School of Engineering

Emotion Detection

An IOS App that detects emotions via a Neuronal Network

Project Documentation

Employer	ZHAW School of Engineering
Authors	Valerio Serafini
	Basil Zinsli
Supervisors	Martin Loeser
	Matthias Rosenthal
Place, Date	Winterthur, 22.12.2023

Declaration of Independence

Declaration regarding the independent writing of a project paper at the School of Engineering. By submitting this project paper, the students assure that they have written the paper independently and without outside help. (In the case of group work, the work of the other group members does not count as external help). The undersigned student declares that all cited sources (including Internet pages) in the text or appendix are correctly referenced, i.e., that the project work does not contain any plagiarism, i.e. no parts that have been partially or completely taken from another person's text or work while claiming authorship or without citing the source.

In the event of misconduct of any kind, paragraphs 39 and 40 (dishonesty and procedure in the event of dishonesty) of the ZHAW Examination Regulations and the provisions of the disciplinary measures of the University Regulations shall apply.

Place, Date Zürich 21.12.2023

Signature Winterthur 21.12.23

Abstract

This document details the development of an innovative iOS application aimed at detecting emotions through a neural network, a project undertaken by the ZHAW School of Engineering. This project extensively explores fields like audio processing, Mel Coefficients computation, and the application of neural network algorithms for precise emotion recognition. It not only addresses the technical aspects of emotion detection but also explores the ethical implications of such technology. The potential applications of this technology are vast, ranging from healthcare, where it can aid in psychological assessments, to music production, enhancing emotional resonance in compositions, and even law enforcement, for better understanding of suspect behaviors. The development involved tools like Python, TensorFlow, Xcode, and Swift, focusing on real-time audio sample manipulation and custom audio effects. A neural network, trained with about 14,000 audio files using TensorFlow, was converted into a TensorFlow Lite model for integration into Swift using "CocoaPods". The final product is capable of recording voice memos, processing them through the model, and visually representing emotions with emojis.

Foreword

This is our project work, which serves as preparation for the bachelor's thesis next semester in our studies at the ZHAW School of Engineering. Important insights about audio manipulation and neural networks were developed, including in Xcode and TensorFlow. We would especially like to thank our supervisors Martin Loeser and Matthias Rosenthal for their pleasant support.

The complete details and resources of our entire project are available on our Git repository for further reference: **https://github.zhaw.ch/zinslbas/PA_IOSAPP.git.**

Table of Contents

1 Intro

1.1 State of the art

1.1.1 Existing Emotion Detection

Deep learning and neural networks: Advances in the field of deep learning have made it possible to recognize complex patterns in large amounts of data. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) networks, are often used for emotion recognition. [1]

Despite advances, accurate emotion recognition remains a challenge. Emotions are subjective and can be expressed culturally differently, making it difficult to develop universally applicable models. Emotion recognition systems have applications in a variety of areas, including customer care, healthcare (e.g. in detecting depression or anxiety) [2], education and in the automotive sector (e.g. to detect fatigue in drivers). [3]

1.1.2 Motivation

As of today, there is no iOS application that uses a neural network for the detection of human emotions. This gap in the market is particularly notable given the impressive strides made in large language models, which simulate human-like interactions with remarkable accuracy. The incorporation of emotion detection capabilities into these models could unlock a multitude of innovative applications, particularly in sectors such as healthcare and entertainment.

In the healthcare industry, the potential for emotion detection systems is extensive. Applications could range from therapy bots capable of recognizing and responding to patients’ emotional states, to virtual diaries that provide empathetic feedback. For mental health professionals, such systems could serve as valuable adjunct tools, offering a supplementary perspective to their clinical assessments. This technology could enhance therapeutic interventions by providing a nuanced understanding of patients' emotional journeys.

Furthermore, in the field of music production, the ability to detect and analyze a singer's emotions in real-time during studio recordings could revolutionize the creative process. This technology could offer insights into the emotional depth and authenticity of performances, enabling producers and artists to refine their work with unprecedented precision.

In law enforcement, emotion detection systems could be explored as a modern alternative to traditional polygraph tests. By analyzing subtle cues in facial expressions and vocal tones, these systems could provide additional layers of insight during interrogations, potentially increasing the accuracy of lie detection methods.

2 Existing Theoretical Concepts

2.1 The Process of Recoding Audio

The starting point of every audio recording is the sound source. This can be voices, musical instruments, electronic sound generators or ambient noise. Each sound source has unique characteristics such as frequency spectrum, dynamic range, and timbre. These characteristics determine the requirements for the recording technique and the choice of equipment. Microphoning is a crucial step in the recording process. The air pressure of the audio signal usually vibrates the membrane in the microphone and converts these sound waves into electrical signals. This is done using various transduction principles, such as the movement of a diaphragm in a magnetic field in the case of dynamic microphones, or by changing the capacitance of a capacitor in the case of condenser microphones.[4] The choice of microphone and its positioning are decisive for the quality of the recording, as they influence the directional characteristic and the frequency response of the recorded sound. After the microphone has converted the sound into an electrical signal, this signal must be amplified to be suitable for further processing. The preamplifier raises this signal to a higher level. A high-quality preamplifier is crucial as it minimizes the background noise and amplifies the signal without noticeable distortion. [5]

The next step is to process the signal. Equalizers are used to boost or cut certain frequency ranges, which is used to shape the sound or compensate for room acoustics effects. Dynamic processors such as compressors and limiters regulate the volume and dynamics of the signal. They ensure that the signal has an even level and that peak volumes are limited. Effect devices such as reverb or delay add spatial depth or special sound effects to the signal. These days most analog effects are digitally available in the form of VST Plug-ins. The analog signal that has now been processed is converted into a digital format by an audio-to-digital converter (AD converter). This conversion is crucial as it transfers the quality of the analog signal into a digital form. Important factors here are the sampling rate and the bit depth. [5]

Sample rate refers to the number of samples taken per second from an analog audio signal to convert it into a digital format. Measured in Hertz (Hz), the sample rate determines how accurately the digital signal mirrors the analog original. Standard values such as 44.1 kHz for CDs and 48 kHz for audio tracks in videos are common, with higher values such as 96 kHz or 192 kHz being used in high-end audio productions to capture even more detailed sound images. The bit rate, measured in kilobits or megabits per second, indicates how much data is used for the audio signal per second. A higher bit rate generally means higher audio quality, as more information is available to describe the sound, but this also increases the storage requirements. The bit depth determines how detailed the amplitude of each sample in a digital audio signal is recorded. A higher bit depth, such as 24-bit in professional audio environments, allows for a more precise representation of the dynamic range, allowing softer sounds and louder peaks to be reproduced without distortion and with fine resolution. The dynamic range describes the difference between the quietest and loudest tones of an audio signal. A greater dynamic range in a digital audio system enables a more realistic reproduction of recordings, which is particularly important in high-fidelity audio systems. Finally, the frequency range defines the bandwidth of frequencies that an audio system can record or reproduce. The human hearing range extends from about 20 Hz to 20 kHz, and high-quality audio equipment is designed to cover this range completely. [5]

