Ostfalia
University of
Applied Sciences

Wolfenbüttel

Faculty Computer Science

Adrian Reimer, 70409042

Development of a Progressive Web App for Speech Emotion Recognition using Deep Learning

Thesis to obtain the university degree

Bachelor of Science

in Computer Science

at the Ostfalia University of Applied Sciences

First examiner: Prof. Dr.-Ing. Detlef Justen
Second examiner: Dipl.-Ing. Christian Freystein
Submitted on September 23rd 2020

This work was done in cooperation with the Bertrandt Group.

bertrandt

Salzgitter
Suderburg
Wolfsburg

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Ort, Datum	Unterschrift

Introduction and Motivation

Emotions are complex biological states associated with the nervous system and include thoughts, feelings, behavior, pleasure and displeasure. Since the past two decades research in emotion increased exponentially with various scientific fields contributing including neuroscience, psychology, endocrinology, medicine, history and computer science. Numerous theories in attempting to explain the origin, neurobiology, experience and function of emotion have been released in recent years. Constant discussions and falsifications made it impossible to find a scientific consensus although emotions were proven to motivate adaptive behaviors that were essential in the past for passing on genes through survival, reproduction and kin selection [59].

The modern world increasingly focuses on connecting people over the internet. Social media, online games and streaming services target the human desire to socialize and communicate with others. Speech emotion recognition systems specify on classifying speech files using discrete or dimensional model approaches. Through the help of a progressive web app predicting emotions per audio input people would have an easily accessible system for training emotion recognition. Such a system may never reach a high accuracy level and is always deceptive through repression and acting yet advantages would outweigh the disadvantages.

Problematic and Objective

Since the recent improvements in deep neural networks audio classification systems are getting increasingly popular. Speech emotion recognition systems have been around since at least the 1990s, but with the help of pretrained convolutional neural networks they recently achieved vastly higher accuracies. Web applications are growing fast in popularity and will eventually dominate the whole application spectrum. This thesis aims at implementing a progressive web application for speech emotion recognition. As web applications written in JavaScript tend to be much slower than typically optimizable languages like C/C++, it would be scientifically useful to prove if speech emotion recognition systems running in the web browser are achievable.

The objective is to prove if such a system is implementable and how stable the application runs on a low tier smartphone. MFCC spectrograms are used to represent sound as images that RGB channel stacked get fed to a frontend or backend model. Splitting frontend and backend prediction achieves running a low CPU intensive client-side model that compared to the backend model is much smaller though less accurate. Full specification for the User Interface and application were defined in appendix a.

Abstract

This thesis implemented a discrete speech emotion recognition system running on a progressive web application using the Javascript libraries TensorFlow and Meyda. The goal was to provide a mobile accessible system that aims to help people who lack skills in properly estimating emotions.

Two Convolutional Neural Networks were transfer learned for frontend and backend prediction using the Keras framework. A split between frontend and backend was done since more complex models on the backend side may achieve higher accuracies. The frontend model used a MobileNetV2 model and the backend model a Xception model. Both were pretrained on the ImageNet dataset. The datasets CREMA, RAVDESS, SAVEE and TESS were used to predict seven different emotions including anger, disgust, fear, happiness, neutrality, sadness and surprise. Speech files were transformed into picture-like MFCC spectrograms and stacked three times to comply with the models RGB color channels. The progressive web application was designed in Balsamiq and implemented in Vue using PouchDB as the database.

The frontend model reached 57.97% and the backend model 62.11% accuracy. The progressive web app achieved a stable and reactive user experience. Inference values tested on the Cubot R9 were 234.91ms for the frontend tensorflow.js prediction and 97.53ms for the backend flask hosted TensorFlow prediction.

The results showed that the split between frontend and backend achieved different results and that recognizing emotions in speech through a progressive web application is achievable even on low tier devices.

Content

1. State of the art

In the following chapters the current state of the art features needed to build performant Speech Emotion Recognition systems will be introduced.

1.1 Emotional models

In the scientific field of human emotion classification there are two distinct types of models: discrete and dimensional. The vast majority of papers rely on discrete emotions while only a fraction uses the dimensional approach.

Discrete

The discrete emotion model uses psychologically finite and separable emotion categories that are assumed to reflect natural kind categories [1]. Common discrete emotion categories include happiness, sadness, anger, disgust, neutrality, fear, surprise and calmness. “Each discrete emotion elicits changes in cognition (e.g., narrowing of attention on a tiger in the distance), judgment (e.g., the risk perceived in the environment), experience (e.g., the recognition that one is afraid), behavior (e.g., a tendency to run away), and physiology (e.g., increased heart rate and respiration),” ([1], 2011, p.835).

Dimensional

The dimensional model reduces emotions to general affective dimensions of typically valence and arousal [2]. The field of Psychophysiology assumes that these two characteristics are fundamental parts of human emotion [14]. Dimensional models have the advantage that if someday emotions can be accurately quantified, they most likely exceed the performance of subjective human labeled discrete emotion models.

1.2 Preprocessing

The next sub chapters are about professional techniques used in audio and speech classification systems.

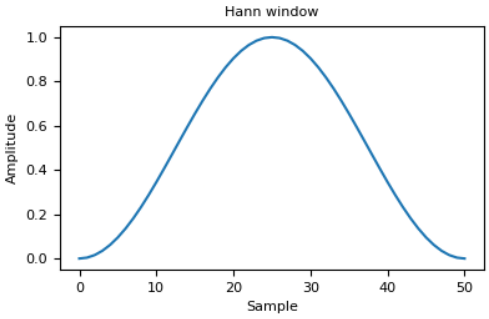
Framing

Audio Signals are typically dynamically changing over time. A static representation of the signal is achieved by periodically extracting short time segments (analysis frames). These analysis frames have a fixed length and are rapidly shifted over the signal with a constant timestep. The step size is generally between 20ms and 60ms, so that the frames are at least overlapping 50% with each other [5].

Windowing

The above extracted analysis frames are then smoothed by a windowing function. Window functions are bell shaped, thus having minima at the boundaries and symmetrically approaching a maximum in the middle of the analysis frame. This avoids abrupt changes at the boundaries of the frame which may cause distortions in the later constructed spectrum [5]. Typical window functions are the Hann and Hamming windows.

$$w(n) = 0.5 - 0.5cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \leq n \leq M-1$$



$$w(n) = 0.54 - 0.46cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \leq n \leq M-1$$

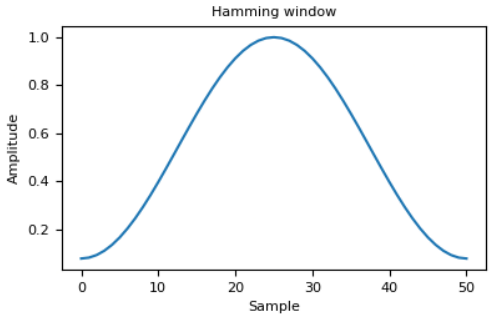


Figure 1.0: Hann and Hamming window

The main difference between the two are the boundary values. While the hann window touches zero, the hamming window only reaches a near zero minimum.

Normalization

In signal processing normalization is used to remove any unwanted channel effects (e.g hall). This is achieved by adjusting the values to a notionally common scale. The most popular normalization method for audio signals is Cepstral Mean Normalization (CMN) [15].

CMN is performed in three simple steps:

1. Calculate the cepstrum
2. Subtract the average from every coefficient
3. Divide by variance

Although CMN is not mandatory for single label datasets, it can still push the accuracy of machine learning models by a few percent.

Spectral Features

Digital sound carries characteristic patterns in both time and frequency content. Therefore, it is common practice to analyze sound in a frequency over time representation. Spectrograms exist in various implementations including Energy Spectral Density (ESD), Power Spectral Density (PSD), etc. [16]. In the following the two currently most used and highly rated spectrograms for SER are discussed. Both the Mel power spectrogram (MPS) and Mel frequency cepstral coefficients (MFCC) build on top of the Mel-scale [16]. It corresponds to an approximation of the psychological sensation of heights of a pure sound (e.g. a pure sinusoid) [5].

MPS are generated by first applying the raw audio with an operation called Fourier transform (FT) [18]. The FT basically transforms the signal domain from time to frequency. Because the audio signal is usually in digital form, the FT can’t directly be taken. A discrete fourier transform (DFT) is needed, which computing optimized is known as Fast fourier Transform (FFT). After applying a FFT the resulting diagram shows the frequency distribution of the entire audio signal. To get a better representation of the signal the Short time fourier transform (STFT) must be calculated, which displays the frequency distribution of the signal over time [17].

Finally, a transform of the frequency spectrogram into Mel-scale needs to be applied followed by logarithm calculations on these values. At the end it results in a MPS which displays frequency over time with values indicating the decibel volume.

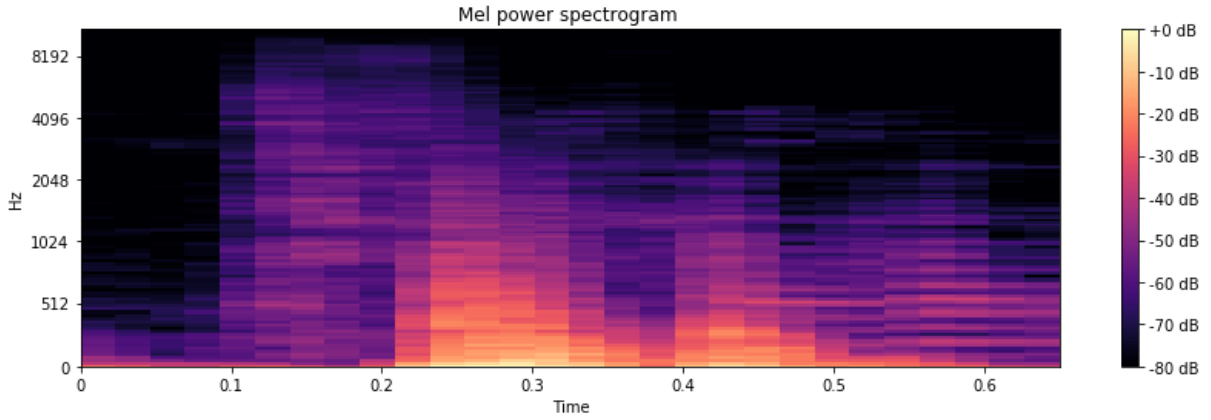


Figure 1.1: Mel power spectrogram

Common spectrograms usually have considerable redundant information between adjacent frequency bins and are often reduced to 32-128 frequency bands using a filter-bank. The Mel scale filter is often used for audio classification because it mimics the human auditory system best, yet other similar filters do exist including Bark scale and Gammatone. Although MPS are not as popular as MFCC, in certain situations where there is a lot of data and computing power available, they can result in a slightly higher accuracy [16].

MFCC's are frequently used spectrograms in speech and audio recognition systems [18]. They are an addition to the already described MPS. The procedure to transform an audio signal into MFCC's is derived as follows:

1. Calculate the STFT of the windowed signal
2. Transform the spectrogram into Mel-scale
3. Take the logarithm of these constructed values
4. Apply a Discrete cosine transform on the MPS
5. The amplitudes of the resulting spectrum form the MFCC

The MFCC in comparison to the MPS uses significantly less data points, which increases the ML models prediction speed and is therefore to be preferred. Especially for mobile devices, who often lack in computing power, this is highly noticeable.

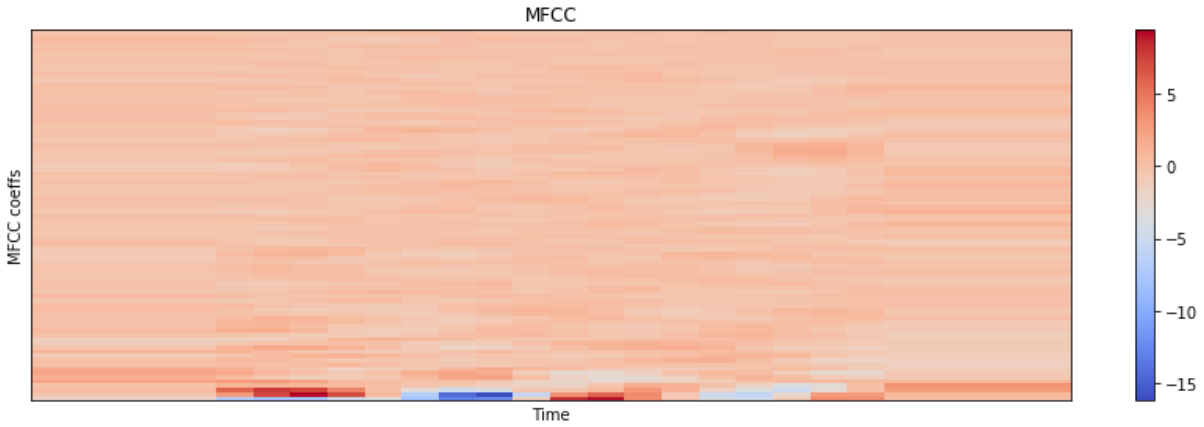


Figure 1.2: Mel frequency cepstral coefficients

Noise Reduction

Noise is still a frequent part of natural environments. Even though most datasets are recorded in a professional laboratory, there are still use cases where noise reduction techniques come into use [19]. Especially in live audio stream capture systems it is a good practice to apply at least a simple noise reduction algorithm. Two well-known noise reduction techniques are Adaptive Wiener and Adaptive LMS[1] filtering [6].

