagation algorithm, called backpropagation through time (BPTT) [63], can be used to efficiently train recurrent neural networks. [19]

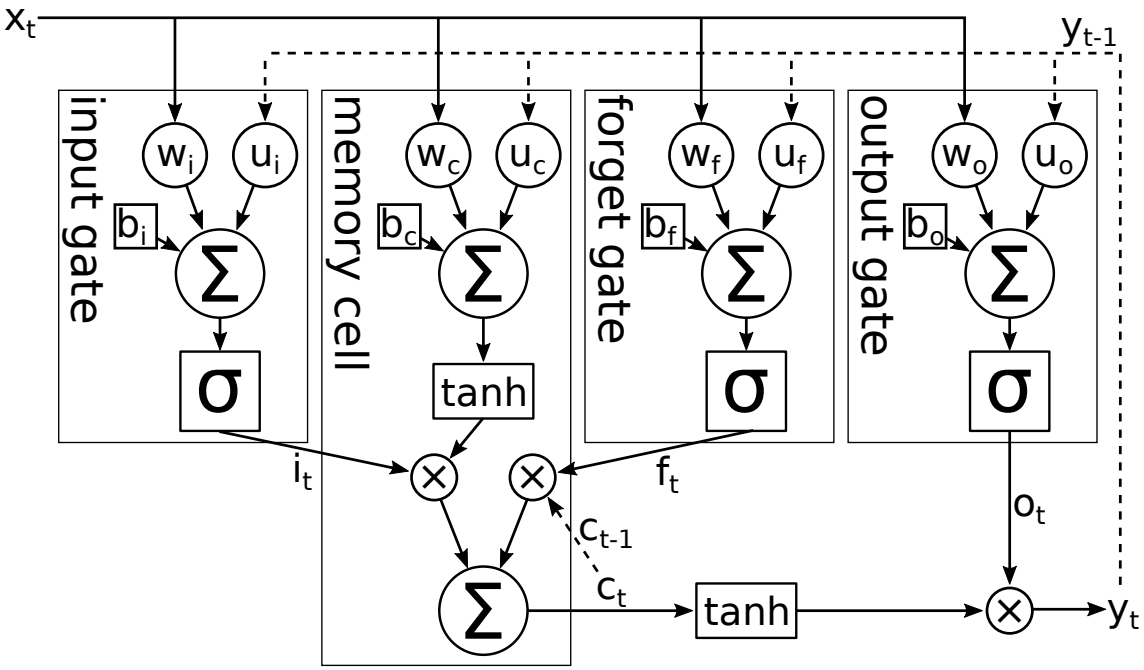

Figure 8: The structure of a long short-term memory neuron.

The forward pass of an LSTM neuron is defined by equations [19]:

$$
\begin{aligned}
i_t &= \sigma(w_i x_t + u_i y_{t-1} + b_i) \\
f_t &= \sigma(w_f x_t + u_f y_{t-1} + b_f) \\
o_t &= \sigma(w_o x_t + u_o y_{t-1} + b_o) \\
c_t &= i_t \tanh(w_c x_t + u_c y_{t-1} + b_c) + f_t c_{t-1} \\
y_t &= o_t \tanh(c_t),
\end{aligned}
$$

where x_t is the input vector, f_t, i_t, and o_t are the activation vectors of the forget gate, the input gate, and the output gate, respectively, c_t is the cell state vector, y_t is the output vector of the neuron, σ is the logistic sigmoid function, and w, u, and b are the weight matrices and the bias vectors.

4 Implemented method

We implemented a multi-label speaker recognition model that can identify known speakers in a given audio track. For each trained speaker the model outputs a probability of speaking at each time frame in the audio track. The output is visualized in figure 9. In this model it is possible that multiple speakers are speaking at the same time. The model is trained with an audio track and a synchronized binary label track for each speaker. The label track used in training has the same structure as the program output, but with only binary values at each time point as shown in figure 1 in the introduction chapter.

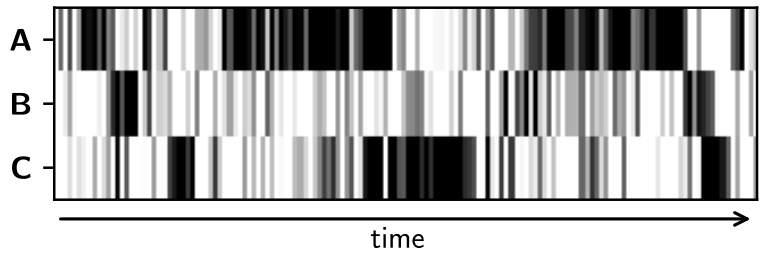

Figure 9: Program output with predicted probabilities shown as grayscale values.

The pipeline for making speaker activity predictions is shown in figure 11 (see the next page). The input audio track is given in PCM format. It is sliced to time frames of 20 milliseconds and 18 MFCC audio feature values are calculated for each frame. These MFCC features frames are given to the classifier model and it will output a probability vector for each frame. The probabilities represent the likelyhood of each speaker being active during the frame. With an ideal classifier the output probabilities would mimic the binary ground truth labels where each speaker is either speaking or not on each time frame. However, in practice the output contains time frames with uncertain predictions which may be incorrect.

In training phase (figure 10) the pipeline has the same preprocessing and feature extraction stages, but instead of making predictions in the classification stage the model is given both the MFCC features vectors and the ground truth labels. The

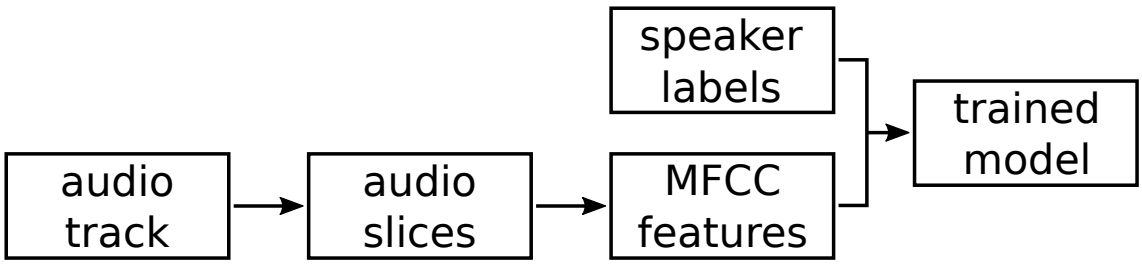

Figure 10: Training the classifier model.

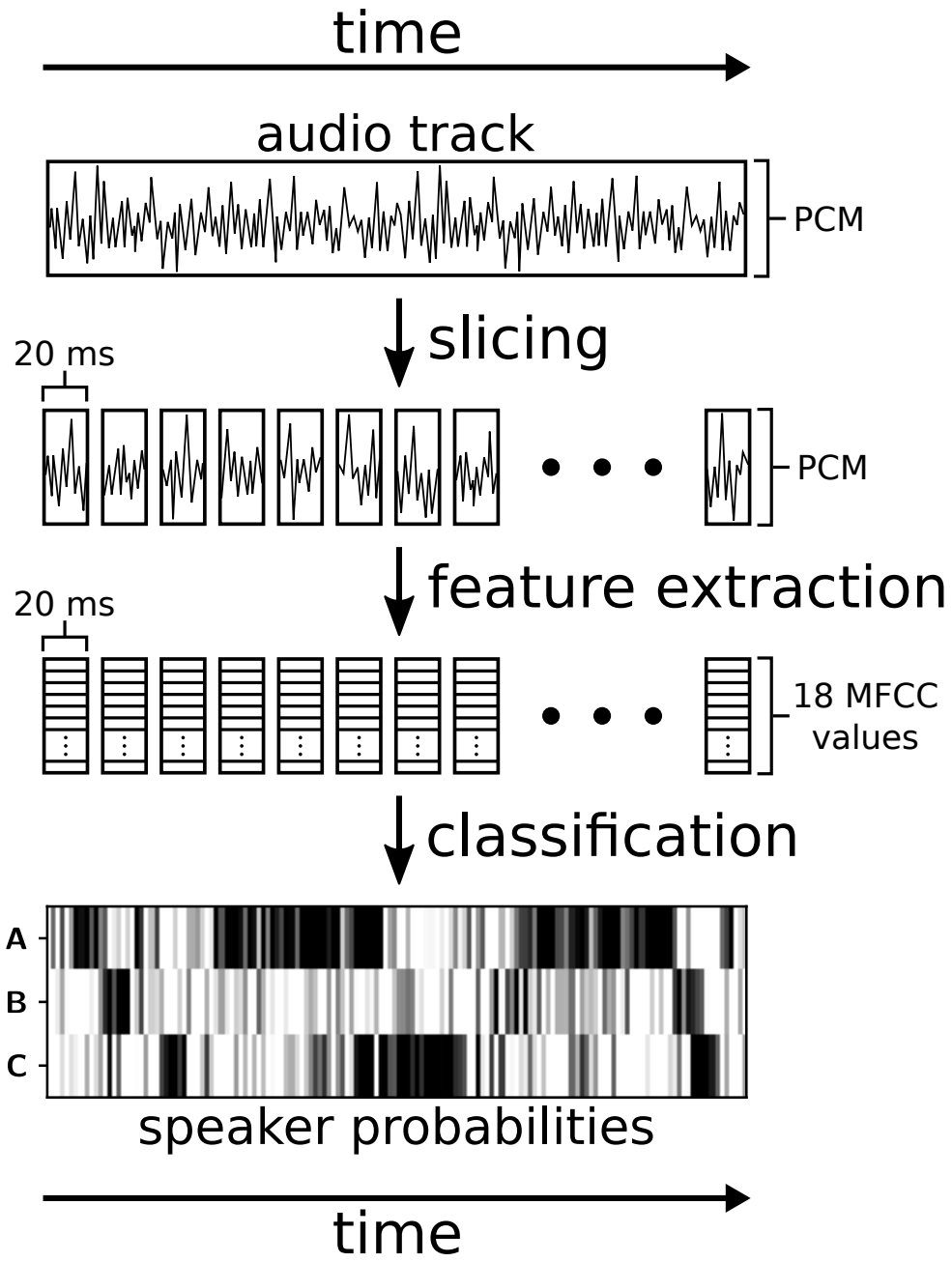

Figure 11: Pipeline for making speaker activity predictions on a given audio track.

model will try to fit its internal parameters so that it will be able to make useful predictions.