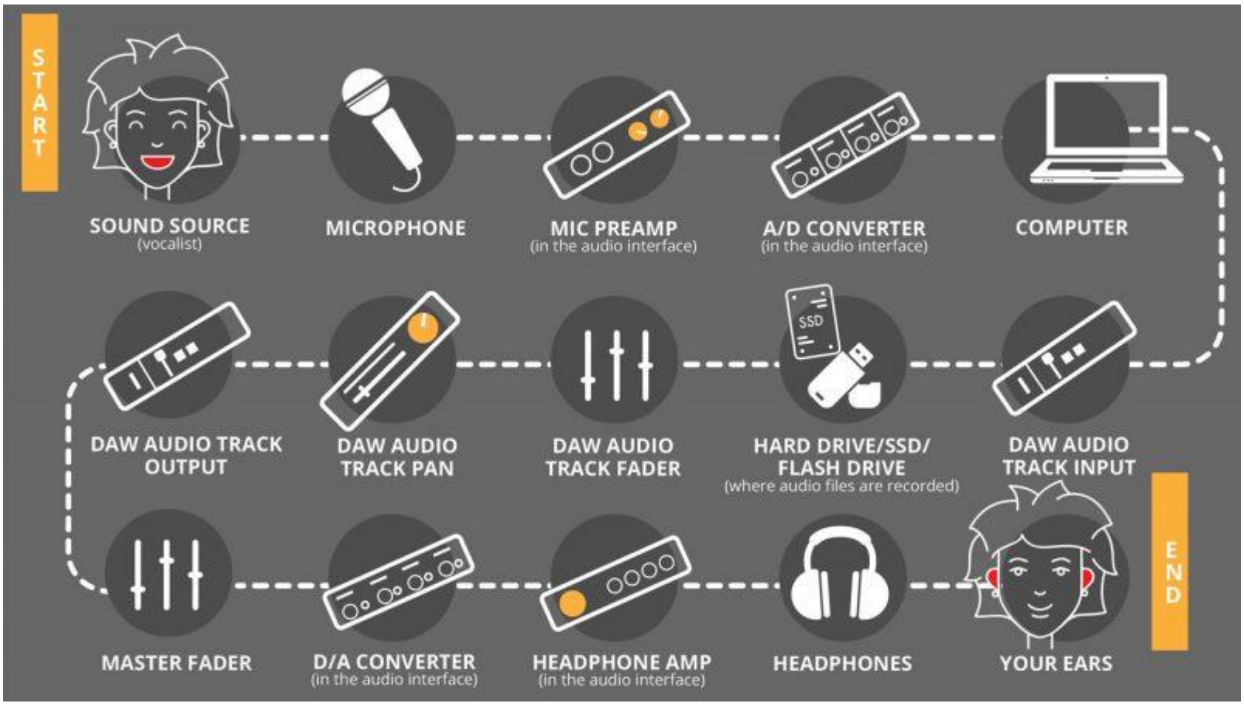


Figure 1 Audio Signal Flow Chart[6]

2.2 The Process of Playing Audio

Playing a digital audio file through to its perception by our ears is a fascinating process that involves both modern technology and the complexity of human hearing. First, the audio is in a digital format, such as an MP3 or WAV file. This file stores the audio signal as a series of digital samples defined by the sample rate and bit depth when originally recorded. During playback, the file is read by a playback device such as a computer, smartphone, or specialized audio player. The digital audio signal must be converted into an analog signal, as only analog signals can be played back by loudspeakers or headphones. This is done by a digital-to-analog converter (DAC). The DAC reconstructs the original analog audio signal by converting the digital samples into continuous voltage changes that represent the sound waves. The quality of the DAC is crucial for the quality of the sound. The analog signal is then often amplified in order to achieve a sufficient signal level for playback. This is particularly important for power-intensive output devices such as loudspeakers. The amplified signals then drive the loudspeakers or headphones. In speakers, the signal causes the speaker membranes to move and emit sound waves into the environment.

2.3 Perception of sound waves

The generated sound waves propagate in the environment and finally reach the human ear. Here they are caught by the pinna and pass through the auditory canal to the eardrum, which begins to vibrate. The vibrations of the eardrum are amplified by the ossicles in the middle ear and transmitted to the inner ear. The inner ear contains the cochlea, a fluid-filled spiral filled with thousands of hair cells. These hair cells convert the mechanical vibrations into electrical nerve signals. The nerve signals are transmitted via the auditory nerve to the brain, where they reach the auditory cortex. This is where complex processing of the signals takes place, enabling us to identify, localize and interpret sounds. A key component of this processing is real-time Fourier transformation, where the brain breaks down the complex sound signal into its individual frequency components. This analysis allows us to recognize and distinguish between different sounds, pitches and timbres. [4]

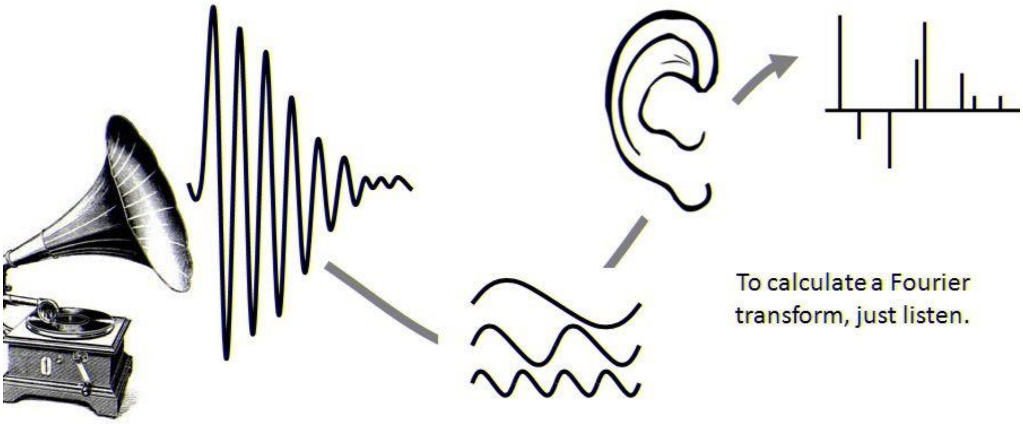


Figure 2 Fourier Transformation via auditory nerve[7]

2.4 Fourier Transformation

In a complex system that integrates TensorFlow, neural networks and emotion recognition, the Fourier transform plays a central role. The mathematical method is crucial for processing and analyzing audio signals, a key element in emotion recognition. First, the Fourier transform is used to convert time or space signals into frequency signals. This transformation makes it possible to identify patterns or characteristics in the frequency components of an audio signal that may not be apparent in the original time or space domain. This transformation is particularly useful in audio processing, as it makes it possible to extract essential features from audio signals that contain information about frequency distribution, harmonics, and other relevant aspects.

Within a TensorFlow-based system that uses neural networks, the Fourier transform is a key tool for pre-processing audio data. By converting the audio signals into the frequency domain, certain characteristics of the audio that are important for training the neural network can be emphasized. This is particularly effective for neural networks such as Convolutional Neural Networks (CNNs), which are specifically designed to process audio data. Using frequency features instead of raw data in the time domain can make training these networks more efficient and effective.

In emotion recognition, the extraction of features that indicate emotional states is crucial. The Fourier transform helps identify features such as intonation, pitch and timbre in the human voice that can provide emotional cues. Providing a detailed frequency analysis of the audio signal through the Fourier transform can increase the accuracy of classification algorithms in neural networks. This accuracy is essential in emotion recognition, as subtle nuances in the audio signal are often crucial to identify the correct emotion.

2.5 Xcode

Xcode is Apple's integrated development environment (IDE), which is specially designed for developing applications on iOS, macOS, watchOS and tvOS. It offers developers a variety of tools for designing, programming, and debugging applications. The Interface Builder in Xcode makes it easy to create user interfaces, while the Code Editor offers features such as syntax highlighting and code completion for Swift and Objective-C. Xcode integrates advanced compilers and a powerful build system as well as a debugger for efficient debugging.