Adaptive Wiener Filtering is one of the most basic noise reduction algorithm used in modern applications. It is computed by providing a related signal containing noise as input and statistically estimating the filtered output signal. The main advantage is that the filter minimizes the mean square error between the estimated and desired signal. A heavy disadvantage is that the noise signal is finite, meaning the sources of interference must be known pre-launch [6].

[1] Least mean squares

Adaptive LMS filtering is an effective but computational complex alternative. The algorithm estimates a filter by calculating the filter coefficients that relate to producing the LMS of the error signal (difference between actual and desired signal). “The adaptive filter is more effective when the carrier frequency of the narrowband interference is offset from the carrier of the spread spectrum signals” ([6], 2014, p.356).

Jitter

The Jitter describes the time distortions of digital audio signals [20-23]. The Analog-Digital converter (ADC) transforms analog voltage values recorded from the microphone to digital values on the computer. Vice versa the Digital-Analog converter (DAC) transforms digital values stored on the computer back to analog voltage values on the connected speaker. During the transform the converters sample data points from the signal with the help of a clock. The sampling points could be acquired in changing time intervals because the clock does not always provide an acceptable frequency. These time deviations between the sampling points of a signal is called Jitter. The lower the Jitter value the better the audio signal.

Shimmer

Shimmer is defined as the variability of peak-to-peak amplitude values of the audio signal [20, 22, 23]. Usually the local Shimmer is performed which includes the calculation of the average absolute difference between the amplitudes of consecutive periods and dividing it by the average amplitude. The lower the Shimmer value the better the audio signal.

HNR

In audio signals the Harmonics to Noise ratio (HNR) defines the proportion of harmonic sound in contrast to additive noise [20, 24]. The lower the HNR value, the more additive noise is in the audio signal. The average global HNR is calculated by simply dividing the power of the harmonic signal with the power of the additive noise signal.

1.3 Datasets

Accurate, large scale Datasets are the foundation of every ML model. The following sub chapters introduce four commonly used discrete datasets for emotion classification. These datasets are all easily openly accessible and overwhelmly overlap in their labels although other harder to get datasets do exist.

CREMA

The CREMA[2] dataset consists of 7442 sound files from 91 different actors [3]. These sound files vary in 12 evenly distributed sentences that cover six different emotions and four intensities levels. The actors consist of 48 males and 43 females with ages between 20 and 74 coming from various cultural ethnicities. Emotions include anger, disgust, fear, happiness, neutrality and sadness.

The dataset was captured in a professional environment. The actors performed standard sentences like "*It's eleven o'clock.*" and were coached by professional theatre directors.

The sound files are rated in three categories: the video alone, the audio alone and the combination of both. Each of the 2443 Crowd Sourced Participants rated 90 unique recordings where 30 belong to each category. "*The human recognition of intended emotion for the audio-only, visual-only, and audio-visual data are 40.9%, 58.2% and 63.6% respectively.*" ([3], 2014, p.1). 95% of the sound files have more or equal than 7 ratings. The dataset is freely available under an Open Data Commons Attribution License.

RAVDESS

The RAVDESS[3] dataset contains 1440 sound files from 24 professional actors [4]. The actors vocalized two sentences with two emotional intensities (normal and strong) in a standard North American accent. Actors are evenly split between male and female. Emotions include calmness, happiness, sadness, anger, fear, surprise, and disgust.

This dataset was recorded in a laboratory at Ryerson University. The performed sentences include "Kids are talking by the door" and "Dogs are sitting by the door".

[2] Crowd Sourced Emotional Multimodal Actors
[3] Ryerson Audio-Visual Database of Emotional Speech and Song

The sound files are each rated 10 times on emotional validity, intensity and genuineness [4]. A total of 319 individuals rated the clips. 72 participants provided test-retest data. “High levels of emotional validity and test-retest inter rater reliability were reported” ([4], 2018, p.1).
The dataset is freely available under a Creative Commons License.

SAVEE

The SAVEE[4] dataset consists of 480 sound files from four students and researchers at the University of Surrey [25-27]. The actors have a strong British accent and are aged between 27 and 31 years. The sound files consist of 15 sentences per emotion where 3 are common, 2 emotion-specific and 10 generic. Emotions include anger, disgust, fear, happiness, neutrality, sadness and surprise.

The dataset was captured in a 3D vision laboratory during different times of the year. Text and emotion fields were displayed on a screen in front of the actors. The emotion screen consisted of a video and three emotion pictures. The actor then needed to perform the sentences displayed in the text field in those three emotions. Unsatisfying utterances were repeated until a satisfying result was achieved.

The recordings are indirectly rated by the actor and researcher. There are no additional people involved in rating this dataset. The dataset is freely available for research purposes.

TESS

The TESS[5] dataset consists of 2800 sound files performed by two female actresses aged 26 and 64 years [28]. Each recording contains a single sentence that starts with the phrase “Say the word” followed by one of 200 filler words. Emotions include anger, disgust, fear, happiness, pleasant surprise, sadness and neutrality.

The dataset was captured in a laboratory at the University of Toronto. This constant sentence was performed for each emotion, focussing on voice features rather than content.

There are no additional people involved in rating this dataset. The dataset is freely available for non commercial use under a Creative Commons License.

[4] Surrey Audio-Visual Expressed Emotion
[5] Toronto Emotional Speech Set

1.4 Transfer learning

The field of Transfer Learning has emerged as a simple and effective technique in deep learning tasks [7]. As needed large scale networks require high amounts of training data, a DNN pretrained on big datasets would be a logical train of thought. Therefore, fairly complex tasks may be solvable using just a tiny dataset. Transfer learning tries to focus on exactly that.

Deep learning is a subclass of machine learning that focuses on training artificial neural networks (ANN). An ANN is the most fundamental part of DL [9]. It consists of multiple connected neurons that mimic the cognitive functions of various biological lifeforms. The neurons contain linear functions that get its inputs from previous neurons and produce an output for following neurons. Optional activation functions can be applied on top of these linear functions. They help to express nonlinearity between the neurons input and output. To get a better visual understanding, multi-layer perceptrons are usually depicted.

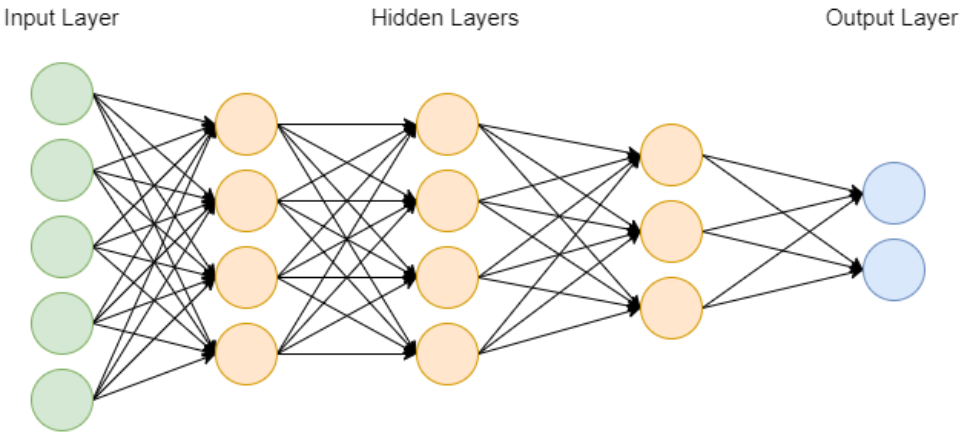


Figure 1.3: Multi layer Perceptron

Transfer learning uses models that are pre trained on a data heavy task. The ImageNet dataset, which contains thousands of images depicting real world objects labeled in 100 categories, is usually used. It forces the model to learn every little feature from a picture, which helps classification of later tasks [29]. These tasks do not necessarily have to be the classification of real-world objects.

Surprisingly, Audio and Speech recognition systems are as well profiting from pre trained ImageNet models. Providing the input is a two-dimensional matrix with three color channels similar in dimensions of the ImageNet pictures.

MobileNetV2

MobileNets are efficient models for mobile and embedded devices [8]. They consist of significantly fewer model parameters while still guaranteeing near top notch accuracy. On Top of that they occupy way less memory than its more accurate counterparts at about 15 MB. In comparison the currently most accurate model NASNetLarge takes about 343 MB memory.

The foundation of MobileNets is formed on depth wise separable convolutions. Convolution is a technique used in CNNs[6] [10]. Depth-wise separable convolutions are more efficient than standard convolutions. They need fewer parameters and floating-point operations that usually can't be ignored for low tier GPUs[7] present in mobile and embedded devices.

While 2D convolution is performed over multiple input channels depth wise separable convolutions keep each channel separately [30]. Besides that, an additional 1x1 convolution is performed across those channels. This can be repeated multiple times for different outputs. The output channels then mix in these different 1x1 convolutions to the resulting channel separate convolutions.

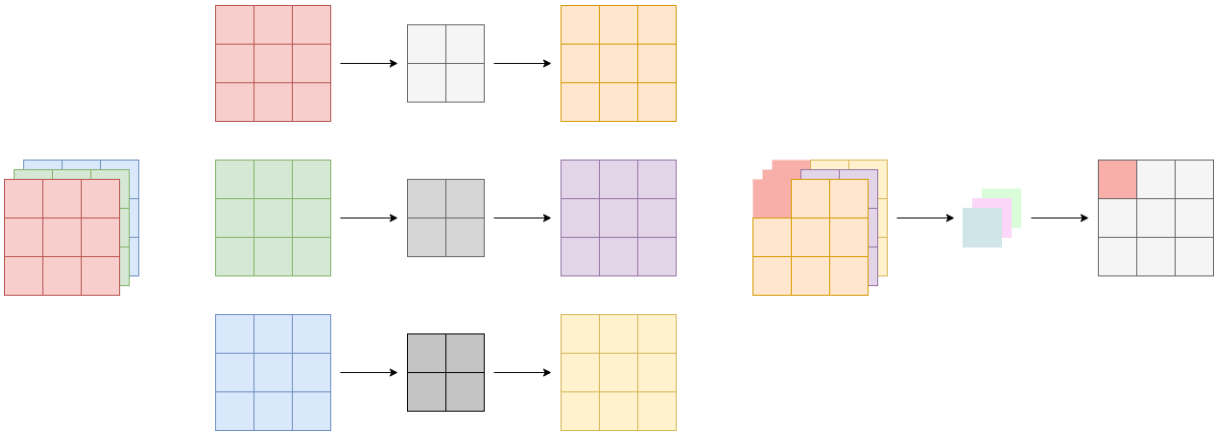


Figure 1.4: depthwise separable convolution

While the first version of the MobileNet introduced the depth wise separable convolution, the second version added an expansion layer in the pipeline to get an expansion-filtering-compression system [11]. This expansion layer simply scales the input tensor to a higher dimensional space that then later gets compressed into an output tensor.

[6] Convolutional neural networks
[7] Graphics processing units

Xception

The Xception network is inspired by the 2014 introduced Inception network. Inception networks use inception blocks that unlike traditional CNNs compute multiple convolutions and pooling in parallel and then concatenate these outputs back into a filter. The Xception network uses the same logic only replacing the standard convolutions with the in Chapter 1.6.1 introduced DS convolutions.

While the amount of model parameters for both the Xception and Inception are equal, the Xception outperforms the Inception model significantly [12]. This is solely achieved by the increased efficiency of the Xception models parameters.

1.5 Progressive web app development

The next sub chapters are about fundamental web application development.

HTML[8] and **CSS**[9] are two fundamental parts of today's websites. Alongside with JavaScript (JS) they form the essential stack for any professional online service.

HTML is a language for configuring the structure or skeleton of a web page. Every website defines an index html file that gets requested as soon as a user visits the site. The language uses markup syntax to define elements that label straightforward pieces of the site's content.

CSS is used to describe the visual layout of a page on various devices. It allows developers to dynamically apply styles like colors and fonts to the website. The separation between markup and style files increases the flexibility because styles are easy to share and outsource. Furthermore, it increases overall maintainability especially needed for large scale projects.