The classifier model, as shown in figure 12, is a neural network with LSTM layers and a densely connected output layer. The MFCC feature frames are given as the input sequence for the first LSTM layer. The number and dimensionality of the LSTM layers can be adjusted. The LSTM layers also have an adjustable dropout parameter.

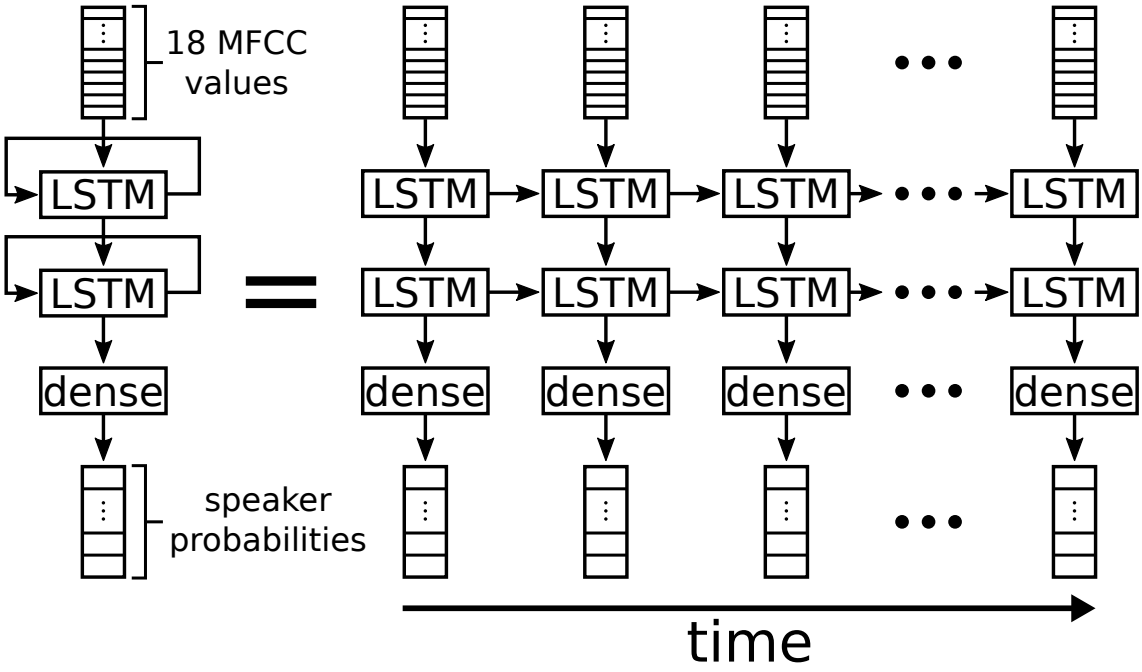

Figure 12: The structure of the classifier model with two LSTM layers.

Our model can use a look-ahead buffer to delay the network evaluation and feed the input signal to the model so that it can use some future samples to make the prediction. The length of the look-ahead buffer can be adjusted. It does not need to be very long and so it would only add a short delay when doing real-time prediction.

We have implemented the model in Python using Keras [10], an open-source neural network library. MFCC audio features are generated with `python_speech_features` library [38].

5 Evaluation

In this chapter we present the datasets and evaluation metrics we used and then analyze the performance of our model with different hyperparameter combinations.

5.1 Datasets

Our main dataset for model development is AMI Meeting Corpus [8]. It is a publicly available, Creative Commons licensed set of recorded meetings with multiple audio tracks along with various additional signals and annotations. The meetings were recorded in English by native and non-native speakers in acoustically different rooms using various kinds of microphones. Our interest is mainly on the high quality synchronized transcript that we can use to train and test speaker recognition models.

AMI Meeting Corpus consists of recording sessions. The sessions have unique identifiers, for example ES2016. Each session has a fixed set of participants and meetings they recorded. Typically a session has four speakers in four meetings. In the dataset each speaker have been assigned a unique identifier, for example MEE016 or FIE073. The first letter in the identifier corresponds to the gender of the speaker (M=male, F=female). These identifiers are shown in some of the result visualizations later. Each session has a new group of participants, but some participants have joined multiple sessions. Most of the meetings are simulated for the dataset, but with the aim to have natural, uncontrolled conversations. [8]

Each meeting in the dataset is recorded with multiple sets of different microphones. The dataset includes individual audio tracks for each speaker that could be used for speaker diarization based on the track volumes alone. However, they are not used in this thesis, because we are interested in a classifier that can distinguish different speakers from a single audio track. To avoid fitting to microphone locations, which may affect the volume and other details of the voice in the audio track, we use a different set of microphones in the training phase. Training uses sound mixed from lapel microphones and evaluation uses mixed headset microphone tracks.

We train our speaker recognition model for each group of speakers separately. Out of the four meetings in a session, three are used for training and the remaining one for validation. Our goal is to find a generalized model and hyperparameter values that perform well with all groups of speakers. We have reserved some sessions for model selection and hyperparameter adjusting and the remaining ones are used for testing.

The transcript annotations in the dataset are in XML based formats that must be processed and combined in order to have useful speaker labeling for our case.

5.2 Evaluation metrics

[TODO]

5.3 Results

Our classifier model is not remarkably deep or otherwise complex. Yet there are still many tweakable hyperparameters that have significant impact on the classification result. The main hyperparameters are the number of stacked LSTM layers and the size of those layers. As discussed in chapter 3, larger networks are prone to overfitting. To combat this, our third major hyperparameter is dropout. In this part of the thesis we are evaluating the different values for these hyperparameters to see which combinations works best. The hyperparameters to be tested and the values we are going to choose from are listed in table 3.

hyperparameter	**examined values**
number of LSTM layers	1, 2
size of LSTM layers	16, 64, 256, 1024
dropout	0 % (disabled), 50 %

Table 3: Examined hyperparameters and their values.

The number of training steps can also be seen as a hyperparameter for the model. It is expected that good classification results require a certain amount of training steps. In an ideal case the classification accuracy would improve over time and converge to some level. However, mainly due to overfitting, the accuracy may start to decrease at some point with more training. We are trying to analyze these behaviors with learning curves where classification accuracy is plotted as a function of training epochs.

First we examine a model with one LSTM layer and no dropout. The layer size is varied to see how it affects the results. In figure 13 there are two loss curves for each tested LSTM layer size. The dashed lines represent training losses and the solid lines validation losses. Initially the losses decrease rapidly, but after around epoch 5 the validation loss for all models except the one with layer size 16 start to increase while training losses continue to decrease. This most likely indicates overfitting. In figure 14 there is ROC AUC score curve for each tested layer size. We can see

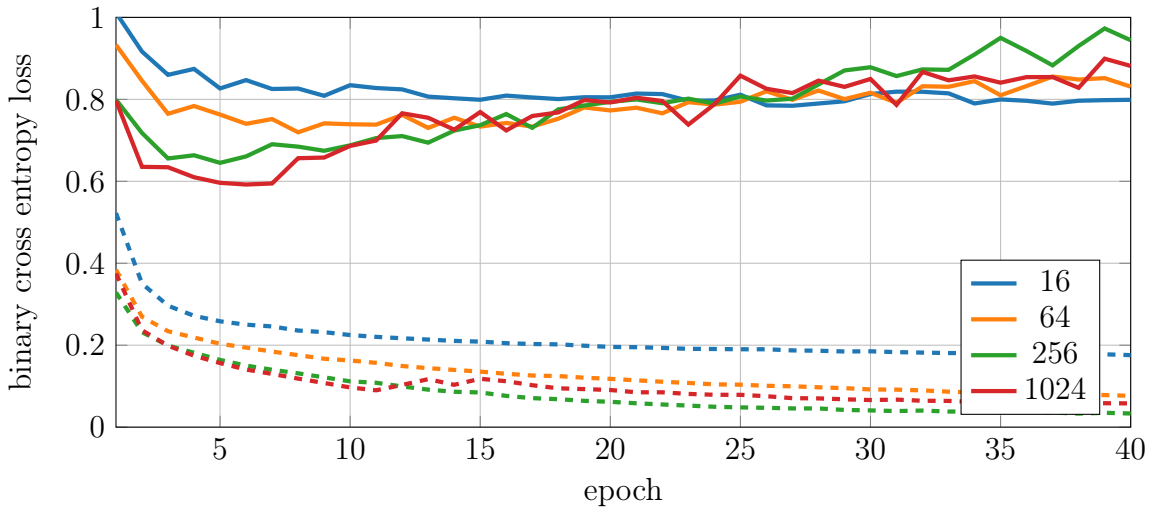

Figure 13: The effect of layer size on training loss (dashed lines) and validation loss (solid lines).