A particular advantage of Xcode is the ability to integrate TensorFlow models into projects via CocoaPods. CocoaPods is a dependency manager for Swift and Objective-C projects that allows developers to easily integrate external libraries and frameworks, such as TensorFlow, into their Xcode projects. This integration makes it easier to implement machine learning and advanced data analysis functions in apps.

In addition, Xcode provides a simulator for testing apps and simplifies the process of provisioning and code signing. The tight integration with Apple services and the provision of extensive documentation and learning resources make Xcode an indispensable tool for developers working on Apple platforms. [8]

2.6 Swift

Swift is a modern and powerful programming language designed by Apple for the development of iOS, macOS, watchOS and tvOS applications. It is characterized by the following features:

- Security: Swift minimizes common programming errors through a secure design.
- Speed: The language is optimized for high performance and efficiency
- Modern syntax: Swift offers a clear and concise syntax that simplifies programming.
- Interoperability with Objective-C: Swift can be seamlessly combined with Objective-C, making it easier to migrate existing projects.

Swift is particularly user-friendly thanks to features such as interactive "playgrounds" for immediate code testing. It has an active developer community and is continuously developed, supported by extensive resources and documentation. Mainly used in app development, Swift is also gaining traction in server-side programming and as a scripting language. Its flexibility and future-oriented development make Swift a key language in modern software development. [9]

2.7 AVFoundation

The AVFoundation Library is a comprehensive framework from Apple that is used for audio and video editing in iOS, macOS, watchOS and tvOS. It offers developers a powerful interface for working with multimedia content. AVFoundation is often used in apps that require functions such as video playback, audio recording, music playback or the creation and editing of multimedia content. The framework is suitable for both simple and complex multimedia applications and offers developers a high degree of flexibility and control. [10]

Reasons AVFoundation was chosen for this project:

- Media playback and recording: The library enables the playback and recording of audio and video content. It supports various media formats and codecs.
- Editing and processing: AVFoundation offers functions for editing audio and video files, such as cutting, merging, and converting file formats.
- Access to camera and microphone: The framework allows developers to access the built-in camera and microphone devices to take photos or record audio and video.
- Advanced audio functions: Developers can perform complex audio processes, such as mixing and adjusting audio tracks, implementing audio effects and processing audio signals in real time.

2.8 TensorFlow

TensorFlow is an open-source machine learning library developed by Google. It is known for its flexibility and wide range of applications in various areas of artificial intelligence, such as deep learning, neural networks, and data analysis.

Reasons TensorFlow is fitting for this project:

- Versatility: TensorFlow supports a wide variety of algorithms and models, making it a favorite tool for researchers and developers in the world of machine learning.
- Scalability: The library is known for its ability to work on small devices such as smartphones as well as on large, powerful servers. This makes TensorFlow particularly attractive for projects that require scaling from prototypes to large, production-ready systems.
- Graphical representation of data flows: TensorFlow uses a graphical approach to represent data flows, making it easier to construct and understand complex networks and algorithms.
- Extensive community and resources: As an open-source project, TensorFlow has a large and active community that regularly contributes to the development and improvement of the library.

TensorFlow is used in a wide range of applications, from simple classifications to complex systems such as self-driving cars or real-time translation services. Its ability to handle large amounts of data and train complex models makes it a powerful tool in the world of machine learning and artificial intelligence. Overall, TensorFlow offers developers and researchers a comprehensive and customizable platform for the development of machine learning and artificial intelligence, supported by a strong community and extensive resources. [11]

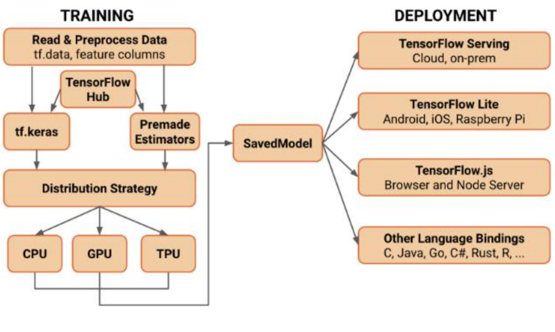


Figure 3 TensorFlow training Flow Chart[12]

2.9 Emotion Detection via a Neural Network

Emotion recognition with neural networks is an area of artificial intelligence that aims to identify and classify human emotions from various data sources such as text, speech, facial expressions, or physiological signals. This process begins with the collection of relevant data, which can vary depending on the area of application. Once the data has been collected, a pre-processing phase follows. Here, the data is cleaned and normalized to prepare it for training the neural network. This can include removing background noise from audio data, standardizing text data or adjusting images to a uniform size and format.

A neural network is then trained with this pre-processed data. The network learns to recognize patterns and features in the data that are typically associated with certain emotions. In speech analysis, for example, this could be timbre of voice, volume, and speed of speech. After training, the neural network can be used to recognize emotions in new, unfamiliar data. For example, when a new voice recording is analyzed, the network evaluates it based on the features learned during training and assigns it to one or more emotions. The challenges in this area lie not only in the technical implementation, but also in the accuracy and emotion recognition. As emotions are complex, subjective and have cultural differences, the technology must be constantly developed and adapted to different contexts. [13]

2.9.1 Background and Key Terms of Neural Networks

Machine Learning is a wide-ranging concept that encompasses numerous topics, including Deep Neural Networks, which fall under the category of Machine Learning/AI. This discussion will clarify important terms and background in this area. In Deep Neural Networks, there's a distinction between classification, where relevant features produce a discrete output, and regression, which uses features to generate a continuous output. Our focus is on classifying unstructured data and associating it with emotions. Other key terms include Supervised and Unsupervised Learning. Supervised Learning uses labeled data to train a model for making predictions on new data, whereas Unsupervised Learning operates without labeled data, as shown in figure 4. In our case, we used labeled data in a Supervised Learning mode, which is a commonly used approach. This also broadly encompasses the concept of Neural Networks. [13]

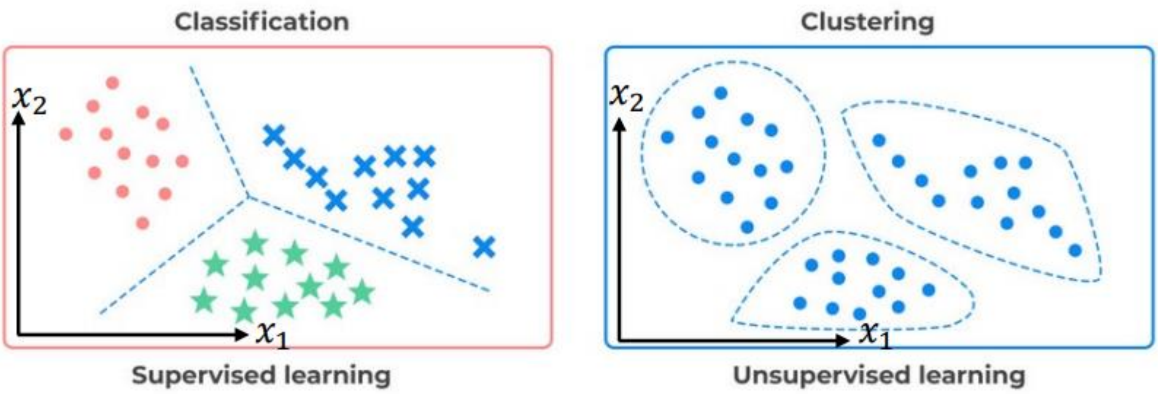


Figure 4 Difference between supervised and unsupervised learning [14]