[8] Hyper text markup language
[9] Cascading style sheet

JavaScript

For the construction of any dynamic and interactive website a preferable lightweight interpreter language is used. JS fills in this gap since 1995 and furthermore allows JIT[10] compilation, which simply involves compilation of source- or bytecode during run time to machine code understandable by the current device [32]. Although JS being a web scripting language it shares many syntax similarities with C. Besides some old features JS offers many uniquenesses including prototyping, asynchronous functions, promises, etc... On top of that JS code can be written in all three programming paradigms: imperative, object oriented and declarative. A big downside of JS is its inability to open new executing threads. That means every command is implemented for a single thread only special features like asynchronous functions or promises having the ability to change the current function context for a small timespan.

In the following two fundamental libraries used in web-based audio and speech recognition systems will be introduced.

TensorFlow (TF) is a free and open source library developed by Google mainly and aimed for machine learning tasks [33]. It implements many useful functions for solving neural network problems. The term Tensor in TensorFlow refers to a mathematical multilinear object that describes relationships between objects in a vector space. In 2016 Google released a special hardware chip called TPU[11] that specializes in running machine learning models built with TF. They provide an extremely high data throughput at a reasonably low energy consumption.

TFJS is a recently ported TF library aimed for usage in web apps [34]. It implements all important functions needed to build basic ML models in JS. Already developed models can easily be transferred for usage in TFJS[12]. Export scripts translate the model specific data notation to JSON[13] format with additional shards that can be cached by the web browser. These shards are binary files containing the model's weights and biases.

Meyda is the go-to library for developing efficient audio-based JS apps [35]. It provides a feature rich function set for client-side audio extraction and analysis. Although many audio extraction libraries are developed in a more real time suited programming language, Meyda

[10] Just in time
[11] Tensor processing unit
[12] Tensorflow for Javascript
[13] Javascript object notation

first introduced a high performant alternative for the web. The library itself shares source code with the python equivalent Librosa library, which is often used for building speech and audio ML projects. Meyda is developed and released freely under the MIT License.

ECMAscript

Since 1997 JS is regulated by ECMA[14] International to support code functionality between multiple browser engines. Even though JS and ECMAScript (ES) describe the somewhat same language JS is still a trademark owned by the Oracle Corporation. While ES defines the actual web scripting language JS and similar scripting languages are only “dialects” of that language. Although JS is commonly used for describing the web scripting language. The majority of ECMA code is written for front-end tasks while a slowly increasing fraction is written for back-end services. ES 11 is as of June 2020 the newest standardized web scripting language.

Browser engines

A browser engine is a fundamental part of every web browser [36]. It transforms HTML, CSS and JS files into an interactive visual representation. As soon as a web page is visited, the browser engine creates a DOM[15]. The DOM stores information about objects in a tree structure. JS has complete power over this model and can dynamically modify it at run time.

Currently Chrome’s Blink engine ranks first in market share followed by Safari’s Web kit and Firefox’s Gecko engine [13]. Depending on what feature the user values more one browser engine might be more suitable for a certain task. In general Blink values execution speed first while Gecko prefers security.

Frontend Frameworks

Frontend web development has become an increasingly complex task over the past years. While single page websites get by using plain HTML, CSS and JS, large scale company projects need to rely on bigger management. A Framework provides a standardized toolchain to build and deploy web apps [31]. It usually consists of important libraries, routing functionality, templating and further techniques to manage dynamic sites. The three most common frontend JS frameworks are Angular, React and Vue. Each focusing on suiting to a specific task.

[14] European computer manufacturers Association
[15] Document object model

Angular is well known in big enterprises for its ability to produce rich single page applications [37]. The framework implements dozens of features which makes it the heaviest and hardest to learn of three. Angular uses typescript, a superset of JS, as its official programming language. It is developed by Google and used for Google and Microsoft's web presence.

React - being no framework per se rather a library - is broadly adopted in small businesses and startups. It's component-based architecture forced the other frameworks to adapt it as well. React is extremely lightweight and flexible which forces the development team to include additional needed features [38]. It's even possible to write code for the backend. The framework is developed by Facebook and used for Facebook and Instagram.

Vue is a quite new framework and yet not adopted by any big company [39]. It implements many interesting features like its progressive project design which makes adopting portions of a project extremely simple. Furthermore, Vue has by far the best documentation and is thus preferred by beginners. Currently Nintendo and Gitlab use Vue for their web presence.

Web Assembly

Modern browsers can execute Web Assembly code. Web Assembly compiles into bytecode - code for a virtual machine - that almost reaches native execution speed [40]. Although it is possible to write code directly in Web Assembly, most times the development is outsourced to more popular programming languages and then later compiled into Web Assembly. Only since recently Web Assembly has been officially defined as a web standard. This adoption makes it easy to polish a web app's performance to an almost native like level. Web Assembly is funded by big corporations like Google and Microsoft, which ensures further optimization and additional language support.

Transport Layer Security

Transport Layer Security (TLS) is a hybrid encryption technique usually used in combination with HTTP[16] [41]. It is part of the OSI[17] model's Session Layer and critical for secure communication through the internet [42]. There are two major versions of TLS: TLS 1.2 and 1.3. In the following paragraph the procedure of the new TLS version 1.3 is shown. The TLS 1.3 protocol can be split up into four parts:

1. **Client Initialization**
   After The client generates a private and public key it sends a message containing a merged private and public key alongside the pure public key to the server.
2. **Server Initialization**
   The server now having the private and public key of the client on top of its own generated private key generates the master key using the Diffie-Hellman algorithm. This key will be used to encrypt and decrypt all later messages. Lastly the server sends back its private key merged with the clients public key to the client.
3. **Client request**
   The client then also generates the master ending the security part of the TLS protocol. Now the client can request the content of the site encrypted by the master key.
4. **Server response**
   The server finally responds with the content encrypted by the master key ending the protocol

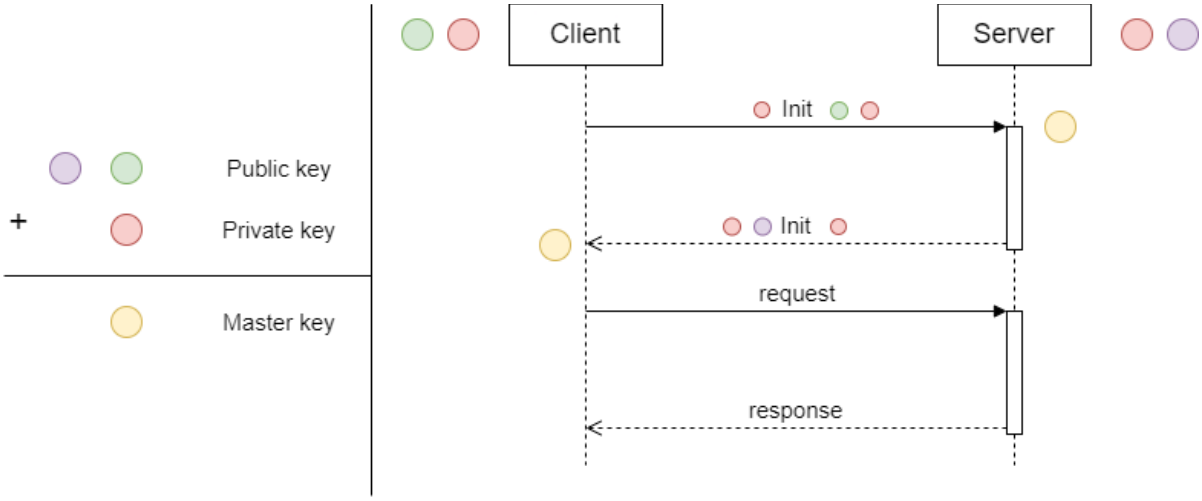


Figure 1.5: Sequence diagram of TLS 3.0

[16] Hyper text transport protocol
[17] Open Systems Interconnection

Databases

A Database (DB) organizes collections of data stored on a computer. The vast majority of DB's are relational, meaning they organize data in tables containing rows and columns [43]. The DBMS[18] then interacts with the DB by using a query language. SQL is adopted by almost every DBMS and thus the go to tool for communication between application and database. Recently however there is a trend to use non-relational databases, which is popular by the term NoSQL expressing the use of different query languages. NoSQL DBMS often use object structure like documents such as XML[19] or JSON and use the containing attributes for querying. This often results in a heavy overall performance gain compared to other standard SQL DBMS [44].

[18] Database management software
[19] Extensible markup language

2. Experimental setup

The following chapters contain experimental setups of both the DNN and web application.

2.1 Deep neural network

The DNN needs to be optimized for running in the frontend and backend. In the next chapters the model selection for each environment will be discussed in combination with furthermore training explanation.

2.1.1 Model selection

As described in chapter 1.4 there are two state of the art models with extreme differences in depth, parameters and thus contained memory space.

The MobileNet model is already well adopted in low performance devices and increasingly gaining in popularity for various frontend tasks. It takes about 15 mb memory space and is therefore easily catchable by the browser while still achieving near state-of-the-art accuracy.

The Xception model's accuracy is slightly higher than the MobileNet, although it takes about 88mb memory space and is performing way worse.

A MobileNetv2 model for the frontend and the Xception model for the backend recognition were selected. Both models are easily accessible with Keras and require only small input shapes compared to other models [45]. The MobileNetv2 model defines a minimal input shape of 32x32 which makes it besides its low memory usage extremely flexible for finding the right spot between accuracy and performance [46]. The Xception model, although its accuracy is slightly worse than some even larger models, only needs a 71x71 input and thus decreases overall network bandwidth [47].

2.1.2 Preprocessing

Recalling chapter 1.3 discussed current state of the art datasets for Speech emotion recognition. After examining each dataset independently for subjective believability jitter, shimmer and HNR calculations were applied as described in chapter 1.2 using the Praat software tool [48]. The goal was to objectively measure the datasets in sound quality.

Dataset	Jitter (%)	Shimmer (%)	HNR (db)
CREMA	4.40	22.62	3.09
RAVDESS	2.76	11.36	9.21
SAVEE	2.56	11.41	11.96
TESS	1.22	6.23	17.28

Figure 2.0: Average global jitter, shimmer and HNR values calculated with Praat [48]

As depicted in figure 2.0 the values are in acceptable ranges and furthermore used in other scientific papers [49, 50]. All four datasets were decided to be used for the SER system.

Now knowing the datasets each sound file needs to be transformed to a certain length. After examining the length of each sound file in the datasets it concluded that almost every file ends after three seconds. At first the silent parts must be cut at the beginning and end, because they contain no significant information and would later get ignored by the model. Secondly all sound files that are now below three seconds need to be expanded. There are two possible solutions for this problem. Either the rest of the audio clip must be filled up with white noise or repeated until the signal is at least three seconds long. Tests concluded that repeating the signal later resulted in a slightly higher accuracy, so this option was selected. Finally, all remaining files that are greater than three seconds by standard were abandoned, knowing those files were not of a single class that could potentially cause later balancing problems.

2.1.3 Data Augmentation

By far Data augmentation techniques are one of the most valuable techniques for training DNNs. It is used to increase the input set by artificially generating new input files. These generation methods use the original input and apply further random distortion algorithms that slightly changes the input. At best, these distortion algorithms reflect natural like effects that would occur in the systems execution environment. Focusing on SER random White Noise, shift, pitch and volume alterations were applied on each speech file.

White Noise is a random audio signal - usually Gaussian - that produces equal intensities at different frequencies, resulting in a constant power spectral density. This option was applied to simulate overall background noise that would especially occur on low quality mobile device microphones. Best results were achieved at a noise level of 20 decibel using an Additive White Gaussian Noise signal.

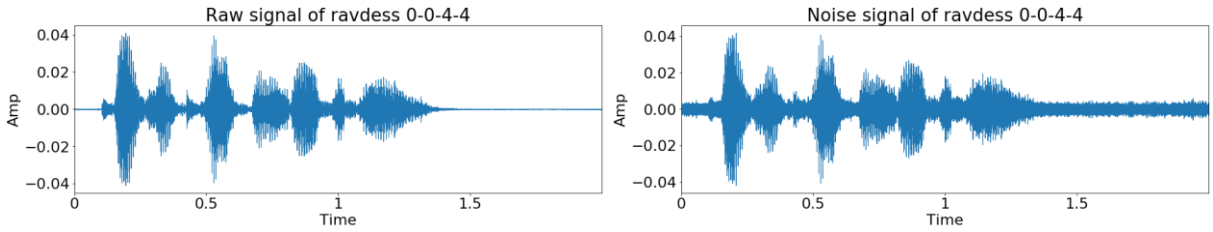


Figure 2.1: Raw and computed noise signal of the ravdess 0-0-4-4 speech file

Shift operations are randomly applied to achieve a more robust that is able to generalize different kinds of speech input files. As described in chapter 1.3 especially the TESS dataset always starts and ends with the same word. This could result in an unwanted bias by the trained model to assume the input always needs to comply with this rule. This problem was solved by randomly shifting the speech files at maximum half the signal in random directions and filling up the rest with white noise.