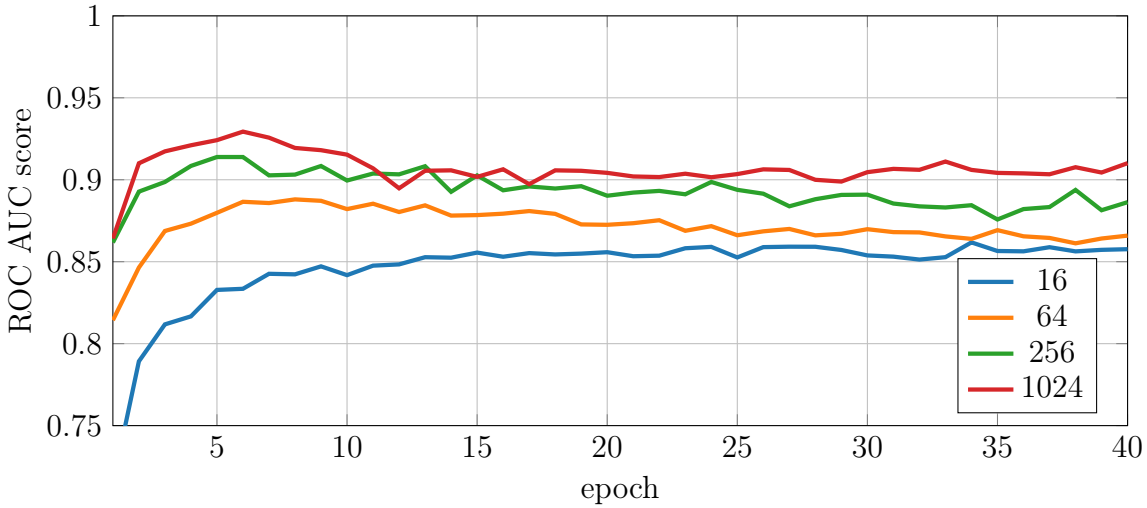

Figure 14: The effect of layer size on ROC AUC score.

that scores increase on the very first epochs. On layer size 16 the score continues to increase slowly, but the score of the models with greater layer sizes start to decrease.

Next, we test how adding dropout changes the results. Dropout may reduce overfitting allowing us to use larger network sizes. Figure 15 has learning curves with training and validation losses for models with layer size 256 and dropout both enabled and disabled. Shaded regions are one standard deviation error bands for the variance between meetings. Enabling dropout increased training loss. As discussed earlier in chapter 3, this is only an artifact caused by dropout itself. Dropout makes fitting to training data more difficult by design. Validation loss and other metrics should be used when comparing the trained models. The model with dropout has lower validation loss on all epochs compared to the model without dropout. The curve with dropout is converging and has less variance. In contrast, the validation

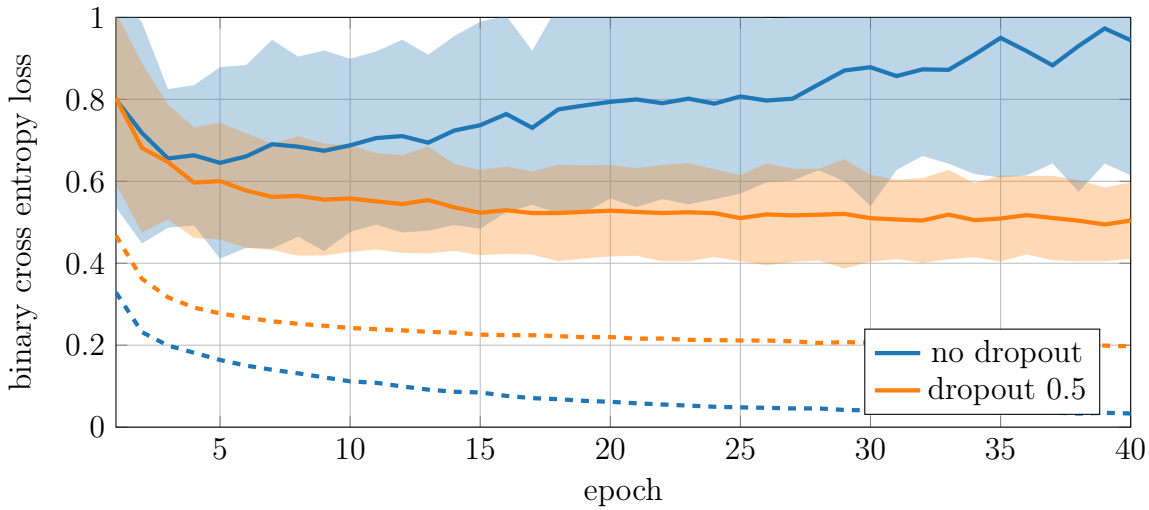

Figure 15: The effect of dropout on training and validation losses (LSTM, layer size 256).

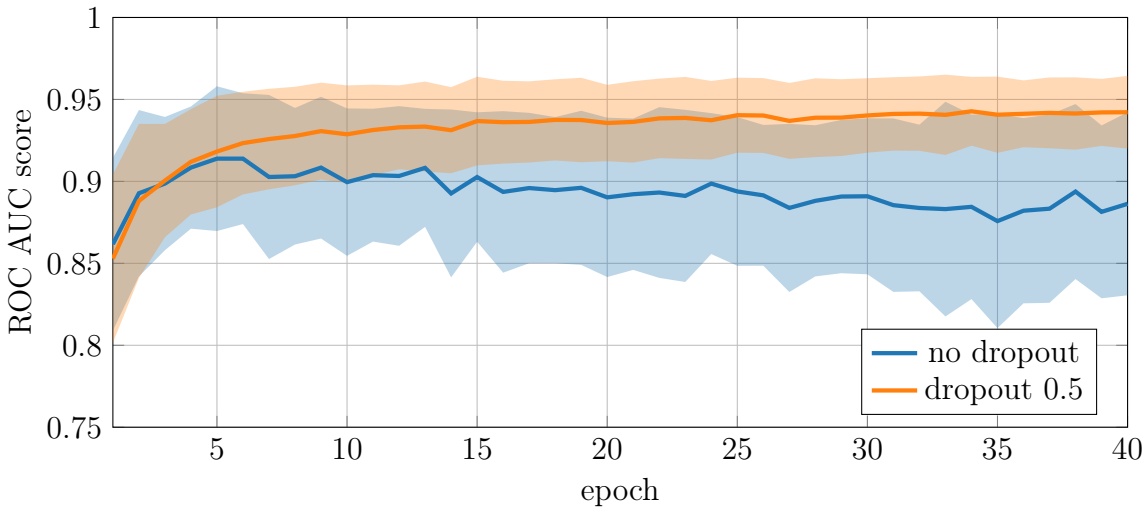

Figure 16: The effect of dropout on ROC AUC score (LSTM, layer size 256).

loss for the model without dropout has significantly more variance and starts to increase after around 5 epochs of training showing overfitting. At this layer size these observations would support selecting the model with dropout over the one without. We can futher see the difference in figure 16 where ROC AUC score is compared.

To find the best hyperparameter combination, we now test more layer sizes with dropout enabled. Figure 17 shows how varying the layer size affects training and validation losses when dropout is enabled. Comparing to figure 13 it can be noticed that with dropout the loss curves do not start to increase anymore. ROC AUC scores with dropout enabled in figure 18 are converging and reaching better values than without dropout in figure 14 where most of the scores started to decrease. It can be seen that dropout is useful for reducing overfitting and making the models more stable.

From the tested layer sizes it would appear that 256 and 1024 are the best models

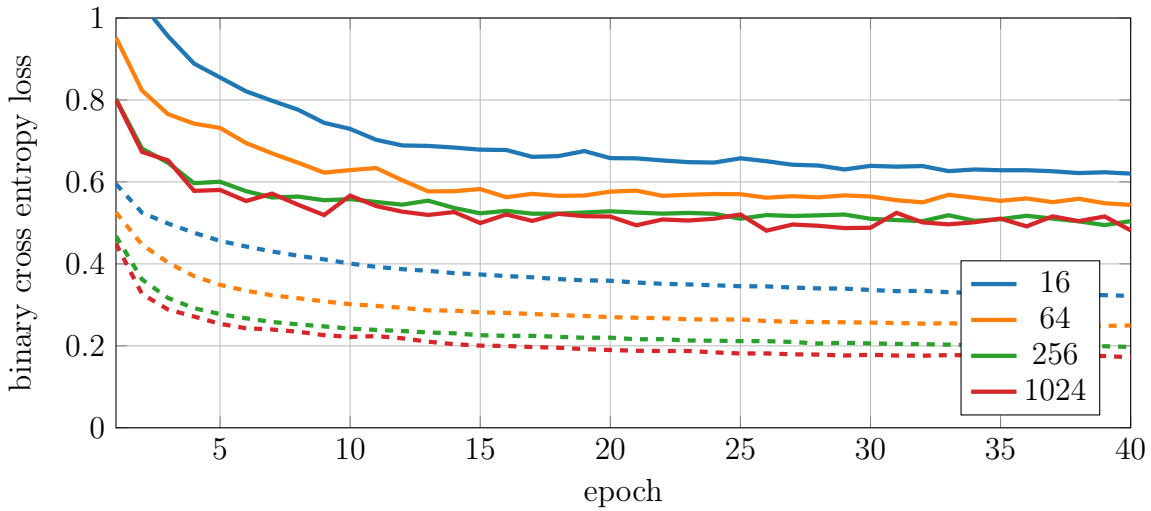

Figure 17: The effect of layer size on training and validation loss with dropout enabled.