2.9.2 Structure of Neural Networks

A neural network is a network composed of individual neurons, which serve as nodes. It consists of an input layer of a specific size, one or more hidden layers for computation, applying weights and biases, and an output layer formatted to the desired output. Non-linear activation functions enable the activation of individual neurons based on type and weighting. Additionally, during the training phase, the network adjusts these weights and biases to minimize errors, a process known as back-propagation. This structure allows the neural network to learn from data and make complex pattern recognitions. [13]

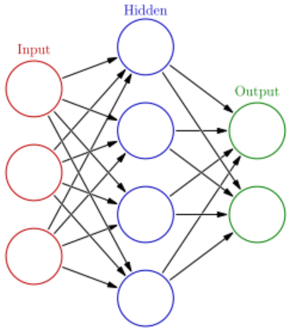


Figure 5 Structure of a Neural Network [14]

2.9.3 Artificial Neurons

To activate neurons and adjust weights in artificial neural networks, the process can be computed as follows:

$$v = w1x1 + w2x2 + w3x3 + b = w\hat{}T\,x + b$$

$$y = \varphi(v) = \varphi(w\hat{}T\,x + b) = \varphi(x\hat{}T\,w + b)$$

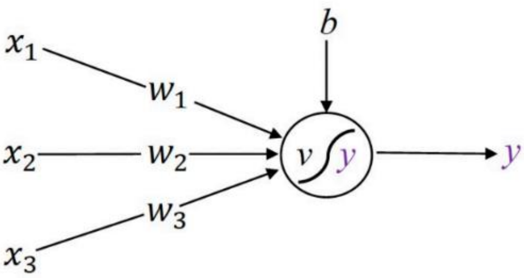


Figure 6 Artificial Neuron with inputs weights and bias [14]

v: *The weighted sum of inputs and bias*

$w1, w2, w3$: *Weights applied to each input*

$x1, x2, x3$: *Input values*

b: *Bias term added to the weighted sum*

y: *Output of the neuron*

$\varphi(v)$: *Activation function applied to the weighted sum*

The output y of a neuron is termed as its activation or output. This output is obtained by passing the weighted sum of the inputs through an activation function $\varphi()$.

These formulas are crucial for determining the output of each neuron, considering the inputs and weights. [13]

2.9.4 Activation Functions

In terms of activation functions, there are several different types, with the most well-known being the following three.

Threshold Function is one of the most trivial ones it activates if input v is greater than or equal to zero:

$$\varphi(v) = \begin{cases} 1, & v < 0 \\ 0, & else \end{cases}$$

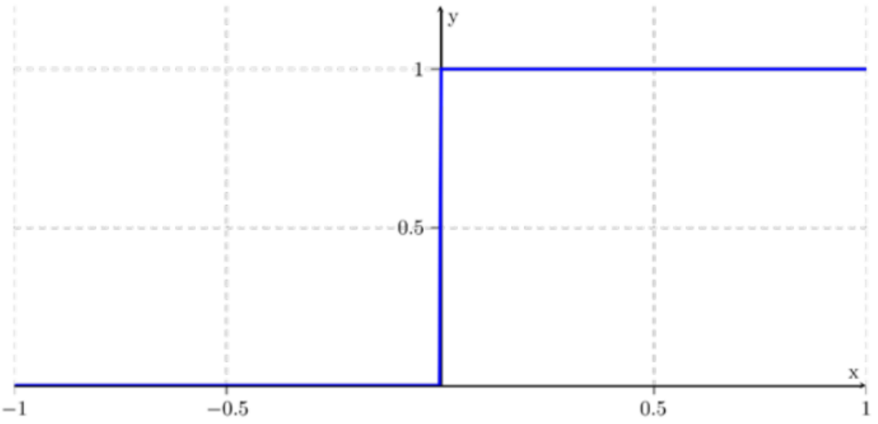


Figure 7 Threshold activation function [15]

Sigmoid Function, Gradually activates neurons between 0 and 1, often used for binary classification. [13]

$$\varphi(v) = \frac{1}{1 + exp(-a * v)}$$

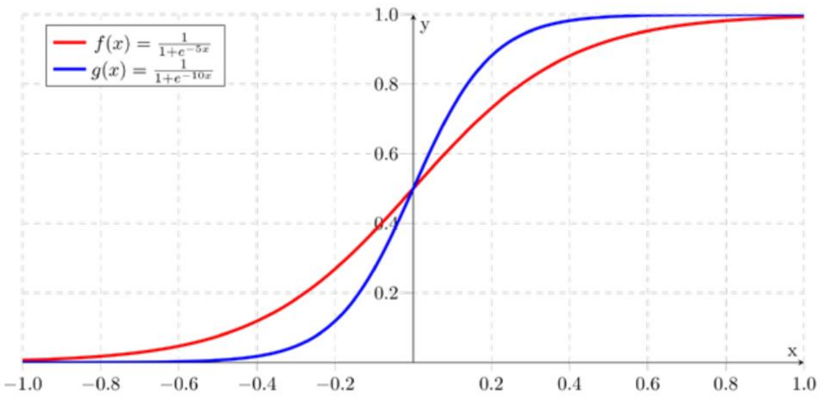


Figure 8 Sigmoid activation function [14]

The Rectified Linear Unit (ReLU) activates neurons linearly for positive inputs while setting negative inputs to zero. It is considered the most used activation function. [13]

$$\varphi(v) = max(0, v)$$

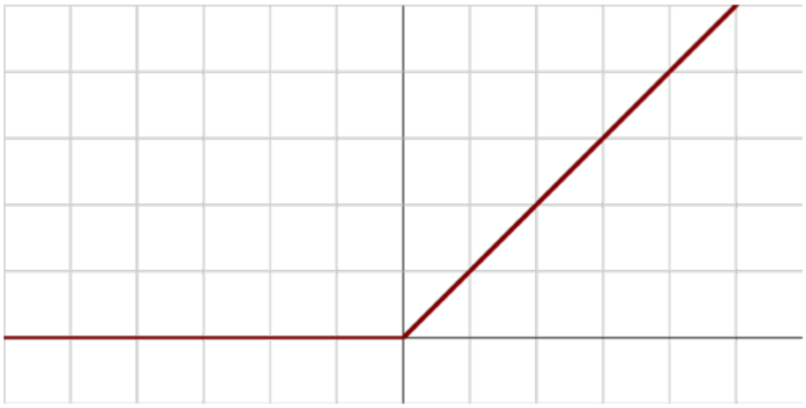


Figure 9 Rectified Linear Unit activation function [15]

2.9.5 Model Types

In the domain of neural networks, a variety of topologies and architectural designs exist to cater to different computational tasks. This has led to the creation of diverse models and variations, each optimized for specific applications.

RNN Recurrent Neural Networks

Recurrent Neural Networks are a type of artificial neural network designed for sequential or time series data. They are key in solving ordinal or temporal problems, such as language translation, natural language processing, speech recognition, and image captioning, and are used in applications like voice-activated assistants. RNNs are distinct in their ability to use "memory" from previous inputs to influence current processing. [16]

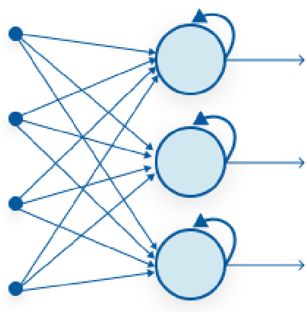


Figure 10 Recurrent Neural Network illustration [16]

LSTM Long Short-Term Memory

Long Short-Term Memory is a type of specifically a variant of Recurrent Neural Networks. It's designed to overcome the vanishing gradient problem faced by traditional RNNs. LSTM networks are capable of learning and retaining information over long sequences, making them ideal for tasks involving sequential data like natural language processing and speech recognition. [17]

CNN Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep neural network that's good at processing data with a grid-like structure, like images. They have a layered hierarchy, with each layer having a set of filters that it learns to apply to the input data. As these filters move over the input, CNNs pick out important features from the data and keep track of where things are in relation to each other.