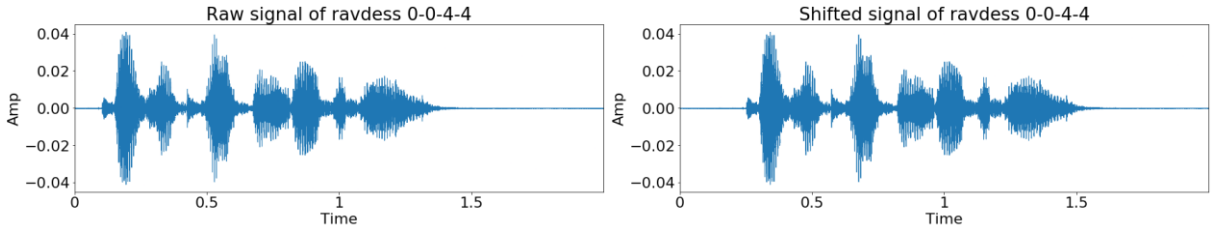


Figure 2.2: Raw and shifted signal of the ravdess 0-0-4-4 speech file

Pitch variations were used to minimize human and especially gender specific voice differences. Although pitch is a significant factor in distinguishing emotions, it is still a frequent altering method used in modern SER systems. Best results were achieved at a pitch altering percentage of around five in increasing and decreasing direction.

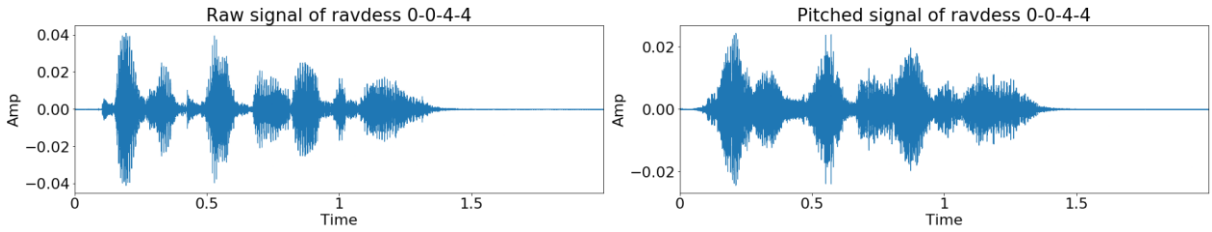


Figure 2.3: Raw and pitched signal of the ravdess 0-0-4-4 speech file

Volume alterations were applied to simulate real world fluctuations in which the user may be close or more away to the mobile device. On top of that mobile devices strongly vary in the user's microphone intensity level. Best results were achieved by varying the volume half the global average decibel value of the signal in increasing and decreasing direction.

Balancing and optimizing these methods is a quite repetitive and time-consuming task. The augmentations were distributed evenly on the speech files so that each had a probability of 25 percent. Augmentation methods were applied in batches while training solely on the training set.

2.1.4 Data Normalization

After having preprocessed and augmented all four datasets, normalization needs to be applied to these sound files. As described in chapter 1.2 the CMN was applied after transforming the speech files into MFCCs. Having executed the CMN for every sound file results in a decently noise robust speech emotion system although furthermore Noise reduction techniques can be applied to reach a commercial level. Further noise reduction methods were abandoned, because neither the frontend nor backend needed to be more complex as it already is.

2.2 Progressive web application

The objective of this thesis was to focus on the development of a simple and easy to use Progressive web app. In the following thoughts and acts on building a professional Progressive web app for Speech emotion recognition are discussed.

2.2.1 User Interface

A good UI is the very foundation of any serious application. As with most things in life it needs to be rightly balanced between functionality and usability. Strong convoluted function dense interfaces may be ideal for proficient scientists that are familiar with electronic devices though inexperienced users would probably drop the app in a matter of seconds. Low convoluted interfaces on the other hand may please the inexperienced users but certainly lack critical controls. The right UI tries to confirm to both sides although slight preferences toward one is common practice. Figure 2.4 shows a use case diagram for the in appendix a defined specification and helps at building the coming UI.

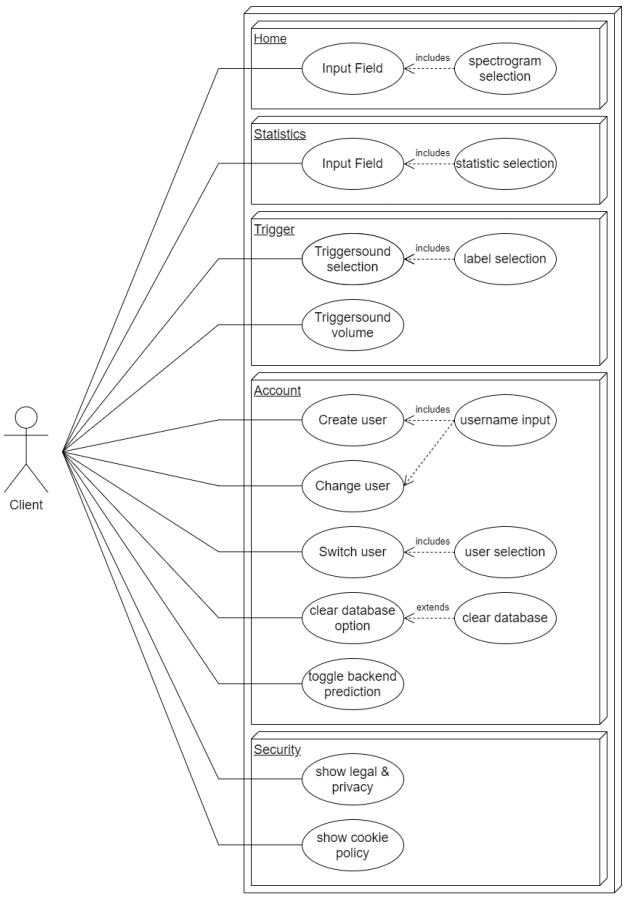


Figure 2.4: Use case diagram of the UI phase

Complying with these practices the prototype can now be built. A UI prototype solely defines the User Interface and not yet the internal algorithms. It was decided to stick to a commonly used application called *Balsamiq* in which creating intuitive Interfaces becomes extremely easy [51]. Additionally, Balsamiq can be used by multiple team members simultaneously using their web equivalent.

Views serve the purpose of separating unrelated content from a website. Before going over all the different main views at least one fundamental logic needs to be discussed. The app needs a way to distinguish different users for the purpose of saving user specific information. A query of the username accomplishes the flexibility of operating multiple close persons or environments (work / leisure) on a single device. For that a view that forces first-time visitors to deposit a valid username needs to be defined. This view simply needs to contain a text field for entering the username and a button for submitting the information. Additional pop ups may be inserted for signaling non-conforming usernames. Always guaranteeing at least one username the visitor should not be able to skip this query.

Now having defined the username query view the five main views separating application critical content can be focused on.

The **home** view contains the actual main logic of the app and internally separates Input and Output. Inside the input area the user should be able to switch between multiple spectrogram depictions and the raw audio stream. Although rendering the input is not important for the function of the app and scarce processing time is needed it plays a significant role in presenting the user the app is running properly. The output area, ideally being a third in size compared to the Input area, serves the purpose of rendering the predicted output computed by the DNN. For a better understanding across different languages it should be preferred to depict an emoticon related to the predicted emotion besides the sole text. On top of that the output area needs to be able to switch between the last rendered prediction and a loading sequence for signaling the user the DNN is executing in the background while the newly windowed spectrogram remains interactive. This home view is by far the most frequent one the user remains on and thus should be especially optimized.

The **statistical** view focuses on presenting useful statistical information about the predicted labels accumulated over time. To comply with a simplistic application four different time spans needs to be included: daily, weekly, monthly and yearly separation. Furthermore, it would be useful to split these time spans into two sections again. One section containing the overall distribution of predictions for the currently selected time span and a second section for showing more time specific information. Additional statistics might be useful for some users however it was decided to lean towards a more simplistic application again making it not too complicated for inexperienced users.

The **trigger** view handles possible sounds that play once a user selected label is being predicted. This feature helps especially people that find it hard to guess a person's emotion solely by speech. Furthermore, they can use this app's feature to train their skills on rightly recognizing emotions although false predictions might occur frequently. Adding checkbox buttons for selecting a trigger sound's label and showing an active or inactive state is recommended. An additional button for previewing the sound must also be included. Besides that, the user should be able to modify the sound's volume preferably by a slider. Warning the user that a defined volume threshold is exceeded to reduce hearing impairments is common practice.

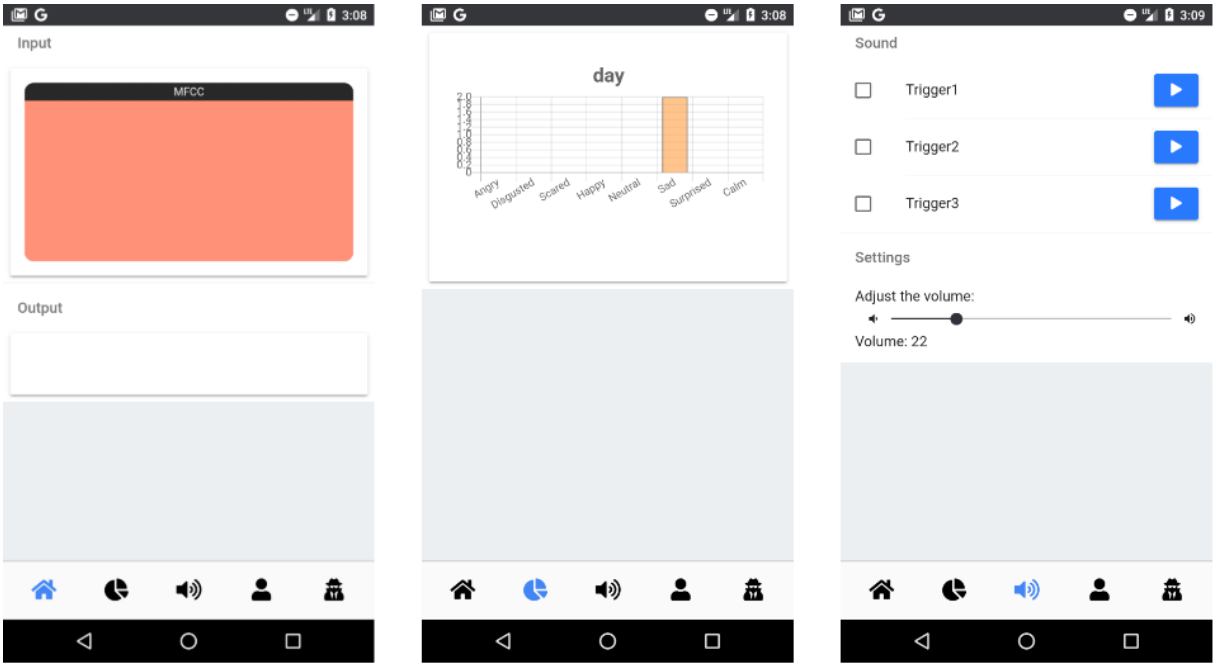


Figure 2.5: Home view (left), statistics view (middle) and trigger view (right)

The **account** view displays the current user's name as well as further account and app specific interactions. In this view the user should be able to create new users as well as having the option to switch between those created users. Moreover, there needs to be an option to change the current username as well as clearing the currently active user's local database. At last there should be a way to toggle the usage of backend prediction if the user does not want to constantly send data to the server. It is recommended to implement all interactions with simple toggle buttons.

The **security** view contains fundamental app security documents including the legal & privacy and cookie policy. These documents must be present in every openly accessible web application and must be reachable within at maximum two clicks from any active view.