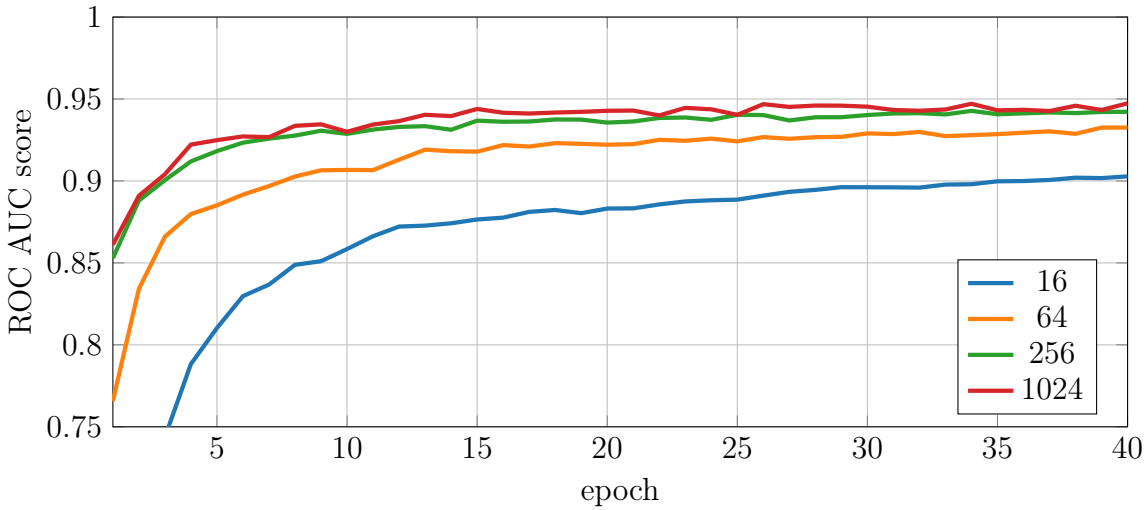

Figure 18: The effect of layer size on ROC AUC score with dropout enabled.

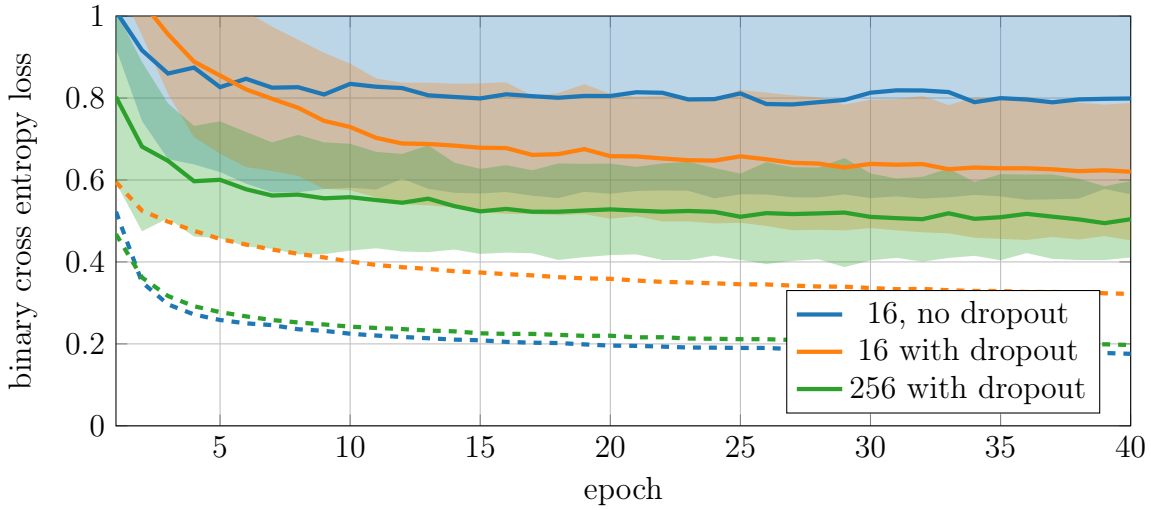

Figure 19: Training and validation losses comparison of some models with dropout enabled and disabled.

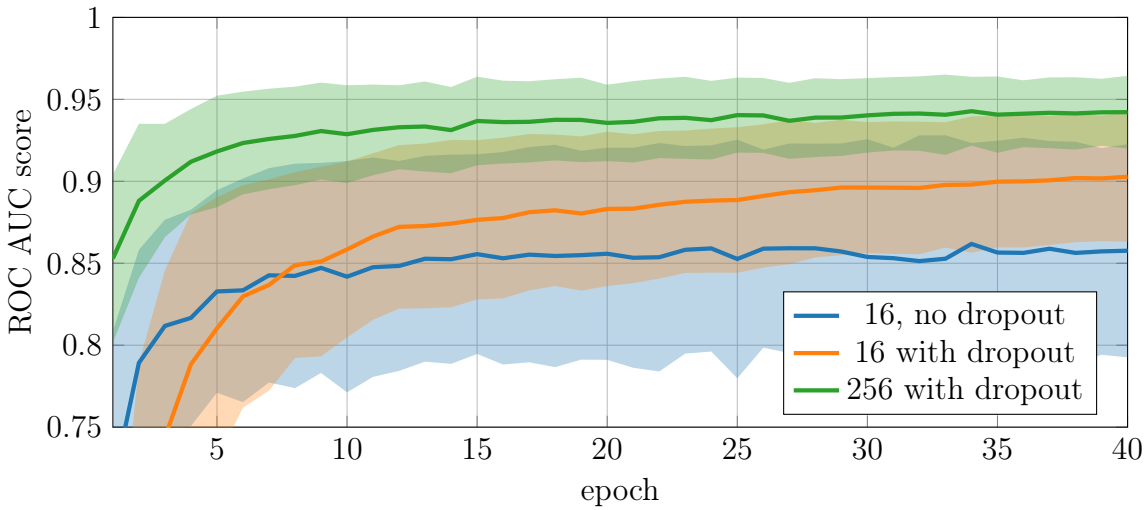

Figure 20: ROC AUC score comparison of some models with dropout enabled and disabled.

on both validation loss and ROC AUC score. Layer size 1024 may have marginally better results, but the difference is very small. Layer size 64 is not far behind, but size 16 does not seem to be large enough to fully capture the underlying phenomena. However, even with the layer size 16 the loss and score values are better with dropout than without. This can be seen in figures 19 and 20 where the model without dropout (layer size 16) is compared to models with dropout enabled. At layer size 16 the model learns faster without dropout but the model with dropout enabled will reach and exceed the performance with further training.

The predictions made by our classification system are shown in figures 21 and 22. The figures also have the ground truth labels for comparison. The x-axis is time in

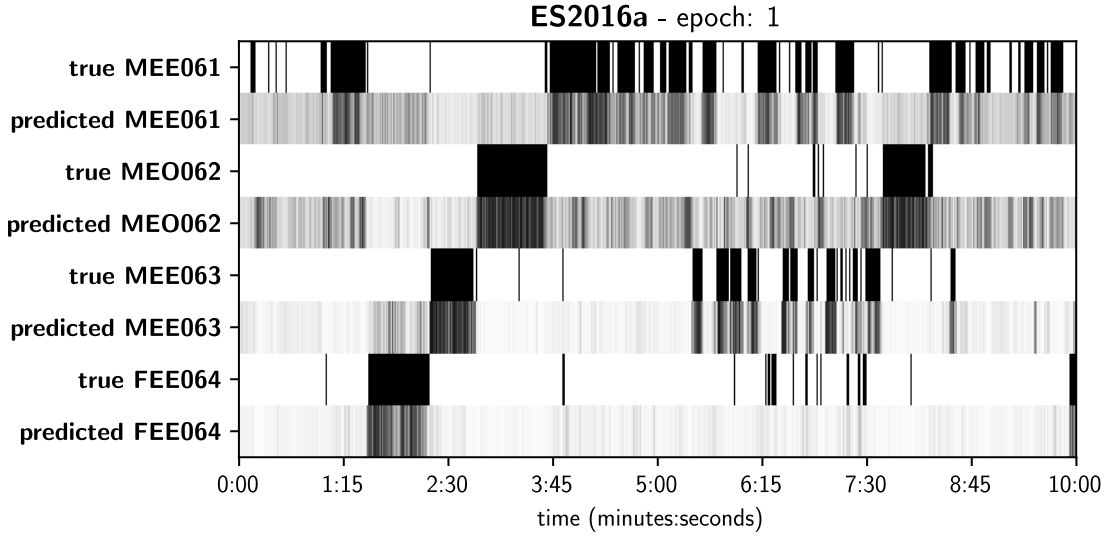

Figure 21: Predictions compared to ground truth labels after only one epoch of training (layer size 256, dropout).