CNNs are also commonly used for speech and emotion recognition. In these cases, instead of images, the input is a spectrogram or a Mel-frequency cestrum coefficients (MFCC) spectrogram, which visually represents the signal's frequency and amplitude over time. These spectrograms are treated like images, allowing the CNN to scan for patterns that correspond to specific sounds or emotional tones in the speech, much like it would look for patterns in a picture to identify objects. [13]

2D CNN

In our work, we specifically focused on 2D Convolutional Neural Networks (CNNs) for image processing (Spectrogram processing). As demonstrated in figure 11, a 2D CNN operates by applying filters, or kernels, to an input image to capture features. Each pixel's value, typically between 0 and 255, represents the RGB color intensity. The filters, commonly sized 3x3 or 5x5, move over the image in predetermined stride lengths. At each step, the CNN calculates a correlation by multiplying the filter's values with the pixel values and summing them, as shown in figure 11. This creates a feature map that accentuates the image's patterns. The CNN is trained to determine the best filter values for efficient feature recognition. [13]

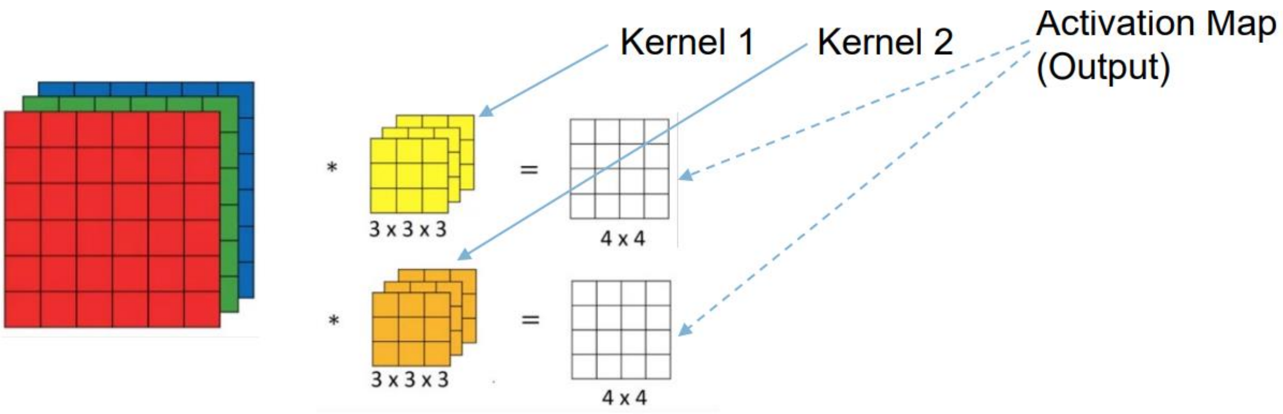


Figure 11 Concept of 2D CNN with 2 Kernel [14]

A neural network often consists of more than just a convolutional layer. It includes elements like pooling layers, which reduce the size of the feature maps using techniques such as max or average pooling, as the examples on picture 12 shows for MaxPooling.

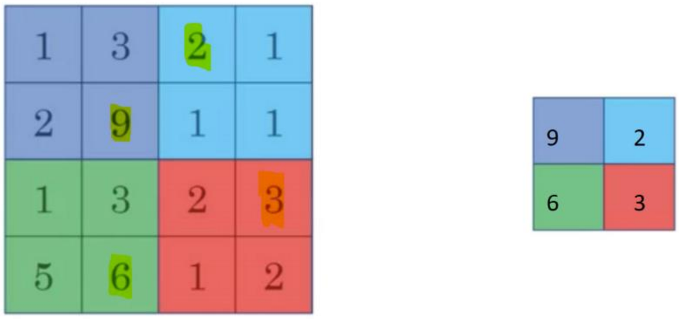


Figure 12 Concept of MaxPooling [14]

Following this, there are dense layers that contribute to decision-making by interpreting the features, and a flattening step that transforms the 2D feature maps into a 1D vector. This vector is concatenated and processed through additional dense layers, culminating in an output layer that frequently employs the SoftMax function to provide probabilistic classification outputs. Figure 13 demonstrates the series of steps applied to an example of a handwritten digit '7'. [13]

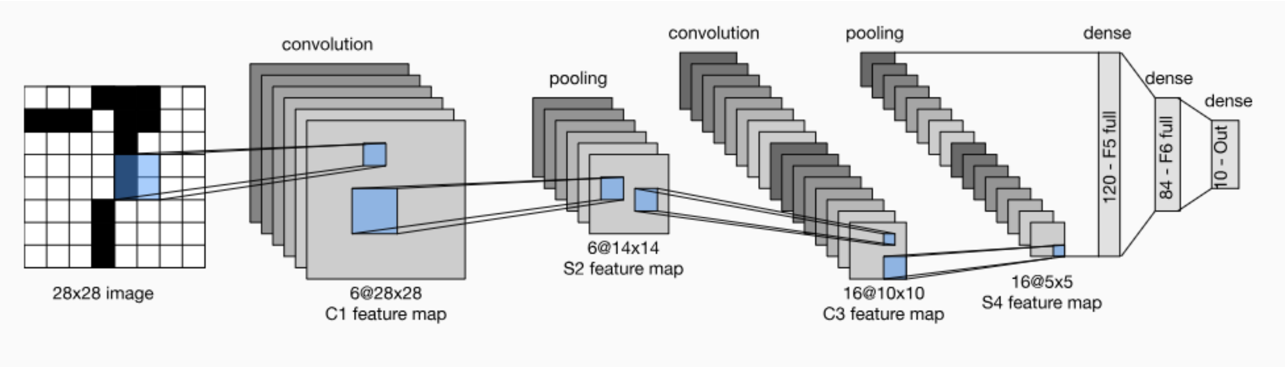


Figure 13 Concept of a 2D CNN Model with all commonly used layers [15]

2.9.6 Data Collection

Data collection and the quality of datasets are crucial in neural network applications, thus the dataset is a pivotal component. Data can be self-generated or downloaded from repositories one of the largest communities for datasets is Kaggle, which offers widely used and high-quality data along with labels. In the realm of emotion recognition, two benchmark datasets are the CREMA-D and RAVDESS datasets.