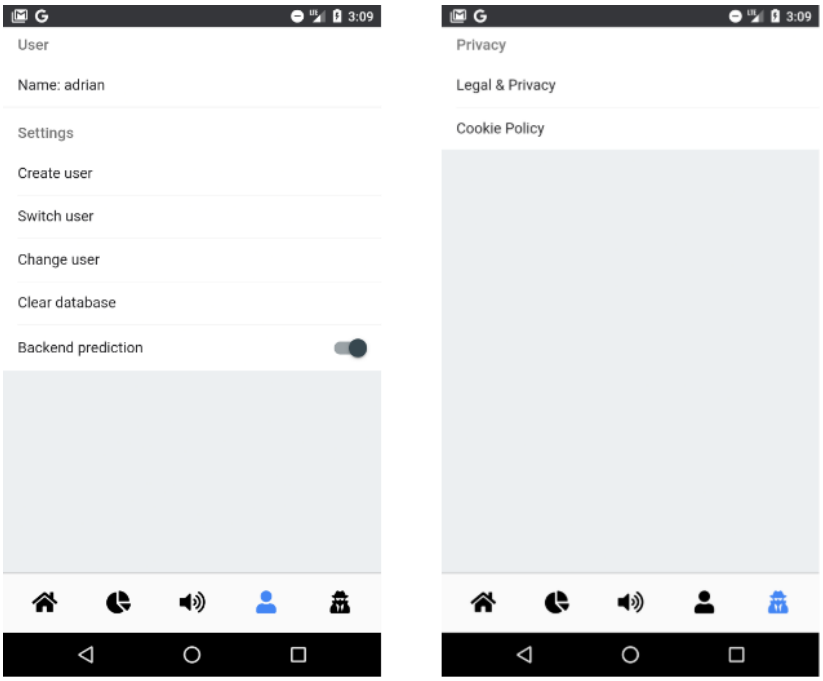


Figure 2.6: Account view (left) and security view (right)

2.2.2 Model conversion

As most of the development for deep neural networks is done in python and at least the JS frontend needing a json style model for operating efficiently the typical python DNN file format needs to be converted to json. Depending on what deep learning development environment is used there are different default file formats. TensorFlow typically saves models as Saved Model format while Keras generally uses the HDF5 format. Surprisingly the TensorFlow team responsible for the JS equivalent developed both a python module as well as a command line tool [52]. It supports all file formats implemented in both TensorFlow and Keras. The frontend model was converted using the command line tool.

```
$ tensorflowjs_converter --input_format=keras /tmp/model.h5 /tmp/tfjs_model
```

Figure 2.7: Tensorflow's model conversion clt [52]

As shown in figure 2.5 there are critical arguments to pass to the tool for guaranteeing correct conversion. Depending on what development environment was used to build the DNN the input format is either Keras or TensorFlow. The second argument defines the input model and the third the model output name. Besides the converted json model defining the structure of the network there are additional shards containing the model's weights. In order for modern browsers to cache these weight shards they have to occupy at maximum 4 MegaBytes in memory. Therefore, both the converted json model and the weight shards need to be accessible on the web server.

3. Experimental execution

In the coming chapters the experimental execution of the DNN and PWA will be explained.

3.1 Deep neural network

Now having set models for both frontend and backend the setup considerations need to be applied and the models need to be trained. As Keras is still the go-to deep learning library that shines with its simplicity it was used for building the models. Keras already provides pre-trained models including the Xception and MobileNetv2 which drastically simplifies code complexity. Keras is used in conjunction with the python programming language.

Before discussing the training phases, the entire preprocessing stage needs to be clarified. As described in chapter 2.1 all speech files were preprocessed from the selected Datasets removing duplicate files that only differ in volume intensity. Having preprocessed the speech files, they now need to be imported in python. Importing and managing files can be done in various ways and out of unimportance will not be explained here although most rely on the module pandas. Now ideally having a Dataframe containing the speech file paths in combination with the relevant label a small percentage of the data needs to be split for later test accuracy measurement. The test set should be 20 percent the size of the original dataset and must not be augmented for ensuring real unseen speech files. After that the augmentation methods as well as their distributions need to be defined.

```
def wnoise(data):
    amp = 0.03 * np.random.uniform() * np.amax(data)
    data = data.astype('float64') + amp *
                        np.random.normal(size=data.shape[0])
    return data

def shift(data):
    range = int(np.random.uniform(low=-1, high = 1) * 1500)
    return np.roll(data, range)
```

Figure 3.0: white noise and shift augmentation functions

```
def pitch(data, sample_rate):
    intensity = 1.05 * np.random.uniform()
    data = librosa.effects.pitch_shift(data.astype('float64'),
                                       sample_rate, n_steps=intensity,
                                       bins_per_octave=12)
    return data

def volume(data):
    vol = np.random.uniform(low=0.5,high=1.5)
    return (data * vol)
```

Figure 3.1: pitch and volume augmentation functions

Figure 3.0 and 3.1. depict the four augmentation methods that were applied on the training set. These augmentations were periodically used on train batches with each having a 25 percent change to trigger.

Continuing a batch generator was implemented to mainly reduce needed ram size that would otherwise sum up to multiple gigabytes of created MFCC spectrograms. The batch generator periodically takes speech files, splits them into augmentable training data and forwards the remaining validation data. After that both sets will be converted into MFCCs and then normalized using CMN. The Librosa module was used for this conversion step. Finally, the MFCCs together with their labels get yielded by the generator allowing simultaneous generation of new batches while the previous one is used for training the model. To be able to train the coming models the *fit_generator* method provided in Keras needs to be used. Furthermore, the validation and train data both have to independently call this method passing the relative dataset in combination with a batch size.

```
def batch_generator(audio_wav_paths, labels, batch_size, istraining):
    lb = LabelEncoder()
    while True:
        batch_mfcc= []
        batch_label = []
        for i in range(batch_size):
            if istraining:
                rnd_idx = random.randint(int(len(train_set))
                X, sample_rate = librosa.load(data.path[rnd_idx],
                                              duration=3,
                                              res_type='kaiser_fast')
                label = data.label[rnd_idx]
                aug_audio_wav = rnd_aug(X)
            else:
                rnd_idx = random.randint(int(len(validation_set))
                X, sample_rate = librosa.load(data.path[rnd_idx],
                                              duration=3,
                                              res_type='kaiser_fast')
            label = data.label[rnd_idx]
            mfcc = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40)
            mean = np.mean(mfcc, axis=0)
            std = np.std(mfcc, axis=0)
            mfcc = (mfcc - mean) / std
            mfccs = np.stack((mfcc,) * 3, axis=-1)
            batch_mfcc.append(mfccs)
            batch_label.append(np_utils.to_categorical(label, label_amt))
        yield(np.asarray(batch_mfcc), np.asarray(batch_label))
```

Figure 3.2: Keras compatible batch generator function

Figure 3.2 shows the implementation of the batch_generator method. Two batch sets one for holding MFCC spectrograms and a second for the relative labels are defined at the top. The for loop iterates over either training or validation data represented by the Boolean *istraining* and randomly loads the speech files. Training data gets additionally augmented while the validation data gets pushed through. Finally, the speech files get converted into MFCCs and then stacked three times to respect needed RGB color channels used by the models. The resulting MFCCs then get appended and yielded for simultaneous training and batching.

The following two chapters discuss the differences between the frontend and backend model.

3.1.1 Frontend model

The MobileNetv2 pretrained on the ImageNet dataset was used for frontend speech recognition. An input shape of 40x130x3 was used resulting in 40 MFC coefficients scattered over 130 time segments and stacked three times.

3.1.2 Backend model

The Xception model again pretrained on the ImageNet dataset was used for backend speech recognition. An input shape of 80x130x3 was used resulting in a concatenation of two 40 MFC coefficients scattered over 130 time segments and stacked three times.

3.1.3 Model training

Flatten followed by a SoftMax activated output layer were added to adjust the class distribution value. The models were compiled using categorical cross entropy as the loss function and Adam as the optimizer. A train and validation batch size of 20, an epoch step size of 50 and an overall epoch amount of 20 was selected.

3.2 Progressive web application

Recalling the UI definition phase of chapter 2.2 the implementation phase is following now. As the UI defines at least five main views containing a decent amount of logic the use of a frontend framework must be considered. Vue is steadily gaining recognition among seasoned frontend web developers for its simplicity and documentation. Therefore, Vue was used to develop a web-based SER system. The final code must be executed from a TLS certified server in order to guarantee secure transfer of speech spectrograms. A cheap and flexible way of running those are Virtual private servers (VPS). In combination with Caddy, an open source HTTPS enabled web server written in Go, it allows a low-cost TLS communication [53]. On top of that Caddy supports the functionality of a static file server, that is needed for sharing the frontend model with the client. Caddy may also be extended by a variety of extra modules including GZIP and ZSTD encoders.

The main logic of the PWA is in the home view where the audio stream of the microphone gets captured, MFCCs are generated and the output label gets predicted. Meyda and TensorFlow are frequently used in this view. All other views are using OnsenUI for making an optimized User Interface for Desktop as well as Android and iOS [54]. PouchDB is used for saving local data quickly and thus reducing the risk of server-side user data thefts and additional law agreements [55]. Starting with the microphone capturing there are already defined and documented functions from official Meyda projects. Only minor changes needed to be applied in order to guarantee compatibility with both models. The number of MFC coefficients need to be adjusted to 40 and the feature set needs to contain all spectrogram options specified within the input field.

```
const analyzer = Meyda.createMeydaAnalyzer({
    audioContext: audioCtx,
    source: src,
    bufferSize: 512,
    featureExtractors: features,
    callback,
    numberOfMFCCCoefficients: 40,
});
```

Figure 3.3: Meyda's audio analyzer

Figure 3.3 depicts a fraction of Meyda's predefined *onMicDataCall* function that needs to be adjusted to match system criteria.

Before the frontend model is ready for prediction it must be loaded by TensorFlow's loadLayersModel function. TensorFlow then returns a promise that will be periodically scheduled to load the model while execution continues. Referring to chapter 1.5 JS allows only one execution thread.

```
loadModel() {
    tf.loadLayersModel('path').then((loadedModel) => {
        this.model = loadedModel;
    });
}
```

Figure 3.4: Tensorflow's frontend model import function

The final prediction logic is called when the MFCC reaches the model's predefined input shape. As execution must be continued to ensure correct performance of the audio stream and overall application stability, the prediction function must be made asynchronous.

```
async predict(input) {
    const stacked = tf.stack([input, input, input]);
    const reshaped = stacked.reshape([1, 40, 261, 3]);
    const model_pred = this.model.predict(reshaped);
    const pred_label = this.labelDict[tf.argMax(model_pred,
                                    tf.axis = 1).dataSync()];
}
```

Figure 3.5: Asynchronous frontend model prediction function

Figure 3.5 shows the asynchronous prediction of incoming speech MFCC spectrograms. The MFCC gets stacked three times to match the RGB channel specification of the model. Additionally, the stacked MFCCs get reshaped to the defined input shape. Now the model tries to predict the output class. As the output is an array of possibilities adding up to one the final step is to calculate the index holding the maximum value and converting it to the relative label.

The backend used a simple flask server running python thus making it possible to use TensorFlow and Keras functions to load the model and predict outputs. Another reason for choosing a python backend was to minimize server-side code complexity as much as possible. Heroku was chosen as the hosting service since it offered a simple and free way to serve the backend model over TLS. The client side created MFCC was sent using asynchronous JavaScript and XML (AJAX) posts to the server. The server then stacked the MFCC to reach the desired shape of 80x130x3 and sent the predicted label back to the client. A full overview of the Backend code is defined in appendix C.

```
@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json()
    mfcc = np.array(data)
    mfcc = np.stack((mfcc,) * 2)
    pred = labelDict[np.argmax(model.predict(mfcc))]
    return jsonify(pred)
```

Figure 3.6: Backend model prediction function

Figure 3.6 shows the implementation of the backend flask prediction. Incoming MFCC spectrograms stored as json are accepted and after transformed into NumPy arrays. Afterwards these arrays get stacked and fed into the Xception model. The predicted label then finally gets transformed into json and sent back to the client.

4. Evaluation

In the last subject of this thesis the trained Xception and MobileNet models are examined. Additionally, fundamental web problematics including latency and security as well as the performance of the converted front and backend models are discussed.

4.1 Deep neural network

There are two main values guiding the training phase of a DNN model naming accuracy and loss. The accuracy value guides the training and testing phase of a neural network and exists in three major forms. Accuracy is calculated by taking the weighted arithmetic mean of precision and inverse precision as well as the weighted average mean of recall and inverse recall. A model is underfitted if the training and testing error is high and overfitted if the training error is low and the testing error is high. Both the underfitting and overfitting problem need to be taken into focus to verify that the model correctly and extensively learned the features and is later able to distinguish inputs best. If the model is trained usually a final test accuracy is calculated to measure the model's accuracy on yet unseen data. In combination with accuracies a loss value is printed. It determines the summation of errors made for each fed input. Besides accuracy and loss values a final confusion matrix is constructed. Confusion matrices plot a model's true labels in contrast to its predicted labels, so that a visualization of the model's performance arises. As described in chapter 2.1 and 3.1 the models were trained in python 3.7 using the CREMA, RAVDESS, SAVEE and TESS datasets in combination with Keras 2.4.