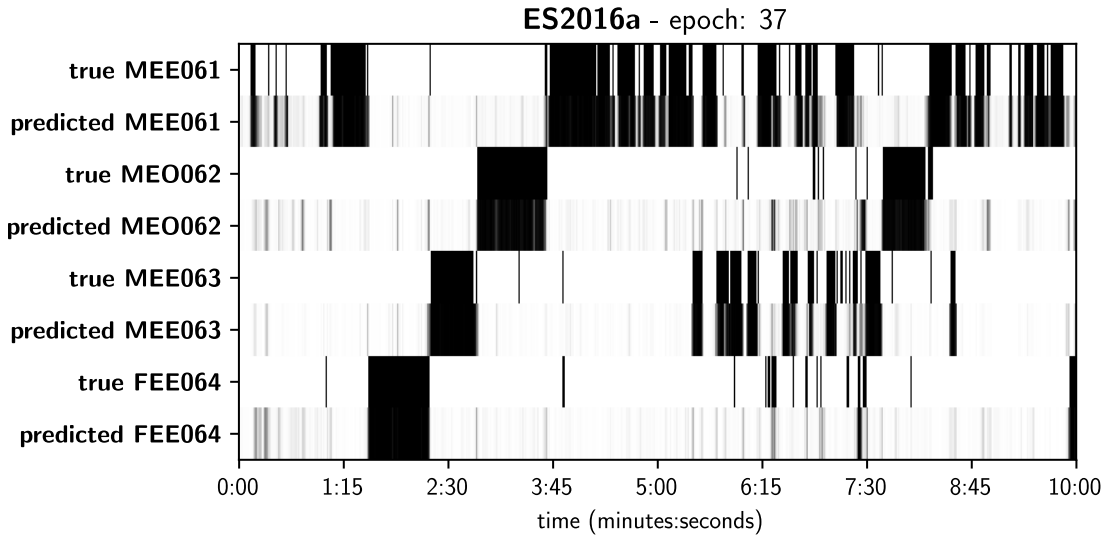

Figure 22: Predictions after 37 epochs of training (layer size 256, dropout).

the audio track starting from the beginning of meeting. The meetings are longer, but for visualization reasons these figures are limited to the first ten minutes. The predictions in figure 21 are made after only one epoch of training. At that stage the classifier is still underfitted and the predictions are mostly uncertain and include lots of random noise. Figure 22 has the same classifier and configuration, but after 37 epoch of training at the peak accuracy. The prediction signal is now significantly stronger. There are still some incorrect predictions, but most of the speaking segments are correctly identified. For comparison, figure 23 has the predictions for the same meeting made by a classifier model with layer size 16 and dropout disabled. This figure looks similar, but especially the predictions for MEO062 and FEE064 have more mistakes and this classifier is incorrectly more confident about them compared to the mistakes in figure 22.

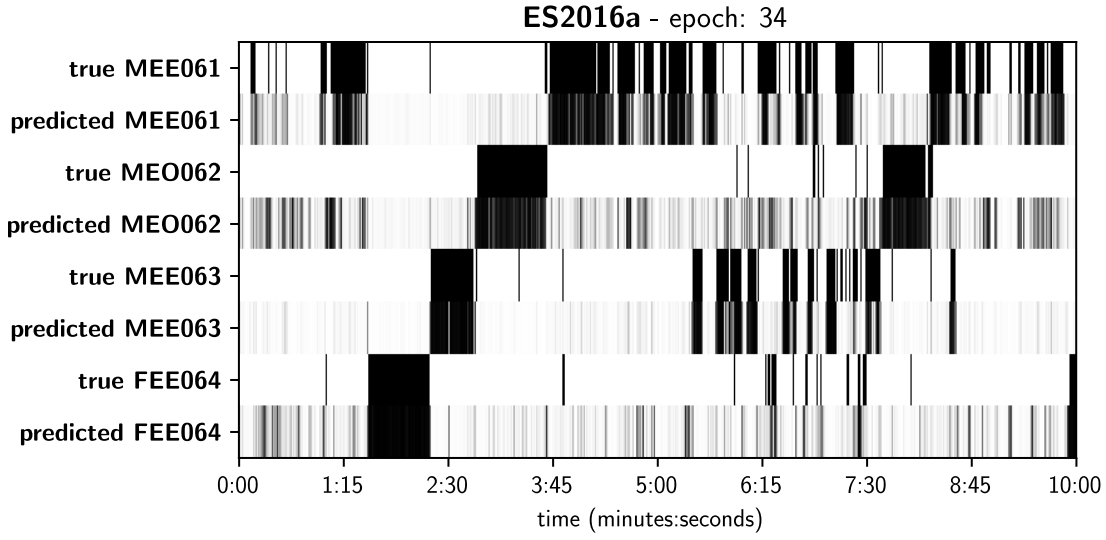

Figure 23: Predictions made by a model with layer size 16 and no dropout.

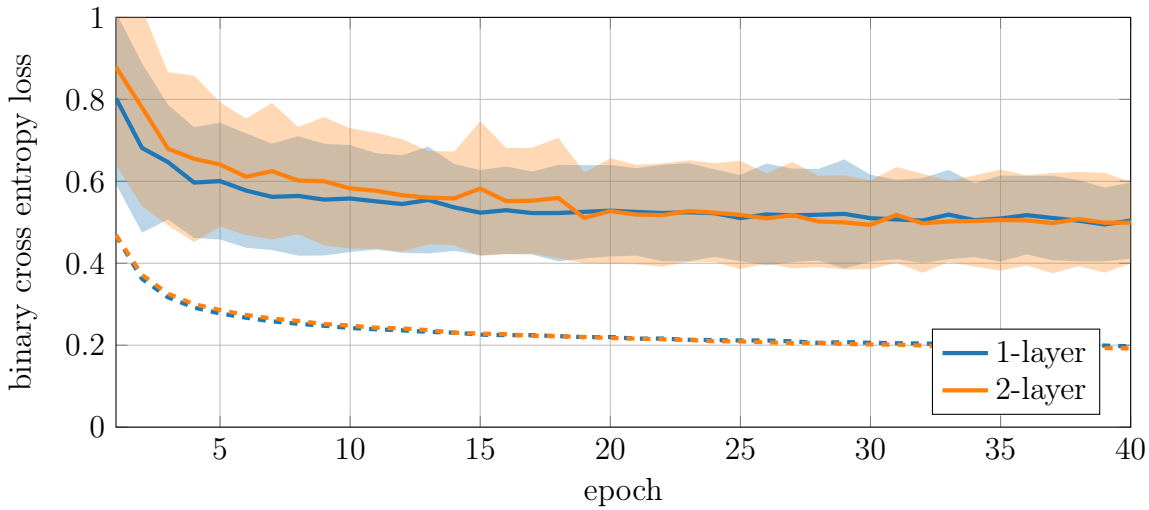

Figure 24: Loss comparison of 1-layer and 2-layer models with layer size 256 and dropout.

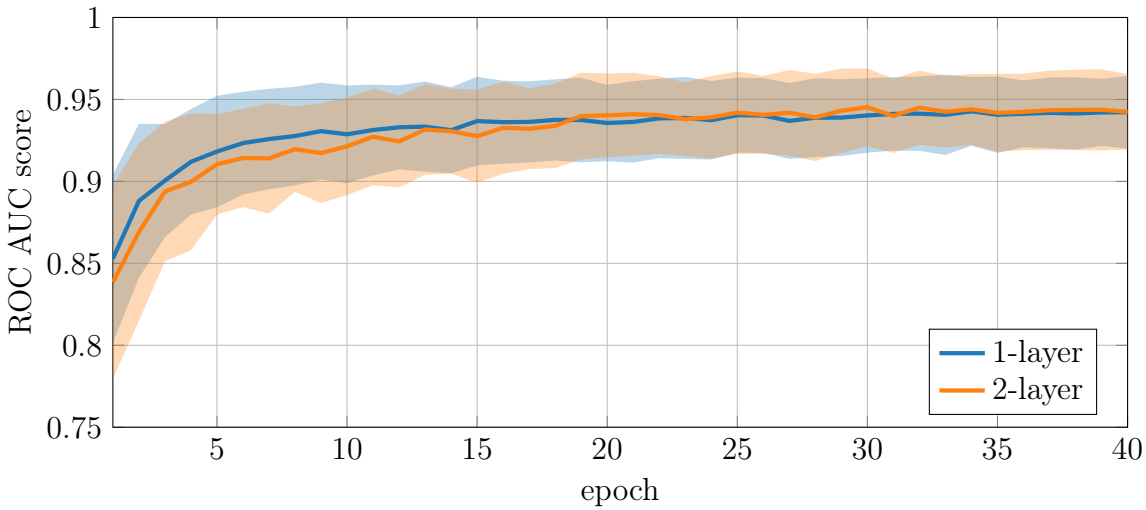

Figure 25: ROC AUC comparison of 1-layer and 2-layer models with layer size 256 and dropout.

Stacking another LSTM layer to the network model is a way to increase the learning capacity. Figures 24 and 25 compare 1-layer and 2-layer models with layer size 256 and dropout enabled. The 2-layer model performs almost exactly as well as the 1-layer model with same parameters. The simpler 1-layer model learn a bit faster, but with more training the 2-layer model reaches the same results. The 2-layer model may be marginally better, but the difference between the results is not clear. It seems that the 1-layer model at this layer size is already complex enough to accurately fit the problem.