In the realm of datasets, the most critical aspect is to know the methods used for data collection and the quality of labeling. A large dataset is of limited value without labels, as it cannot be utilized for supervised learning, which relies heavily on labeled data for model training and validation. [13]

2.10 Preprocessing Data

In audio emotion recognition, preprocessing data is crucial for effective neural network performance. Mel Frequency Cepstral Coefficients (MFCCs) are considered state-of-the-art in this field. MFCCs are features derived from the audio signal that efficiently represent the power spectrum of sound. They are obtained by applying a Mel-scale filter to the power spectrum, capturing the perceptually relevant aspects of the audio. This process helps in emphasizing the features that are important for human hearing, making MFCCs particularly suitable for tasks involving speech and emotion recognition. The use of MFCCs in preprocessing ensures that the neural network receives data that is both relevant and in a format conducive to learning and identifying emotional cues in audio signals. [18]

2.10.1 Preprocessing Steps

Normalizing Audio

Normalizing audio involves adjusting the waveform of the audio track to have a standardized loudness level. This is achieved by calculating the mean and standard deviation of the waveform's amplitude. The waveform is then transformed so that its mean is zero, and its variance is one, by subtracting the mean from the waveform and dividing by the standard deviation. This process ensures that the audio maintains a consistent volume level, which is important for analyzing the data in a neural network, as it helps to reduce variability caused by volume differences.[19]

$$normalized\_waveform = (waveform - mean) / std\_dev$$

$normalized\_waveform$: *The waveform after normalization.*

$waveform$: *The original audio waveform data.*

$mean$: *The average amplitude of the waveform.*

std_dev: *The standard deviation of the waveform's amplitude.*

FFT Fast Fourier Transformation

The Fourier Transform transforms a signal from the time domain to the frequency domain. This transformation is crucial for analyzing the frequency components of time-based signals, like audio. It decomposes a time-domain signal into its constituent frequencies, revealing the amplitude and phase of each frequency component. This step is the base for Audio Emotion Recognition.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e_{-j\omega t}\, dt$$

$F(\omega)$: *Fourier Transform of the signal.*

$f(t)$: *timedomain signal.*

ω: *angular frequency.*

Spectrogram

The creation of a spectrogram from frequency domain data involves visually representing the spectrum of frequencies in an audio signal as it varies over time. After applying the FFT to the audio, the spectrogram maps time on the horizontal axis and frequency on the vertical axis. The intensity of each frequency at each point in time is depicted using color or grayscale levels, providing a detailed view of the audio signal's spectral properties.

Mel coefficients and spectrogram

After the FFT and spectrogram creation, MFCCs are computed through a process that aligns closely with human auditory perception. It begins with dividing the audio signal into short frames and computing the power spectrum for each. These spectra are then passed through a Mel scale filter bank as shown on figure 14 on the left side, emphasizing frequencies most relevant to human hearing. The outputs of these filters are transformed logarithmically and a Discrete Cosine Transform (DCT) is applied. The coefficients obtained from DCT, known as MFCCs, effectively capture the key features of the audio signal, making them invaluable for applications like speech and emotion recognition. The result is a set of MFCCs that provide a compact and perceptually meaningful representation of the sound. [20]

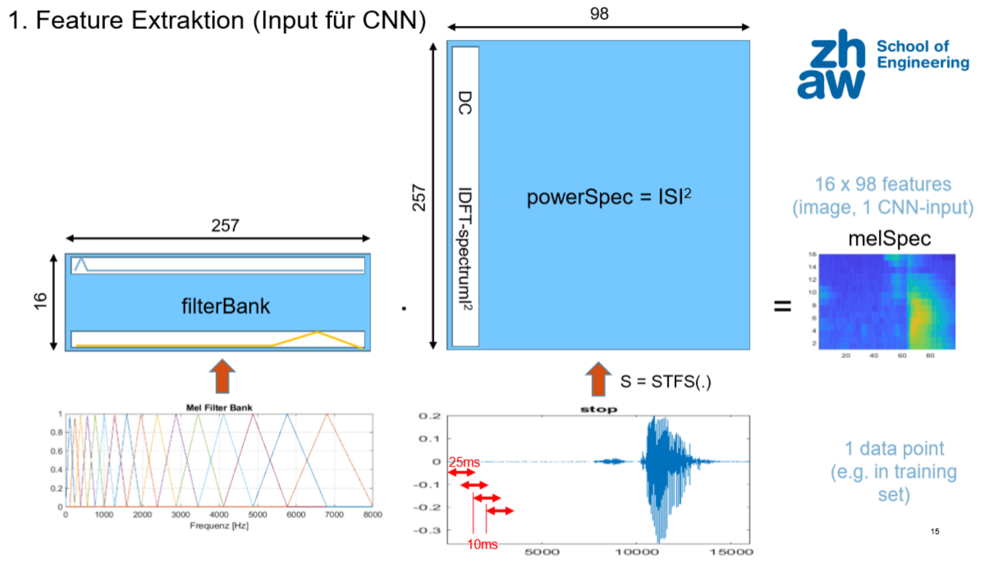


Figure 14 Mel Spectrogram [14]

Integration into Neural network

Once preprocessed, MFCCs are integrated into neural networks for emotion recognition. These MFCCs, which encapsulate crucial audio features in a compact form, are fed as input to the neural network. The network then learns to associate patterns in these features with specific emotional states, allowing it to effectively recognize and classify emotions expressed in the audio signals. This integration is key in developing systems capable of understanding and interpreting the nuanced emotional content in spoken language. [13]

3 Concept/Methods

3.1 Testing the AVFoundation Library

In the initial phase of our project within Xcode, the primary task involved loading an audio file into a buffer to access its samples. These samples were subsequently decoded and played. The AVFoundation library offers various nodes that enable manipulation of these samples, allowing for the application of effects. Additionally, the use of state variables enables the control of audio playback through a user interface. Additionally, the system was capable of fetching royalty-free music through an API and playing back the music. However, we encountered limitations as the effects were restricted to being applied to the entire buffer and could not be modified at runtime. This limitation arose because once a node is connected, it cannot be altered. [10]

3.2 Buffer size

To facilitate the application of effects during runtime, it was crucial to divide the entire audio file into smaller chunks, for example, 1-second parts or those equivalent to 44100 Hz. A function was developed to achieve this, taking a buffer containing the full audio file and outputting an array of smaller buffers. This step was vital for advanced audio processing, whether it involved applying effects or processing through a neural network using TensorFlow. [11]

However, a challenge arose with the structure of the library: it does not provide a means to precisely determine the timing of each sample played, as every chunk is scheduled in a First-In-First-Out (FIFO) style buffer. To address this, we implemented a system that triggers the next sample at the latest feasible moment, based on an estimated processing time for each effect. This approach, while innovative, resulted in a somewhat flexible and less precise solution. Nonetheless, it now enables the toggling of various audio effects in real-time.

In our assessment, we reached the limitations of what could be achieved with the AVFoundation library for our specific goals. Constructing an entire audio player from the ground up would be a potential solution, but this presents considerable challenges. These include complexities in designing real-time audio processing capabilities, ensuring seamless audio playback, and integrating advanced features like dynamic effect application, all of which require significant technical expertise and resources.

3.3 Delay effect

To incorporate an audio effect independently, we executed the implementation of a delay effect. This process involved sequentially writing each audio chunk, while incorporating an overlay of delayed samples. This technique allowed for the creation of a delay effect through careful manipulation and timing of the audio samples, demonstrating a practical application of our custom audio processing capabilities.