4.1.1 Frontend model

The MobileNetV2 model achieved a final test accuracy value of 57.97% at a loss of 1.275. Without augmentation methods the accuracy reached a score of 54.36%. As shown in figure 4.0 the model struggled at separating neutrality from sadness and disgust from fear and sadness. Happiness was surprisingly often confused as fear. The model classified fear and anger labeled speech files more than half of the time correctly. Anger seems to have a weird similarity with happiness which is not just related to volume since surprise was not affected. Although fear and sadness were understandably oft mixed up, since both are considered negative emotions. In 85% of the cases surprise was correctly classified.

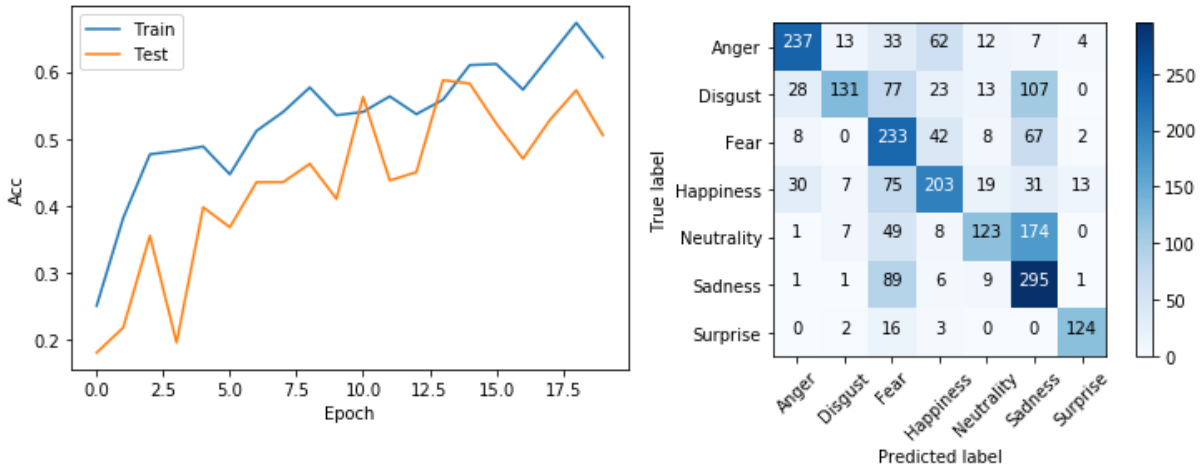


Figure 4.0: Train histogram (left) and confusion matrix (right) of the frontend MobileNetV2 model

4.1.2 Backend model

The Xception model achieved a final test accuracy value of 62.11% at a loss of 1.163. Without augmentation methods the accuracy reached 59.43%. As shown in the figure 4.1 the model was extremely unsure in classifying happiness. It was surprisingly often misclassified as anger, disgust and fear. Neutrality was again mistreated as sadness while fear got mixed up as sadness. Disgust and sadness achieved a relatively decent accuracy score. The model classified anger and sadness more than half of the time correctly. Again the most accurate was surprise which was 81% of the time correctly classified.

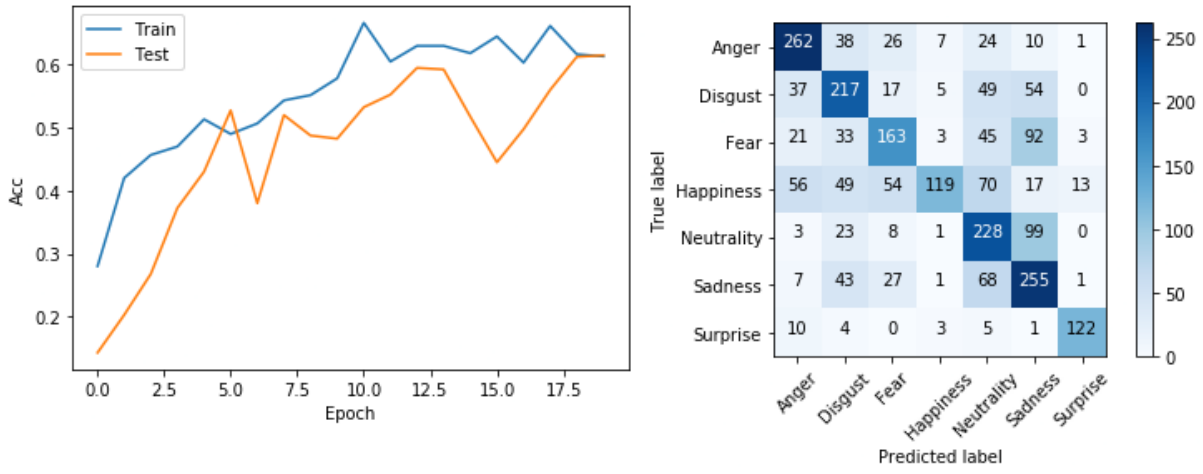


Figure 4.1: Train histogram (left) and confusion matrix (right) of the backend Xception model

4.2 Progressive web application

Focusing on the PWA there are several important factors regarding performance and security. Furthermore, high latencies could result in a poor user experience. In the following both the frontend and backend applications are considered.

4.2.1 Latency

In server communication the latency value describes the time delay between client request and server response. Depending on how accessible the web application needs to be, further locations can be added to minimize overall communication routes. The frontend code is served by a virtual private server (VPS) running a TLS activated caddy server and is hosted in the United Kingdoms. The backend server is served by Heroku running a TLS activated flask server and is hosted in Ireland. The latency was measured in Germany using the windows ping command in default configuration [56].

Application	Min (ms)	Max (ms)	Avg (ms)
Frontend	30	32	30
Backend	127	139	133

Figure 4.2: Minimum, maximum and average latency for the front- and backend server

Figure 4.2 shows that the frontend application is in acceptable ranges for providing a reactive server communication [57]. Depending on how optimized the code and framework are and how powerful the executing device is the UI will highly differ in reaction time. The free hosted backend application has a significant average latency, though a set 1.5 second analysis window (50% overlap) would still be recognized in a decent time frame.

4.2.2 Model performance

Before comparing the front- and backend models, it must be ensured that both models work as intended. This means that giving the same inputs to the fresh trained and hosted model should result in the exact same probability distribution. The test resulted that front- and backend implementations respectively predicted the exact same output and thus ensuring correct model conversion.

Now that the models work as intended the performance and real-world accuracy needs to be measured. Inference is usually used to quantify the performance of a DNN. It refers to the average time it takes to make a prediction using the already trained model. Depending on how powerful the executing device and Interpreter is the inference most noticeably in the frontend will respectively change. Measuring the inference in JS can be tricky because background activities like garbage collection and other promises or asynchronous functions may falsify the result. To reduce this as best and simple as possible the measurement consists of predicting 1000 inputs and taking the average of the prediction time.

Application	Inference (ms)
Frontend	234.91
Backend	97.53

Figure 4.3: Average Inference for the frontend and backend model tested on Cubot R9 (2017)

Figure 4.3 shows the average inference time on the Cubot R9 for both frontend and backend prediction. The backend model is significantly faster than the frontend model even though it is way more complex. Adding up the high latency for the backend server there is no noticeable speed advantage between the two implementations. The Cubot R9 is a low tier smartphone that was introduced in 2017 and is available for around 70€ (2020). It uses a quad core ARM Cortex-A7 CPU at 1300MHz, a dual core ARM Mali-400 GPU and has 2GB RAM at 533 MHz

After having ensured both models work as intended and measured the average inference a final test needs to prove how accurate the models predict real world emotions. Especially the noise reduction solely based on CMN and the Meyda audio analyze function needs to be taken focus on. Furthermore, it must be ensured that the front and backend predictions are calculated and rendered in a reasonable time. For this reason, the models are tested at a noisy environment stored in a pocket and a noise free environment lying on a table. The phone was put with its backside on the table and upside down in the pocket, so the microphone was targeting the playing device. In both environments the device was approximately 1 meter away from the audio source and the intensity of the microphone was set to the maximum. 21 selected and evenly split speech files from the test data set were played on a speaker at mouth level to guarantee the same input.

Application	Location	properly classified	falsely classified
Frontend	pocket	3	18
Backend	pocket	4	17
Frontend	table	10	11
Backend	table	12	9

Figure 4.4: Accuracy in different environments for the frontend and backend model tested on the Cubot R9

Figure 4.4 shows that the frontend and backend predictions for the pocket location were at statistically the same level as simple guessing. Comparing it to the prediction for the table location it seems that the phone could either not measure the volume properly or struggled with too much noise. The table location achieved an acceptable accuracy level although both were lower than the original test accuracy scores. Further increasing the test set would give a more accurate score though it would be beyond the scope of this thesis.

4.2.3 Security

The Security of a web application is rapidly growing in importance over the years. Users and government laws force developers to use state of the art techniques to minimize data theft. As described in chapter 1.5 the two servers that host the frontend and backend application both use TLS to ensure encrypted communication over the internet. The advanced encryption standard (AES) 256-bit encryption commonly used in TLS would result in 2^{256} possible combinations which are impossible to crack at today's computing power.

Focusing on the application security design the data is stored locally on the device making it impossible for hackers to steal user data from the server. As described in chapter 2.2 the account view can create new users and takes in no password, because the JS code that gets requested by each client to run the application would tell exactly how the password was encrypted. This would make it easy for skilled programmers to decrypt it. The only solution would be to make the encryption phase server side, so only people having access to that server know how the encryption works. The implemented application though saves no direct personal information so developing it with focus on client-side saving seems obvious. The only problem that arouses is that every person having access to the phone can see every user's emotions over time which makes the app partially secure. On top of that the database can be deleted by everyone though the app was intended to be used for other environments or trusted close friends/family members. Commercial products would need to find a secure way to save user predicted emotions and delete user content.

The highest security threat would be in the backend prediction function. Even though the constructed MFCC on the running device is securely sent using TLS to the server it gives the backend the opportunity to convert the MFCC back to .wav and thus revealing a slightly distorted representation of the user's voice. There is no direct way to ensure the user the app is not saving anything personal yet European laws strictly forbid saving personal data without consent of the user.

5. Summary and Outlook

The goal of this thesis was to develop a user-friendly progressive web application for speech emotion recognition. To ensure security the app needed to save all relevant data on the running device. In the first chapter current state of the art concepts concerning Speech emotion recognition systems and progressive web apps were discussed. The second chapter handled experimental setup of both the deep neural network and the progressive web app. In this the decision to use the datasets CREMAD, RAVDESS, SAVEE and TESS was taken. Furthermore, a split between front- and backend prediction was defined making it possible to use complex deep neural networks. The frontend version used the MobileNetV2 model and the backend version the Xception model, because they were relatively small in memory size and achieved good accuracy. Both were pretrained on the ImageNet dataset. The UI design discussed important features that must be present in the later application. In chapter 3 the deep neural networks were trained and the UI got implemented in Vue. This chapter made it clear that developing web applications and training deep neural networks are more complex than most people think. The final chapter 4 handled the evaluation of the MobileNetV2 model, Xception model and progressive web app. The frontend MobileNetV2 model achieved an accuracy value of 57.97% and the backend model 62.11%. Considering other papers that focused on speech emotion recognition systems these accuracies were expected. A measurement of the inference on the Cubot R9 showed that the backend model was still more than twice as fast as the frontend model.

This thesis could be extended on implementing face emotion recognition thus having sound and image data for classifying one's emotion. Both would increase the robustness of the emotion recognition system, because at noisy or unclear view the system may still be able to accurately classify the emotion. Besides that, the deep neural networks could be trained on datasets that analyse the spoken words resulting in a possible higher accuracy. Another optimization would be to use the newly hyped transformer that got recently popular in natural language processing. Like CNN’s were surprisingly accurate on audio spectrograms, transformers may work too especially referencing the attention algorithm [58]. On the progressive web app side, the code could be further optimized and tested on different web browsers. Web Assembly could also be used if the support among mobile devices increases. This would boost execution speed to near a native level.

References

[1] Lench HC, Flores SA, Bench SW. Discrete emotions predict changes in cognition, judgment, experience, behavior, and physiology: A meta-analysis of experimental emotion elicitations. Psychological Bulletin. 2011

[2] Barrett LF, Bliss-Moreau E. Affect as a Psychological Primitive. *Adv Exp Soc Psychol*. 2009;41:167-218. doi:10.1016/S0065-2601(08)00404-8

[3] Cao, Houwei & Cooper, David & Keutmann, Michael & Gur, Ruben & Nenkova, Ani & Verma, Ragini. (2014). CREMA-D: Crowd-sourced emotional multimodal actors dataset. IEEE transactions on affective computing. 5. 377-390. 10.1109/TAFFC.2014.2336244.

[4] Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391.

[5] Virtanen, Tuomas & Plumbley, Mark & Ellis, Dan. (2017). Computational Analysis of Sound Scenes and Events. 10.1007/978-3-319-63450-0.