To see if adding a second hidden layer increases learning capacity we compare 1-layer and 2-layer models with layer size 16 and dropout disabled. The loss and ROC AUC score learning curves are shown in figures 26 and 27. It can be seen that the 2-layer model performs better with these parameters. Again, the 1-layer model

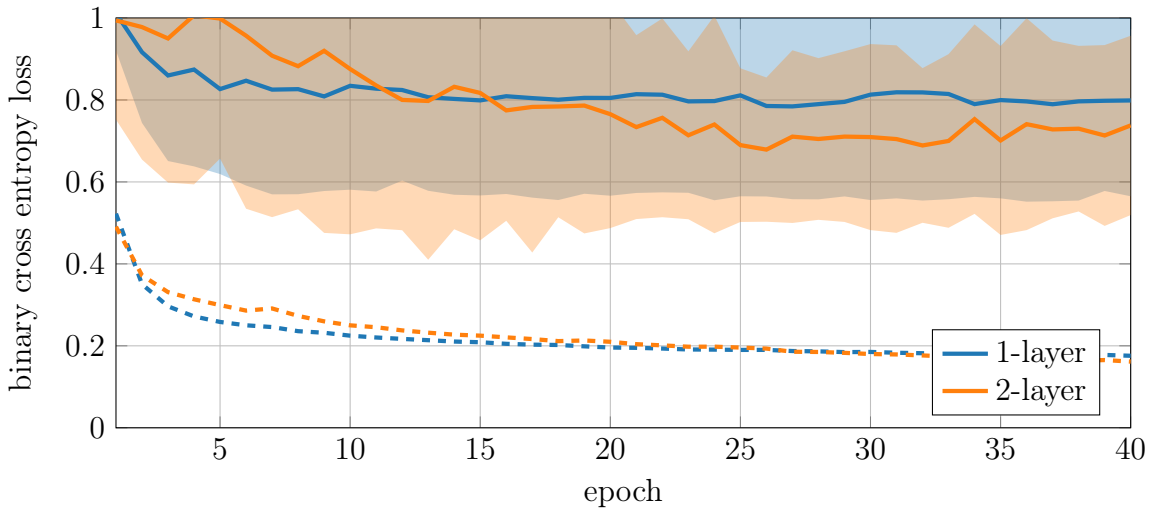

Figure 26: Loss comparison of 1-layer and 2-layer models with layer size 16.

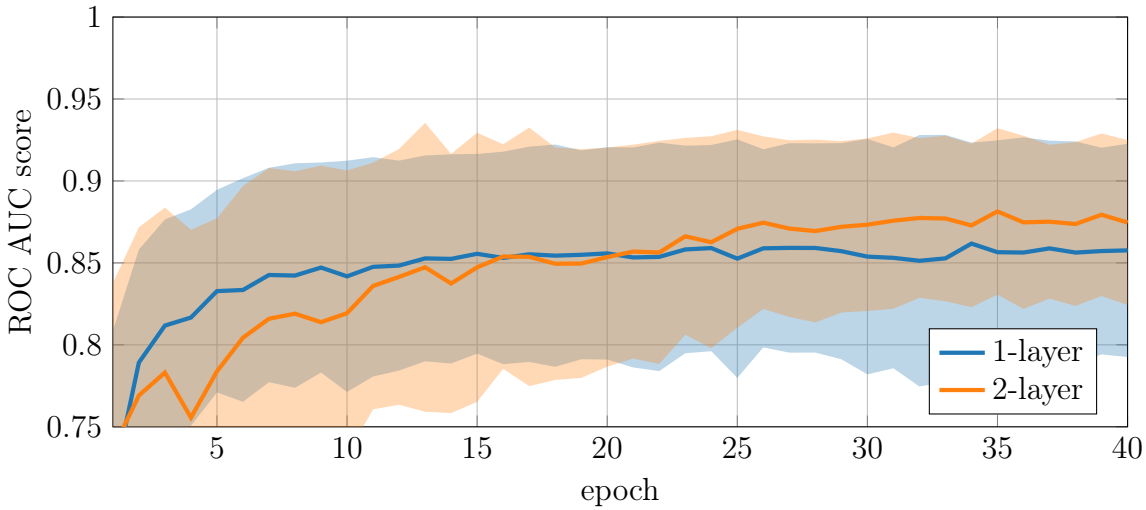

Figure 27: ROC AUC comparison of 1-layer and 2-layer models with layer size 16.

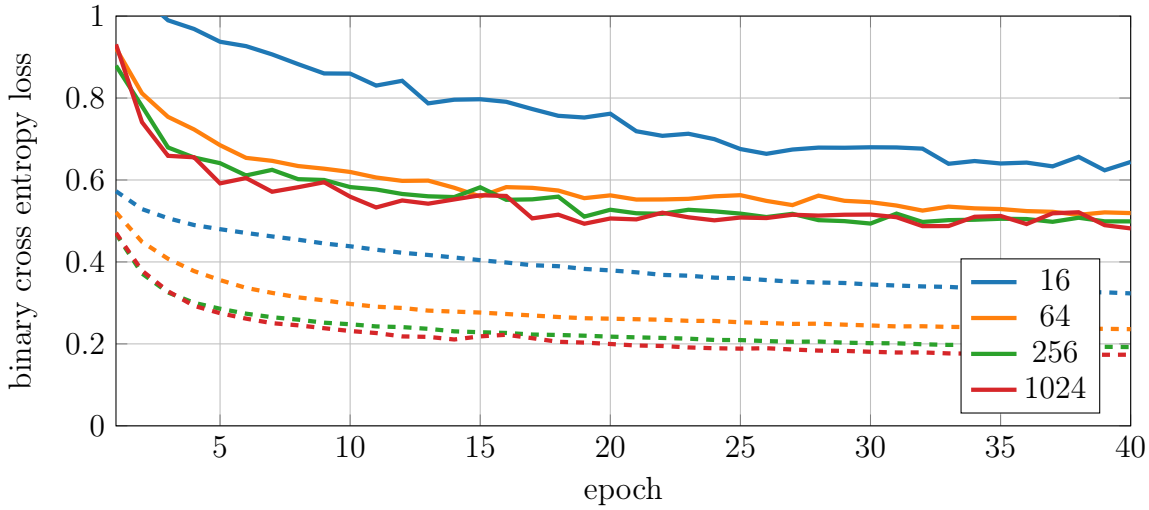

Figure 28: The effect of layer size on training and validation loss with 2-layer models and dropout enabled.

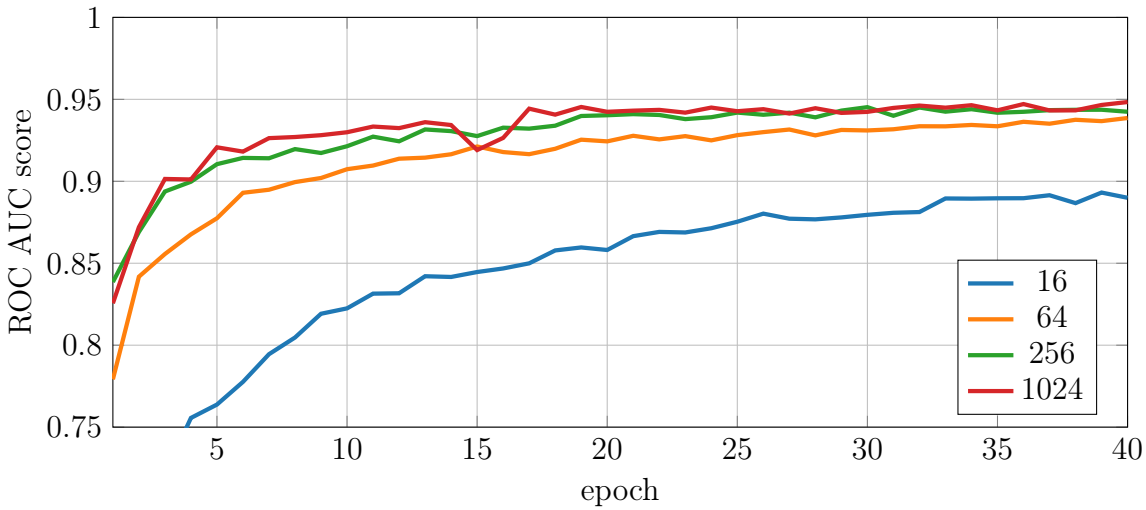

Figure 29: The effect of layer size on ROC AUC score with 2-layer models and dropout enabled.

learns initially faster, but the 2-layer model achieves better results later. The 2-layer model also has less variance in its results.

To further compare the parameter combinations, we test 2-layer models with each layer size. The learning curves for each 2-layer model with different layer sizes are shown in figure 28 and 29. These curves look very similar to the 1-layer model curves in figures 17 and 18. Layer sizes 256 and 1024 still the best ones. Seems that the 2-layer models take more training time to reach the same results and there is no visible benefit in having 2 LSTM layers.

Different models have different computational costs. Table 4 shows how training speed differs for our models. The training speed depends on the underlying hardware setup. In our case the measurements were done on a system with Intel Core i5-4670K processor and GeForce GTX 1060 (6GB) graphics processing unit. We can see that models with layer size 1024 or any 2-layer model takes significantly more time for training than the simpler models. The 2-layer model with layer size 1024 takes about ten times the training time compared to the smaller 1-layer models. This gives us reason to select the 1-layer model with layer size 256 since the more complex models did not achieve better accuracy results.

training time per epoch (s)

	1 layer	**2 layers**
layer size 16	35.1 ± 0.78	73.8 ± 3.50
layer size 64	36.7 ± 1.94	72.1 ± 2.68
layer size 256	35.0 ± 1.01	74.0 ± 1.84
layer size 1024	121.9 ± 5.22	340.1 ± 13.89

Table 4: Training time for different models (dropout enabled).