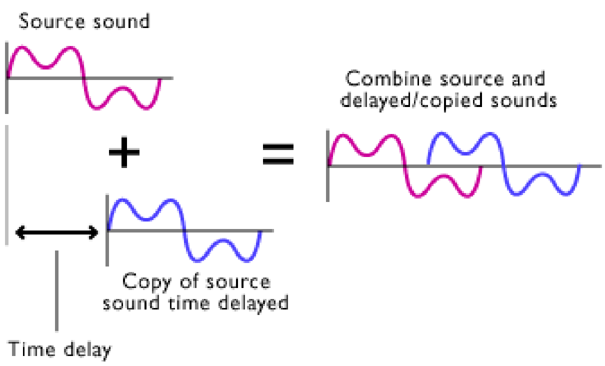


Figure 15 Concept of Audio delay effect [21]

3.4 Butterworth Lowpass Filter

In the subsequent phase, a first-order Butterworth filter was implemented. To achieve this, we calculated the necessary coefficients using MATLAB. MATLAB's robust computational tools enabled us to accurately determine the filter coefficients, which are essential for defining the filter's frequency response characteristics. Once these coefficients were calculated, we integrated them into our audio processing framework, allowing for the effective application of the Butterworth filter to our audio signals. This approach exemplified a blend of theoretical analysis and practical application, leveraging MATLAB's analytical capabilities to enhance our audio processing system. [22]

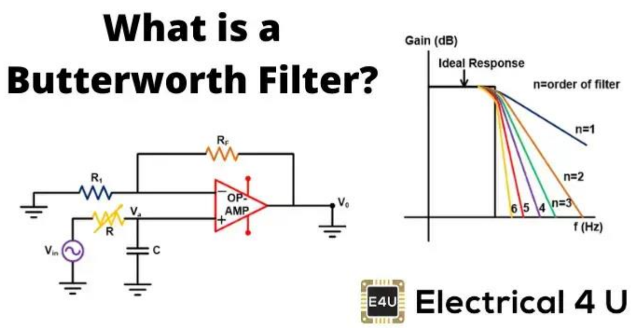


Figure 16 Schematic and concept bode plot of a Butterworth Lowpass Filter[23]

3.5 Deep Neuronal Network

For emotion detection, we implemented our neural network in Python using TensorFlow. This was carried out in both PyCharm and Jupyter Notebook environments. The emotions were selected based on datasets that included the following emotions: Fear, Disgust, Joy, Neutral, Sadness, and Anger. These emotions were consistently present across the datasets we used.

3.5.1 Datasets

We selected the CREMA-D and RAVDESS datasets for their comprehensive range of emotional expressions and high-quality, diverse actor representations. CREMA-D includes 7,442 Audio clips from 91 actors, and RAVDESS provides 1,440 files from 24 actors, each dataset encompassing a spectrum of emotions and intensities, delivered in a neutral North American accent. To complement these datasets and simulate real world scenarios, we generated synthetic noise using Python scripts, creating white, brown, violet, and pink noise across a frequency range of 20 to 2000 Hz. This process was aimed at enhancing the robustness of our emotion recognition system by introducing a variety of non-emotional sounds that could occur in natural environments. [24]

In total, our collection has over 14,700 audio WAV files, encompassing six distinct emotions, with each file serving as a data point for our model training.

As illustrated in the histogram in Figure 17, our dataset comprises nearly 6000 'None' data points, with an average of around 1450 data points per emotion. This distribution reflects the varied emotional states captured in our audio dataset, with a significant emphasis on non-emotional 'None' samples for the robustness. [25]

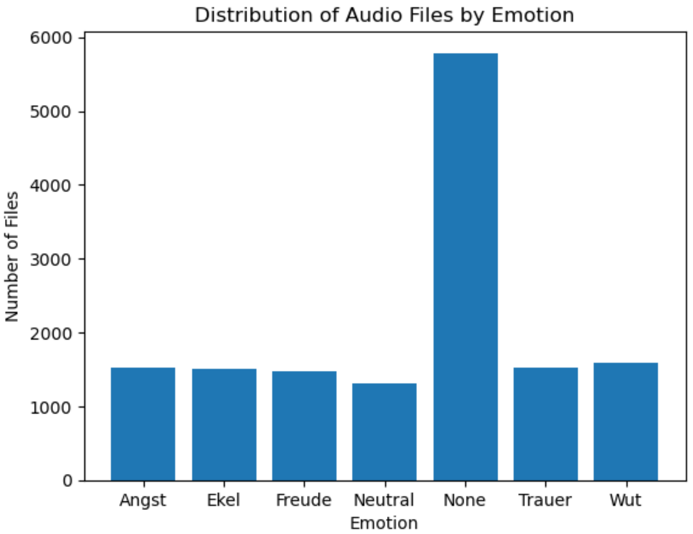


Figure 17 Distribution of Audio Files in Dataset by Emotion

3.5.2 Preprocessing Data

In the preprocessing of the model, as previously described in Chapter 2.10, we followed the following steps: normalizing the audio and then converting it into Mel coefficients using FFT and spectrogram. The exact implementation details are explained as follows:

The normalization of the audio was carried out as described in 2.10.1, ensuring that the amplitude of the audio is consistent throughout. After normalization, we generated a spectrogram using the STFT (Short-Time Fourier Transform) function from TensorFlow, which internally performs FFT. We then transformed it into a Mel spectrogram with frequencies ranging from 50 to 6000 Hz using the function "linear_to_mel_weight_matrix." Finally, we converted the Mel coefficient spectrogram into MFCCs (Mel Frequency Cepstral Coefficients) using the "mfccs_from_log_mel_spectrograms" function.

The decision to focus on the frequency range between 50 and 6000 Hz was strategic. While the most critical frequencies for speech are typically between 300 and 3500 Hz, we chose a broader range to capture the nuances in emotions, which are crucial for our analysis [26]. This wider frequency range provided us with a more comprehensive understanding of the emotional content in speech. Importantly, the best accuracy values in our project were achieved using this frequency range. The output format after preprocessing was 45x35, which proved to be an effective format due to its rectangular shape and favorable aspect ratio. Initially, we had an input of 128x13, which resulted in a highly irregular image that continued to shrink during max pooling.

The code for preprocessing can be found in the Appendix under sections 7.2.1.

3.5.3 Model Architecture and Development

We built our model using a 2D CNN as it's well-suited for processing Mel Frequency Cepstral Coefficient spectrograms, which are converted from audio preprocessing. Treating the spectrogram as an image, 2D CNNs are highly effective. [13]

Our model's architecture comprises the following layers, each tailored to process MFCC spectrogram data for emotion recognition:

1. **Reshape Layer:** Adjusts input shape to (45, 35, 1).
2. **First Conv2D Layer:** Extracts features using 32 filters, output shape (45, 35, 32).
3. **First MaxPooling2D Layer:** Reduces to (22, 17, 32).
4. **First Dropout Layer:** Prevents overfitting, maintains shape.
5. **Second Conv2D Layer:** Processes with 128 filters, output (22, 17, 128).
6. **Second MaxPooling2D Layer:** Further reduction to (11, 8, 128).
7. **Flatten Layer:** Converts to 1D vector (11264 units).
8. **First Dense Layer**: Integrates features, 128 units.
9. **Second Dropout Layer:** Further overfitting prevention.
10. **Second Dense Layer:** Final emotion classification, 7 units.

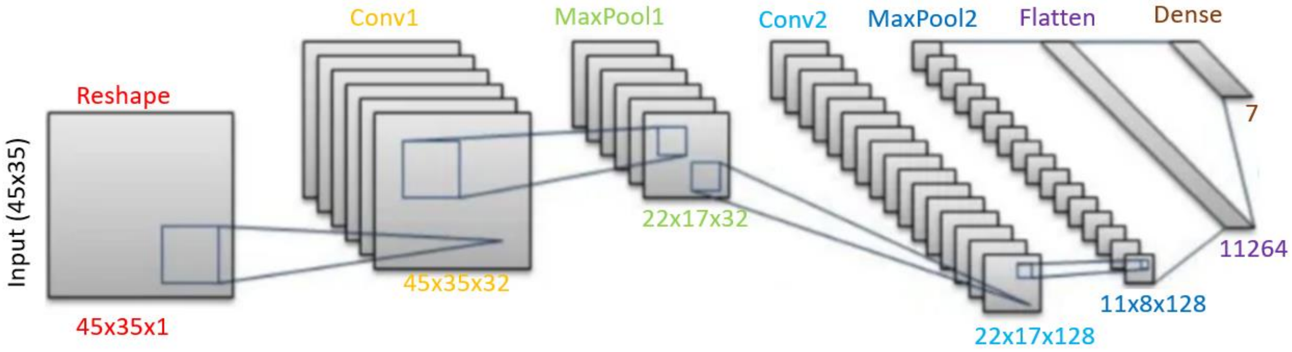


Figure 18 Model Architecture of the 2D CNN Model implemented.