[6] Magre, Smita & Janse, Pooja & Deshmukh, Ratnadeep. (2014). A Review on Feature Extraction and Noise Reduction Technique.

[7] Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683. 2019

[8] Andrew G. Howard and Menglong Zhu and Bo Chen and Dmitry Kalenichenko and Weijun Wang and Tobias Weyand and Marco Andreetto and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861. 2017

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning (Adaptive Computation and Machine Learning series. MIT Press. 978-0262035613. 2016

[10] Umberto Michelucci. Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection. Apress. 978-1484249758. 2019

[11] Mark Sandler and Andrew Howard and Menglong Zhu and Andrey Zhmoginov and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381. 2018

[12] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357. 2016

[13] https://gs.statcounter.com (last visited: 14th September 2020)

[14] Peter J. Lang, Margaret M. Bradley. Emotion and the motivational brain. Biological Psychology. Volume 84. Issue 3. 2010

[15] N. V. Prasad and S. Umesh, "Improved cepstral mean and variance normalization using Bayesian framework," *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Olomouc, 2013, pp. 156-161, doi: 10.1109/ASRU.2013.6707722.

[16] S. Tantisatirapong, C. Prasoproek and M. Phothisonothai, "Comparison of Feature Extraction for Accent Dependent Thai Speech Recognition System," *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, Hue, 2018, pp. 322-325, doi: 10.1109/CCE.2018.8465705.

[17] Z. Su, M. Hong, Fuh-Gwo Yuan. Nonlinear ultrasonics for health monitoring of aerospace structures using active sparse sensor networks. Woodhead Publishing. Pages 353-392. 2016

[18] Md. Sahidullah, Goutam Saha. Design, analysis and experimental evaluation of block based transformation in MFCCcomputation for speaker recognition. Speech Communication. Volume 54. Issue 4. Pages 543-565. 2012

[19] H. Bai, F. Ge and Y. Yan, "DNN-based speech enhancement using soft audible noise masking for wind noise reduction," in *China Communications*, vol. 15, no. 9, pp. 235-243, Sept. 2018, doi: 10.1109/CC.2018.8456465.

[20] A. Ferreira and V. Fernandes, "Consistency of the F0, Jitter, Shimmer and HNR voice parameters in GSM and VOIP communication," *2017 22nd International Conference on Digital Signal Processing (DSP)*, London, 2017, pp. 1-5, doi: 10.1109/ICDSP.2017.8096128.

[21] IEEE Draft Standard for Jitter and Phase Noise," in *IEEE P2414/D02.3, July 2020* , vol., no., pp.1-43, 13 July 2020.

[22] Y. Bennane, A. Kacha, J. Schoentgen and F. Grenez, "Synthesis of pathological voices and experiments on the effect of jitter and shimmer in voice quality perception," *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, Boumerdes, 2017, pp. 1-6, doi: 10.1109/ICEE-B.2017.8192026.

[23] S. S. Upadhya, A. N. Cheeran and J. H. Nirmal, "Statistical comparison of Jitter and Shimmer voice features for healthy and Parkinson affected persons," *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, 2017, pp. 1-6, doi: 10.1109/ICECCT.2017.8117853.

[24] Carole T Ferrand. Harmonics-to-Noise Ratio: An Index of Vocal Aging. Journal of Voice. Volume 16. Issue 4. Pages 480-487. 2002

[25] S. Haq and P.J.B. Jackson, "Multimodal Emotion Recognition", In W. Wang (ed), Machine Audition: Principles, Algorithms and Systems, IGI Global Press, ISBN 978-1615209194, chapter 17, pp. 398-423, 2010

[26] S. Haq and P.J.B. Jackson. "Speaker-Dependent Audio-Visual Emotion Recognition", In Proc. Int'l Conf. on Auditory-Visual Speech Processing, pages 53-58, 2009

[27] S. Haq, P.J.B. Jackson, and J.D. Edge. Audio-Visual Feature Selection and Reduction for Emotion Classification. In Proc. Int'l Conf. on Auditory-Visual Speech Processing, pages 185-190, 2008

[28] Pichora-Fuller, M. Kathleen; Dupuis, Kate, 2020, "Toronto emotional speech set (TESS)", https://doi.org/10.5683/SP2/E8H2MF, Scholars Portal Dataverse, V1

[29] M. Ebrahim, M. Al-Ayyoub and M. A. Alsmirat, "Will Transfer Learning Enhance ImageNet Classification Accuracy Using ImageNet-Pretrained Models?," *2019 10th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, 2019, pp. 211-216, doi: 10.1109/IACS.2019.8809114.

[30] V. Hoang, V. Hoang and K. Jo, "Realtime Multi-Person Pose Estimation with RCNN and Depthwise Separable Convolution," *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, Ho Chi Minh, Vietnam, 2020, pp. 1-5, doi: 10.1109/RIVF48685.2020.9140731.

[31] S. Delcev and D. Draskovic, "Modern JavaScript frameworks: A Survey Study," *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*, Novi Sad, 2018, pp. 106-109, doi: 10.1109/ZINC.2018.8448444.

[32] R. Zhuykov, V. Vardanyan, D. Melnik, R. Buchatskiy and E. Sharygin, "Augmenting JavaScript JIT with ahead-of-time compilation," *2015 Computer Science and Information Technologies (CSIT)*, Yerevan, 2015, pp. 116-120, doi: 10.1109/CSITechnol.2015.7358262.

[33] Martín Abadi and Ashish Agarwal and Paul Barham and Eugene Brevdo and Zhifeng Chen … . TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467. 2016

[34] J. Á. Morell, A. Camero and E. Alba, "JSDoop and TensorFlow.js: Volunteer Distributed Web Browser-Based Neural Network Training," in *IEEE Access*, vol. 7, pp. 158671-158684, 2019, doi: 10.1109/ACCESS.2019.2950287.

[35] Hugh Rawlinson, Nevo Segal, Jakub Fiala. Meyda: an audio feature extraction library for the Web Audio API. Goldsmiths, University of London. 2015

[36] J. Radhakrishnan, "Hardware dependency and performance of JavaScript engines used in popular browsers," *2015 International Conference on Control Communication & Computing India (ICCC)*, Trivandrum, 2015, pp. 681-684, doi: 10.1109/ICCC.2015.7432981.

[37] M. Ramos, M. T. Valente and R. Terra, "AngularJS Performance: A Survey Study," in *IEEE Software*, vol. 35, no. 2, pp. 72-79, March/April 2018, doi: 10.1109/MS.2017.265100610.

[38] A. Javeed, "Performance Optimization Techniques for ReactJS," *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, India, 2019, pp. 1-5, doi: 10.1109/ICECCT.2019.8869134.

[39] N. Adulyanukosol, "Earthquake Damage Report Interactive Dashboard Using Bayesian Structural Time Series and Value-Suppressing Uncertainty Palettes," *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Vancouver, BC, Canada, 2019, pp. 106-107, doi: 10.1109/VAST47406.2019.8986916.

[40] J. Protzenko, B. Beurdouche, D. Merigoux and K. Bhargavan, "Formally Verified Cryptographic Web Applications in WebAssembly," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 1256-1274, doi: 10.1109/SP.2019.00064.

[41] P. Sirohi, A. Agarwal and S. Tyagi, "A comprehensive study on security attacks on SSL/TLS protocol," *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Dehradun, 2016, pp. 893-898, doi: 10.1109/NGCT.2016.7877537.

[42] Y. Li, D. Li, W. Cui and R. Zhang, "Research based on OSI model," *2011 IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, 2011, pp. 554-557, doi: 10.1109/ICCSN.2011.6014631.

[43] Ying Bai; Satish Bhalla, "Introduction to Databases," in *SQL Server Database Programming with Visual Basic.NET: Concepts, Designs and Implementations* , IEEE, 2020, pp.9-65, doi: 10.1002/9781119608493.ch2.

[44] J. de Oliveira Assis, V. C. O. Souza, M. M. V. Paula and J. B. S. Cunha, "Performance evaluation of NoSQL data store for digital media," *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, Lisbon, 2017, pp. 1-6, doi: 10.23919/CISTI.2017.7975844.

[45] C. Tseng and S. Lee, "Design of Digital Differentiator Using Supervised Learning on Keras Framework," *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, Osaka, Japan, 2019, pp. 162-163, doi: 10.1109/GCCE46687.2019.9014634.

[46] https://keras.io/api/applications/mobilenet/#mobilenetv2-function (last visited: 14th September 2020)

[47] https://keras.io/api/applications/xception (last visited: 14th September 2020)

[48] Paul Boersma & David Weenink (2020): Praat: doing phonetics by computer [Computer program]. Version 6.1.16, retrieved 3rd September 2020 from http://www.praat.org/

[49] E. Ghaleb, M. Popa and S. Asteriadis, "Multimodal and Temporal Perception of Audio-visual Cues for Emotion Recognition," *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, Cambridge, United Kingdom, 2019, pp. 552-558, doi: 10.1109/ACII.2019.8925444.

[50] A. R. Choudhury, A. Ghosh, R. Pandey and S. Barman, "Emotion Recognition from Speech Signals using Excitation Source and Spectral Features," *2018 IEEE Applied Signal Processing Conference (ASPCON)*, Kolkata, India, 2018, pp. 257-261, doi: 10.1109/ASPCON.2018.8748626.

[51] A. Delgado and J. Sosa, "Mobile application design of geolocation to collect solid waste: A case study in Lima, Peru," 2019 IEEE XXVI International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 2019, pp. 1-4, doi: 10.1109/INTERCON.2019.8853594.

[52] https://www.tensorflow.org/js/guide/conversion (last visited: 14th September 2020)

[53] M. Furtak and M. P. Wittie, "The ODDness of Webpages," *2019 Network Traffic Measurement and Analysis Conference (TMA)*, Paris, France, 2019, pp. 33-40, doi: 10.23919/TMA.2019.8784552.

[54] https://onsen.io/v2/guide/ (last visited: 14th September 2020)

[55] E. C. Iturralde, M. A. B. Jardeleza, G. Zamora-Racaza, E. G. Penserga and J. D. L. Caro, "Design and development of a registry system for primary vasculitis," *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Chalkidiki, 2016, pp. 1-5, doi: 10.1109/IISA.2016.7785429.

[56] http://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ping (last visited: 14th September 2020)

[57] Jonathan Deber, Ricardo Jota, Clifton Forlines, Daniel Wigdor. “How Much Faster is Fast Enough? User Perception of Latency & Latency Improvements in Direct and Indirect Touch”. 2015. Dept. of Computer Science, University of Toronto Toronto, ON, Canada

[58] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, December 6). Attention Is All You Need.

[59] Cabral, Joao, de Almeida, Rosa. "From social status to emotions: Asymmetric contests predict emotional responses to victory and defeat". 2020

List of Abbreviations

CMN	Cepstral Mean Normalization
ESD	Energy Spectral Density
PSD	Power Spectral Density
MPS	Mel Power Spectrogram
MFCC	Mel Frequency Cepstral Coefficients
MFC	Mel Frequency Cepstrum
FT	Fourier Transform
DFT	Discrete Fourier Transform
STFT	Short Time Fourier Transform
LMS	Least Mean Square
ADC / DAC	Analog-Digital / Digital-Analog Converter
HNR	Harmonics to Noise Ratio
CREMA	Crowd Sourced Emotional Multimodal Actors
RAVDESS	Ryerson Audio-Visual Database of Emotional Speech and Song
SAVEE	Surrey Audio-Visual Expressed Emotion
TESS	Toronto Emotional Speech Set
DNN	Deep Neural Network
ANN	Artificial Neural Network
CNN	Convolutional Neural Networks
GPU	Graphics Processing Unit
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
JS	Javascript
JIT	Just in Time
TPU	Tensor Processing Unit
TF	Tensorflow
JSON	Javascript Object Notation
ECMA	European Computer Manufacturers Association
DOM	Document Object Model
UI	User Interface
TLS	Transport Layer Security
VPS	Virtual Private Server
CPU	Central Processing Unit
RAM	Random Access Memory

List of Figures

Appendix

A. Specifications

1. User interface

1.1. The home view

1.1.1. Must be separated by an Input and Output section

1.1.2. Must signalize the app is retrieving audio signals from the device

1.1.3. Must ask the user for access to the microphone

1.1.4. Must additionally have a button for starting the prediction

1.1.5. The input section

1.1.5.1. Must contain at least the MFCC and raw audio input selections

1.1.5.2. Must contain image and text describe the selections

1.1.6. The output section

1.1.6.1. Must signalize the app is predicting in the background

1.1.6.2. Must show a label and image representation of the predicted emotion

1.2. The statistics view

1.2.1. Must contain selections for daily, weekly, monthly and yearly emotions

1.2.2. Must be large enough for people to click on

1.3. The trigger view

1.3.1. Must contain three different trigger sound buttons that may all be active at once containing a selection of the emotion labels