Up to this point we have used a validation dataset that combine multiple recording sessions. Using more data is good for achieving generalization as it is not desired to optimize the system for any single case. The meetings in the recording sessions may have differences that affect the classification performance. If we examine predictions made for some specific session the results may be deviated from the averages shown before. The variance in the results in different sessions can be analyzed by calculating the metrics for each of them separately. This is done in figure 30 where ROC AUC scores are shown separately for five sessions. The results on some sessions are significantly better compared to others. We can say that, for our classification models, the speaking patterns in IS1008 seem to be easier to classify than in IS1000. The differences can also been seen by comparing the labels and predictions for meetings ES2016a (figure 22), IS1008a (figure 31), and IS1000a (figure 32). No further

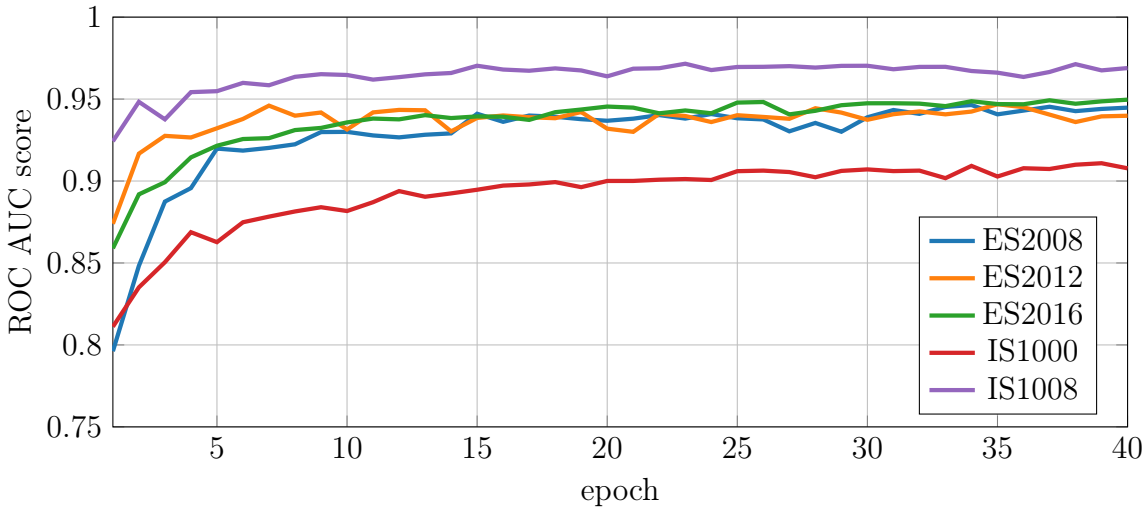

Figure 30: ROC AUC score for individual validation datasets (LSTM, layer size 256, dropout).

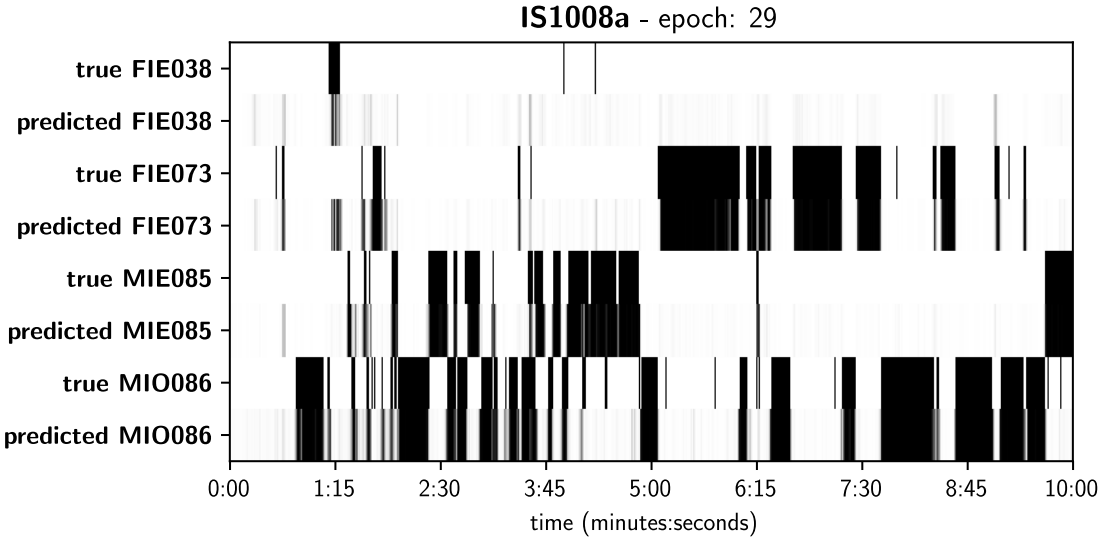

Figure 31: Predictions for meeting IS1008a (layer size 256, dropout).

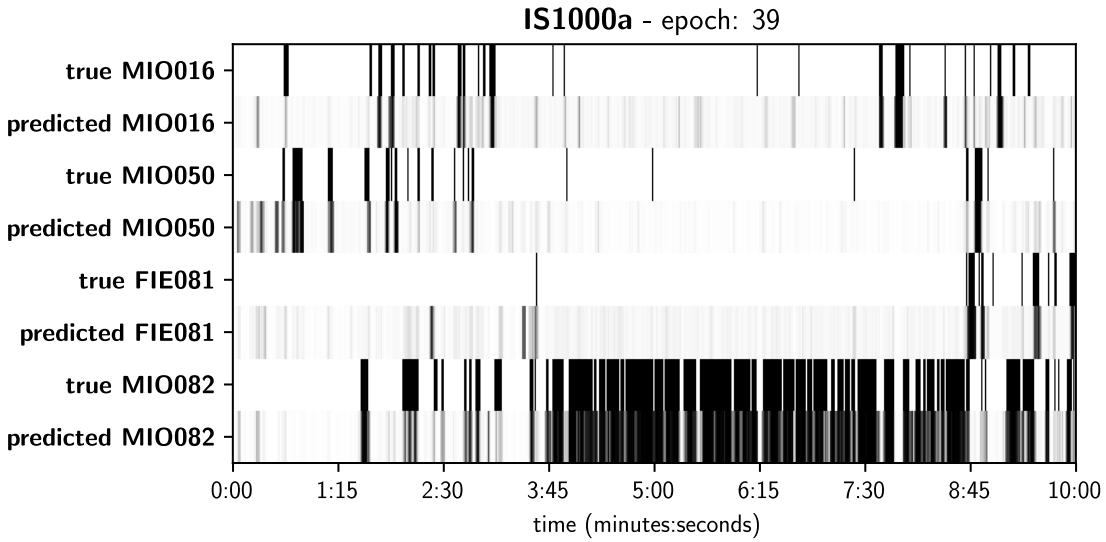

Figure 32: Predictions for meeting IS1000a (layer size 256, dropout).

analysis on the actual differences in the data is done in this thesis.

So far we have used our validation data sets for model selection. While the validation data was not available to the network optimizer itself, it is possible that by choosing the best model structure and hyperparameters we have overfitted to the validation data. We can compare the results to a new, unused test set. The best model chosen for testing is a 1-layer LSTM network with layer size 256 and dropout enabled. Table 5 has evaluation results for both the validation set and test set.

	cross entropy loss	**ROC AUC score**	**F1 score**
validation set	0.504 ± 0.1867	0.942 ± 0.0444	0.763 ± 0.1486
test set	...	...	...

Table 5: Comparison of validation and test results (layer size 256, dropout, epoch 40).

We can also compare how the predictions made for the testing set look. Figure **??** shows predictions made for meeting XYZ1234 in our testing set. The meeting is different so we cannot directly compare it to the meetings showed earlier in figures 22, 31, and 32, but it can be seen that the predictions are around the same level of quality as the earlier ones made with the same set of hyperparameters.

6 Conclusion

This thesis introduced the reader to the theory that is needed to understand our experimental speaker recognition system. We were able to develop a recurrent neural network classifier that succesfully performs multi-label speaker recognition on AMI Meeting Corpus data set. Based on this fact, we can conclude that recurrent neural networks can be used to perform speaker recognition. We can also see that a rather simple long short-term memory (LSTM) network using MFCC features is sufficient for this task. We compared different hyperparameters and noticed that by choosing good values for them can improve the classification results and reduce computational requirements. The best model was a 1-layer LSTM network with layer size 256. More complex models significantly increase the computation cost, but do not improve the results.

One source of uncertainty in the measured results is the inaccuracies in the speaker label annotations. AMI Meeting Corpus was chosen because the annotations seem to be very good, but we cannot expect them to be absolutely correct. We have also limited the speaker label time resolution to 20 milliseconds.

The main limitation of our implementation is that it only recognizes known speakers that were present in the training material. This is caused by our supervised classification architecture. Unknown speakers could be supported with unsupervised learning that only clusters similar utterances together without actually identifying the speakers. However, the supervised approach may be better if the goal is to only recognize few designated speakers for whom we have speech samples. Further research could be done on determining the required amount of training samples per speaker and on minimizing that amount.

6.1 Future work

We used only the AMI Meeting Corpus dataset in our experiments. Further testing could be done with other datasets that do not need to be limited to meeting recordings. In addition, existing data could be augmented to increase the amount of training samples. Possible augmentation include adding noise, distortions, or additional background sounds to the audio track. Artificial meeting scenarios could be created by mixing and matching excerpts from the available material. This would include combining various speakers from different meetings and possible creating new meetings that have more participants than any meeting in the source material.