1.3.2. Must directly show which label the trigger sound activates on

1.3.3. Must show and implement a preview of the trigger sound

1.3.4. Must at least have one master volume slider for regulating the volume of all trigger sounds

1.3.5. Must warn the user by text if the volume exceeds the normal threshold

1.4. The account view

1.4.1. Must show the name of the current user

1.4.2. Must contain functions for creating, switching and changing the current user

1.4.3. Must contain a function for deleting the current user’s database

1.4.4. Must contain a button for showing the state of the current prediction option

1.5. The security view

1.5.1. Must contain functions for showing legal & privacy and cookie policies that are accessible within at maximum two clicks from any stand view

2. The progressive web application

2.1. Must run on the Brave browser

2.2. Must run smoothly on low tier smartphones comparing to the Cubot R9

2.3. Must save data accordingly to the current selected user

2.4. Must not ask any personal information about the user

2.5. The frontend and backend model

2.5.1. Must have a maximum inference of 500ms, so that enough time for showing and interpreting the predicted label exists

2.6. The frontend model

2.6.1. Must be cacheable by the browser

2.6.2. Must be proven to classify as the standard model

2.7. The servers

2.7.1. Must have maximum average latency of 150ms

2.7.2. Must handle multiple requests at once

2.7.3. Must be secured through TLS encryption

2.7.4. Must not save any personal information about the user

3. The neural networks

3.1. Must reach a test accuracy of at least 50%

3.2. Must define speech augmentation methods

3.3. Must prove these augmentation methods increase the test accuracy

3.4. Must use pre trained convolutional neural networks from the Keras framework

3.5. Must use stacked MFCC as inputs complying to the RGB channel specification of the used models

B. Frontend

Vue code for requesting the audio stream:

```
<template>
    <div id="micRequest">
    </div>
</template>

<script>
import Vue from 'vue';
import Meyda from 'meyda';

export default {
  mounted() {
    this.init();
  },
  methods: {
    /*
    get new audio
    context object
    */
    createAudioCtx() {
      const AudioContext = window.AudioContext || window.webkitAudioContext;
      Vue.prototype.audioCtx = new AudioContext();
      return Vue.prototype.audioCtx;
    },

    /*
    create microphone
    audio input source from
    audio context
    */
    createMicSrcFrom(audioCtx) {
      return new Promise((resolve, reject) => {
        const constraints = { audio: true, video: false };
        navigator.mediaDevices.getUserMedia(constraints)
          .then((stream) => {
            const src = audioCtx.createMediaStreamSource(stream);
            resolve(src);
          }).catch((err) => { reject(err); });
      });
    },
```

```
    show(features) {
      // update spectral data size
      Vue.prototype.curMFCC= features.mfcc;
      Vue.prototype.curRms = features.rms;
      Vue.prototype.curBuffer = features.buffer;
    },

    /*
    call given function
    on new microphone analyser
    data
    */
    onMicDataCall(features, callback) {
      return new Promise((resolve, reject) => {
        const audioCtx = this.createAudioCtx();
        this.createMicSrcFrom(audioCtx)
          .then((src) => {
            const analyzer = Meyda.createMeydaAnalyzer({
              audioContext: audioCtx,
              source: src,
              bufferSize: 512,
              featureExtractors: features,
              callback,
              numberOfMFCCCoefficients: 40,
            });
            resolve(analyzer);
          }).catch((err) => {
            reject(err);
          });
      });
    },

    init() {
      this.onMicDataCall(['mfcc', 'rms', 'buffer'], this.show)
        .then((meydaAnalyzer) => {
          meydaAnalyzer.start();
        }).catch((err) => {
          alert(err);
        });
    },
  },
};
</script>
```

Vue code for handling prediction:

```
    draw() {
      // append new MFCCvalues
      if (Vue.prototype.curRms > this.ThresRms) {
        this.mfccHistory.push(Vue.prototype.curMfcc.reverse());
      } else {
        this.mfccHistory.push(this.defaultMfcc);
      }
      // plot
      if (document.getElementById('bufferBar').style.visibility ===
      'visible') {
        this.plot_buffer();
      } else {
        this.plot_mfcc();
      }
      if (this.mfccHistory.length === this.mfccHistMaxLen) {
        // predict output
        this.predict(this.mfccHistory.slice());
        this.mfccHistory = [];
      }
    },
```

```
async predict(input) {
  this.toggleOutput();
  const stacked = tf.stack([input, input, input]);
  const reshaped = stacked.reshape([1, 40, 130, 3]).arraySync();
  this.$pouch.get('userCur', {}, 'account').then((user) => {
    this.$pouch.get('backendPredictionSwitchOn', {},
    `${user.name}home`).then((doc) => {
      if (doc.value === true) {
        this.backendPrediction(reshaped);
      } else {
        this.frontendPrediction(reshaped);
      }
    }).catch(() => {
      this.backendPrediction(reshaped);
    });
  }).catch((err) => {
    console.log(err);
  });
},

frontendPrediction(mfcc) {
  const modelPred = this.model.predict(mfcc);
  const predLabel = this.labelDict[tf.argMax(modelPred, tf.axis =
  1).dataSync()];
  this.postPrediction(predLabel);
},

backendPrediction(mfcc) {
  const Url = 'https://adr-ml.herokuapp.com/predict_api';
  axios.post(Url, mfcc)
    .then((response) => this.postPrediction(response.data));
},
```

C. Backend

Python Flask code for backend prediction:

```
import numpy as np
from flask import Flask, request, jsonify, render_template
from tensorflow import keras
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
model = keras.models.load_model('model.h5')

labelDict = {
   0: 'Angry',
   1: 'Disgusted',
   2: 'Scared',
   3: 'Happy',
   4: 'Neutral',
   5: 'Sad',
   6: 'Surprised',
}

@app.route('/')
def home():
   return render_template('index.html')

@app.route('/predict_api',methods=['POST'])
def predict_api():
   '''
   For direct API calls trought request
   '''
   data = request.get_json()
   mfcc = np.array(data)
   mfcc = np.stack((mfcc,) * 2)
   pred = labelDict[np.argmax(model.predict(mfcc))]
   return jsonify(pred)

if __name__ == "__main__":
   app.run(debug=True)
```

D. Deep Neural Network

Keras code for building the frontend and backend DNN:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import librosa
import librosa.display
import random
from scipy import signal
from scipy.io import wavfile
from tqdm import tqdm
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import keras
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import SimpleRNN, GlobalAveragePooling2D
from keras.applications.resnet import ResNet50
from keras import initializers
from keras.optimizers import RMSprop
import time

# Print Dataset Directory
data_dir = os.listdir('../../../train_ds/')
for data in data_dir:
    print(data)

# Import Dataset
data = pd.DataFrame(columns=['path', 'label'])

cnt = 0
for file in data_dir:
    nm = file.split('.')[0].split('-')
    path = '../../../train_ds/' + file
    label = int(nm[1])

    data.loc[cnt] = [path, label]
    cnt += 1

data.head()
```

```
# Testing imported data
filename = data.path[3]
print (filename)

samples, sample_rate = librosa.load(filename)

def log_specgram(audio, sample_rate, window_size=20,
                 step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, times, spec = signal.spectrogram(audio,
                                    fs=sample_rate,
                                    window='hann',
                                    nperseg=nperseg,
                                    noverlap=noverlap,
                                    detrend=False)
    return freqs, times, np.log(spec.T.astype(np.float32) + eps)

# Plotting Wave Form and spectrogram
freqs, times, spectrogram = log_specgram(samples, sample_rate)

fig = plt.figure(figsize=(14, 10))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + filename)
ax1.set_ylabel('Amp')
librosa.display.waveplot(samples, sr=sample_rate)

ax2 = fig.add_subplot(212)
ax2.imshow(spectrogram.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
ax2.set_yticks(freqs[::16])
ax2.set_xticks(times[::16])
ax2.set_title('Spectrogram of ' + filename)
ax2.set_ylabel('Hz')
ax2.set_xlabel('Seconds')

# CMN
mean = np.mean(spectrogram, axis=0)
std = np.std(spectrogram, axis=0)
spectrogram = (spectrogram - mean) / std
```

```
# Plotting MPS
mps = librosa.feature.melspectrogram(samples,
                                     sr=sample_rate,
                                     n_mels=40)

# Convert to log scale
log_mps = librosa.power_to_db(mps, ref=np.max)

plt.figure(figsize=(12, 4))
librosa.display.specshow(log_mps, sr=sample_rate,
                         x_axis='time', y_axis='mel')
plt.title('Mel power spectrogram ')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()

# Plotting MFCC
mfcc= librosa.feature.mfcc(y=samples, n_mfcc=40)

plt.figure(figsize=(12, 4))
librosa.display.specshow(mfcc)
plt.ylabel('MFCCcoeffs')
plt.xlabel('Time')
plt.title('MFCC')
plt.colorbar()
plt.tight_layout()

print(mfcc.shape)

data2 = data.copy()
data2 = data2.sample(frac=1)
data2.head()

# Convert dataset paths to mfcc
data = pd.DataFrame(columns=['feature', 'label'])

for i in tqdm(range(int(len(data2) * 0.2))):
    X, sample_rate = librosa.load(data2.path[i], duration=3,
    res_type='kaiser_fast')
    sample_rate = np.array(sample_rate)
    mfcc= librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40)
    mean = np.mean(mfcc, axis=0)
    std = np.std(mfcc, axis=0)
    mfcc= (mfcc - mean) / std
    mfccs = np.stack((mfcc,) * 3, axis=-1)
    data.loc[i] = [mfccs, data2.label[i]]
```

```
# Split test set before augmentation
X_test = data['feature'].values.tolist()
X_test = np.reshape(X_test, (int(len(data2) * 0.2), 40, 130, 3))
y_test = data.label
lb = LabelEncoder()
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

# define augmentation methods
def wnoise(data):
    amp = 0.03 * np.random.uniform() * np.amax(data)
    data = data.astype('float64') + amp *
                        np.random.normal(size=data.shape[0])
    return data

def shift(data):
    range = int(np.random.uniform(low=-1, high = 1) * 1500)
    return np.roll(data, range)

def pitch(data, sample_rate):
    intensity = 1.05 * np.random.uniform()
    data = librosa.effects.pitch_shift(data.astype('float64'),
                                       sample_rate, n_steps=intensity,
                                       bins_per_octave=12)
    return data

def volume(data):
    vol = np.random.uniform(low=0.5,high=1.5)
    return (data * vol)

# define augmentation split
def rnd_aug(audio_wav):
    perc = np.random.rand()
    if perc < 0.25:
        audio_wav = noise(audio_wav)
    elif perc < 0.5:
        audio_wav = shift(audio_wav)
    elif perc < 0.75:
        audio_wav = pitch(audio_wav, 44100)
    elif perc < 1:
        audio_wav = dyn_change(audio_wav)
    return audio_wav
```

```
def batch_generator(audio_wav_paths, labels, batch_size, istraining):
    lb = LabelEncoder()
    while True:
        batch_mfcc= []
        batch_label = []
        for i in range(batch_size):
            if istraining:
                rnd_idx = random.randint(int(len(train_set))
                X, sample_rate = librosa.load(data.path[rnd_idx],
                                              duration=3,
                                              res_type='kaiser_fast')
                label = data.label[rnd_idx]
                aug_audio_wav = rnd_aug(X)
            else:
                rnd_idx = random.randint(int(len(validation_set))
                X, sample_rate = librosa.load(data.path[rnd_idx],
                                              duration=3,
                                              res_type='kaiser_fast')
            label = data.label[rnd_idx]
            mfcc= librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40)
            mean = np.mean(mfcc, axis=0)
            std = np.std(mfcc, axis=0)
            mfcc = (mfcc- mean) / std
            mfccs = np.stack((mfcc,) * 3, axis=-1)
            batch_mfcc.append(mfccs)
            batch_label.append(np_utils.to_categorical(label, label_amt))
        yield(np.asarray(batch_mfcc), np.asarray(batch_label))

# define Model
model = keras.models.Sequential([
    keras.applications.MobileNetV2(include_top=False,
             weights = 'imagenet',
             input_tensor = None,
             input_shape = (40, 130, 3),
             pooling = None,
             classes = 7),
    keras.layers.Flatten(),
    Dense(7, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
# train model
hist = model.fit_generator(batch_generator(data2.path, data2.label, 20, 1),
                           steps_per_epoch=50,
                           epochs=20,
                           validation_data=batch_generator(data2.path,
                           data2.label, 20, 0),
                           validation_steps=20,
                           verbose=1,
                           shuffle=1)

# plot model
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Acc')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```