Our classifier works on audio streams that would allow real-time speaker recognition

on a live audio source, but we did not cover this aspect. Our hypothesis is that real-time evaluation would work without extensive modification to the system. If evaluation performance turns out to be a problem, gated recurrent units (GRU) may be able to replace our long short-term memory (LSTM) layers using much less computation time as for example Khandelwal et al. found out in their speech recognition experiments [30].

Alternative classification layers, like the gated recurrent units, could also outperform our system in terms of accuracy. Other methods including bidirectional recurrent neural networks and convolutional neural networks (CNN) [33] could also be tested. Convolutional neural networks might also replace our MFCC based feature extraction layer. Lukic et al. studied this arguing that the commonly used MFCC feature extraction methods do not utilize all the available speaker information in the audio [37]. They also researched speaker recognition as a clustering problem on the latent space of their CNN, which makes it possible to segment speech from unknown speakers.

References

[1] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland and O. Vinyals. Speaker diarization: A review of recent research. 2012. IEEE Transactions on Audio, Speech, and Language Processing 20, no. 2, pp. 356–370.

[2] L.E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. 1966. The annals of mathematical statistics 37, no. 6. pp. 1554–1563.

[3] C.M. Bishop. Pattern recognition and machine learning. 2006. Springer Science.

[4] H.S. Black, and J.O. Edson. Pulse code modulation. 1947. Transactions of the American Institute of Electrical Engineers 66, no. 1. pp. 895–899.

[5] L.E. Boucheron and P.L. De Leon. On the inversion of mel-frequency cepstral coefficients for speech enhancement applications. 2008. IEEE Signals and Electronic Systems. ICSES'08. pp. 485–488.

[6] K. Burnham and D.R. Anderson. Model selection and multimodel inference: a practical information-theoretic approach. 2003. Springer Science & Business Media.

[7] J.P. Campbell. Speaker recognition: A tutorial. 1997. Proceedings of the IEEE 85, no. 9. pp. 1437–1462.

[8] J. Carletta. Announcing the AMI Meeting Corpus. 2006. The ELRA Newsletter 11(1), January-March, pp. 3–5.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014. arXiv:1406.1078.

[10] F. Chollet. Keras. 2015. `https://keras.io/`

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

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

[13] H. Dudley. Phonetic Pattern Recognition Vocoder for Narrow-Band Speech Transmission. 1958. The Journal of the Acoustical Society of America 30, no. 8. pp. 733–739.

[14] G. Fant. Acoustic theory of speech production. 1960. Mouton.

[15] K.R. Farrell, R.J. Mammone, and K.T. Assaleh. Speaker recognition using neural networks and conventional classifiers. 1994. IEEE Transactions on speech and audio processing 2, no. 1. pp. 194–205.

[16] S. Fernández, A. Graves, and J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. 2007. In International Conference on Artificial Neural Networks, pp. 220–229. Springer, Berlin, Heidelberg.

[17] K.R. Foster, R. Koprowski, and J.D. Skufca. Machine learning, medical diagnosis, and biomedical engineering research-commentary. 2014. Biomedical engineering online 13, no. 1. p. 94.

[18] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. 2016. In Advances in neural information processing systems. pp. 1019-1027.

[19] F.A. Gers, J. Schmidhuber, and J. Cummins. Learning to forget: Continual prediction with LSTM. 1999. Neural Computation, vol. 12, no. 10. pp. 2451–2471.

[20] X. Glorot, A. Border, and Y. Bengio. Deep sparse rectifier neural networks. 2011. Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 315–323.

[21] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. 2016. Cambridge, MIT press.

[22] A. Graves, A.R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. 2013. In Acoustics, speech and signal processing (ICASSP). IEEE. pp. 6645–6649.

[23] F. J. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. 1978. Proceedings of the IEEE 66.1, pp. 51–83.

[24] M. R. Hasan, M. Jamil, M. G. Rabbani, & M. S. Rahman. Speaker identification using mel frequency cepstral coefficients. 2004. International Conference on Electrical & Computer Engineering.

[25] S. Haykin. Neural networks: a comprehensive foundation. 1994. Prentice Hall PTR.

[26] S. Hochreiter, J. Schmidhuber. Long short-term memory. Neural Computation, Vol. 9, No. 12, 1997, pp. 1735–1780.

[27] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001. In A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press.

[28] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. 1982. Proceedings of the national academy of sciences 79, no. 8. pp. 2554–2558.

[29] K. Karhunen. Über lineare Methoden in der Wahrscheinlichkeitsrechnung. 1947. Sana, Vol 37.

[30] S. Khandelwal, B. Lecouteux, and L. Besacier. Comparing GRU and LSTM for Automatic Speech Recognition. 2016. Ph.D. dissertation, LIG.

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

[32] S. Lawrence, C.L. Giles and A.C. Tsoi. Lessons in neural network training: Overfitting may be harder than expected. 1997. In AAAI/IAAI. pp. 540–545.

[33] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. 1989. Neural computation 1, no. 4. pp 541–551.

[34] Y. Lei, N. Scheffer, L. Ferrer, and M. McLaren. A novel scheme for speaker recognition using a phonetically-aware deep neural network. 2014. Acoustics, Speech and Signal Processing (ICASSP). pp. 1695–1699.

[35] R.P. Lippmann. Review of neural networks for speech recognition. 1989. Neural computation 1, no. 1. pp. 1–38.

[36] B. Logan. Mel Frequency Cepstral Coefficients for Music Modeling. 2000. ISMIR. Vol. 270. pp. 1–11.

[37] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann. Speaker identification and clustering using convolutional neural networks. 2016. In 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), Vietri sul Mare, Italy, 13–16 September 2016. IEEE.

[38] J. Lyons. `python_speech_features`. `https://github.com/jameslyons/python_speech_features`

[39] E. MacAskill. "Did 'Jihadi John' kill Steven Sotloff?". 2 September 2014. The Guardian. https://www.theguardian.com/media/2014/sep/02/steven-sotloff-video-jihadi-john

[40] M. Mahoney. Large text compression benchmark. http://www.mattmahoney.net/dc/text.html

[41] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. 1985. Proceedings of the IEEE 73, no. 11. pp. 1551–1588.

[42] J. Markel and A. Gray Jr. Linear Prediction of Speech. 1976. Springer–Verlag.

[43] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. The bulletin of mathematical biophysics 5, no. 4. pp. 115–133.

[44] M. Mohri, A. Rostamizadeh, & A. Talwalkar. Foundations of machine learning. 2012. MIT press.

[45] L. Muda, M. Begam, & I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. 2010. Journal of Computing, Vol. 2, Issue 3, March.

[46] D. Nguyen-Tuong, and J. Peters. Model learning for robot control: a survey. 2011. Cognitive processing 12, no. 4. pp. 319–340.

[47] Nuance. Multimodal voice & behavioral biometric authentication technology. Retrieved 18 October 2018. https://www.nuance.com/omni-channel-customer-engagement/security/identification-and-verification.html

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

[49] D. Reynolds. Speaker identification and verification using Gaussian mixture speaker models. 1995. Speech communication, 17(1), pp. 91–108.

[50] D. Reynolds. Automatic speaker recognition using Gaussian mixture speaker models. 1995. The Lincoln Laboratory Journal.

[51] H. Robbins and S. Monro. A Stochastic Approximation Method. 1951. The Annals of Mathematical Statistics 22, no 3. pp. 400–407.

[52] R.C. Rose and D.A. Reynolds. Text independent speaker identification using automatic acoustic segmentation. 1990. Proc. ICASSP, vol. 90, pp. 293–296.

[53] D. O'Shaughnessy. Speech communication: human and machine. 1987. Universities press. p. 150.

[54] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. 1978. IEEE transactions on acoustics, speech, and signal processing 26, no. 1. pp. 43–49.

[55] W.S. Sarle. Stopped training and other remedies for overfitting. 1996. Computing science and statistics, pp. 352–360.

[56] J. Schmalenstroeer and R. Haeb-Umbach. Online speaker change detection by combining bic with microphone array beamforming. 2006. Ninth International Conference on Spoken Language Processing.

[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. 2014. The Journal of Machine Learning Research 15, no. 1. pp. 1929–1958.

[58] S.S. Stevens, J. Volkmann, & E.B. Newman. A scale for the measurement of the psychological magnitude pitch. 1937. The Journal of the Acoustical Society of America, 8 no. 3, pp. 185–190.

[59] S. Tranter and D. Reynolds. An overview of automatic speaker diarization systems. 2006. IEEE Transactions on audio, speech, and language processing 14, no. 5, pp. 1557–1565.

[60] G. Tsoumakas, and I. Katakis. Multi-label classification: An overview. 2007. International Journal of Data Warehousing and Mining (IJDWM) 3, no. 3. pp. 1–13.

[61] O. Vinyals, G. Friedland. Towards semantic analysis of conversations: A system for the live identification of speakers in meetings. 2008. Semantic Computing, IEEE International Conference, pp. 426–431.

[62] P. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. 1974. Ph. D. dissertation, Harvard University.

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

[64] J. Wu, H. Qin, Y. Hua, & L. Fan. Pitch Estimation and Voicing Classification Using Reconstructed Spectrum from MFCC. 2018. IEICE Transactions on Information and Systems 101, no. 2: pp. 556–559.

[65] S.J. Young and S. Young. The HTK hidden Markov model toolkit: Design and philosophy. 1993. University of Cambridge, Department of Engineering.