The Python code and detailed model structure, along with the model summary, can be found in the appendix on chapter 7.2.1. This section provides comprehensive insights into the specifics of our neural network's architecture and the underlying code.

As described in Chapter 2.9.4, there are various activation functions to activate a node. We have tried different activations like ELU, Swish, LeakyReLU, and Mish. We achieved the best results with the ReLU activation function.

3.5.4 Training Process

Our model was trained in Supervised Mode using the dataset of 14,703 data points, which were split into training and validation sets. 20% of the data was used for validation to test the network's accuracy post-training.

The training was conducted on two different computers: one being a gaming PC with a GeForce GTX 3090, where training was directly performed on the GPU. This resulted in a training process duration of approximately 2 minutes for 15 epochs. On our Lenovo IdeaPad Flex 5, the model was trained using the CPU, as TensorFlow easily interfaces with Nvidia GPUs but is more complex with other GPUs. Running on the CPU, the training time for the same number of epochs (15) was approximately 12 minutes. This duration was still within an acceptable range, so for logistical reasons, we opted to train in this manner.

The model was trained using gradient descent with the Adam optimizer, which progressively adjusts the weights of the neurons to minimize loss. The Adam optimizer starts with a learning rate of 0.001 and adjusts accordingly. In addition to gradient descent, backpropagation was also employed, where the loss is propagated back from the output of the network through all layers to update the weights. This occurs in every training step, enabling the network to learn from the training data and improve. [13]

3.5.5 Optimization and Fine-Tuning

Before arriving at the final model, we tweaked several parameters to achieve the best results. Firstly, we experimented with different numbers of CNN layers, testing 2, 3, and 4 convolutional layers. We found that 2 CNN layers offered slightly better accuracy and less overfitting. Overfitting is identified when training accuracy significantly exceeds validation accuracy. Additionally, using 2 CNN layers saved us about 5 minutes per training session, enabling more efficient progress towards the optimal model. Alongside the number of layers, the number of neurons per layer was crucial. We ran a loop over the model with neuron sizes of 32, 64, 128, and 256 for both layers and measured accuracy. The best configuration turned out to be 32 neurons in the first layer and 128 in the second.

We also ran a loop to determine the optimal batch size, but we didn't observe significant differences in performance between batch sizes of 100 and 200. Therefore, we chose a batch size of 200 as it allowed for faster training of the model. Lastly, we attempted to adjust the learning rate, proceeding in a similar fashion with a loop of learning rates 0.1, 0.001, 0.001, and 0.0001. However, accuracy drastically decreased to 30% for all except with the Adam Optimizer at a learning rate of 0.001, so we retained this setting.

Initially, we set the dropout rate quite high at around 0.5, meaning 50% of the data was discarded at each iteration. This resulted in an accuracy of about 50%. Although 0.5 is a common value for dropout rates, it was not suitable for our needs. Therefore, we iterated with dropout rates between 0.1 and 0.4, finding that a dropout rate of 0.175 for the first dropout and 0.15 for the second dropout was most effective. Detailed values and the code for the model can be found, as mentioned earlier, in the appendix under sections 7.2.1. [27]

3.5.6 Solutions

In our solution, we focus on our main model, which has remained unchanged as described above. The preprocessing is done outside of the model, and the relevant code can be found in the appendix under sections 7.2.1. In this model, all six emotions as well as the 'None' data were used as input.

It is important to note that the 'None' data sets are about five times larger than those for the emotions, meaning the accuracy was artificially inflated as the model very reliably recognizes the 'None' data.

The accuracy of this model on the validation data is 75.21%, but the confusion matrix is more telling. It shows precisely how accurately each emotion was detected. In figure 19, you can see how the training and validation loss converge towards zero. [28]

Loss in this context refers to a measure of how well the model's predictions match the actual data. A lower loss indicates better performance, with the model's predictions being closer to the true values.

On the right side, you can see the accuracy of the training and validation data related to the epoch. It clearly shows that the validation data stagnates at around 75%, while the training data continues to increase linearly. Early stopping was employed to prevent overfitting, which is evident from the divergence of the training and validation accuracy over time.

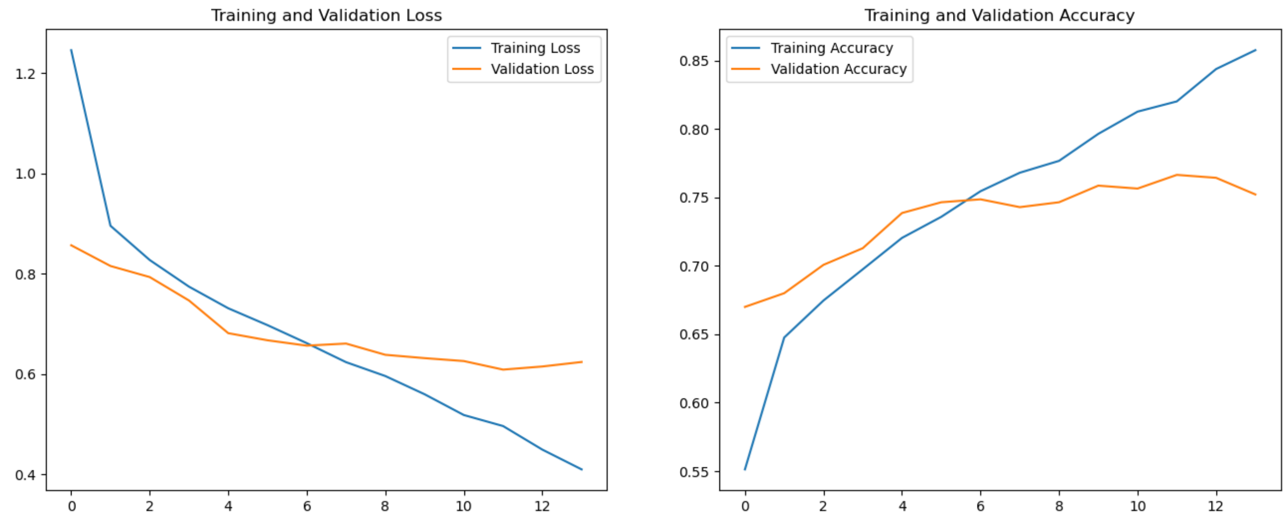


Figure 19 Graphs comparing Training and Validation Loss and Accuracy during model training over epochs.

The confusion matrix effectively highlights the strengths and weaknesses of the model. It has perfectly identified the 'None' data with 100% accuracy. One reason for this could be that the self-generated noise data used in the model follows a consistent pattern, which the model has learned to recognize precisely. Additionally, other 'None' data from the WHAM dataset, which includes background noises from environments such as subways and libraries, are accurately identified because these recordings do not contain active speech. This distinction makes it easier for the model to detect them, as the resulting Mel spectrogram would look significantly different from those of recordings that include speech.

Furthermore, the model is particularly adept at recognizing the emotion 'Anger,' achieving over 75% accuracy. We assume that 'Anger' can be more easily detected due to its association with high intensity, higher pitch, and faster speech rate.

The most crucial aspect of the confusion matrix is that the diagonal, which represents correct classifications, is the most dominant. This indicates that, except for 'Fear', each emotion is identified correctly more than 50% of the time, ensuring that the model can accurately detect each emotion at least every other attempt.

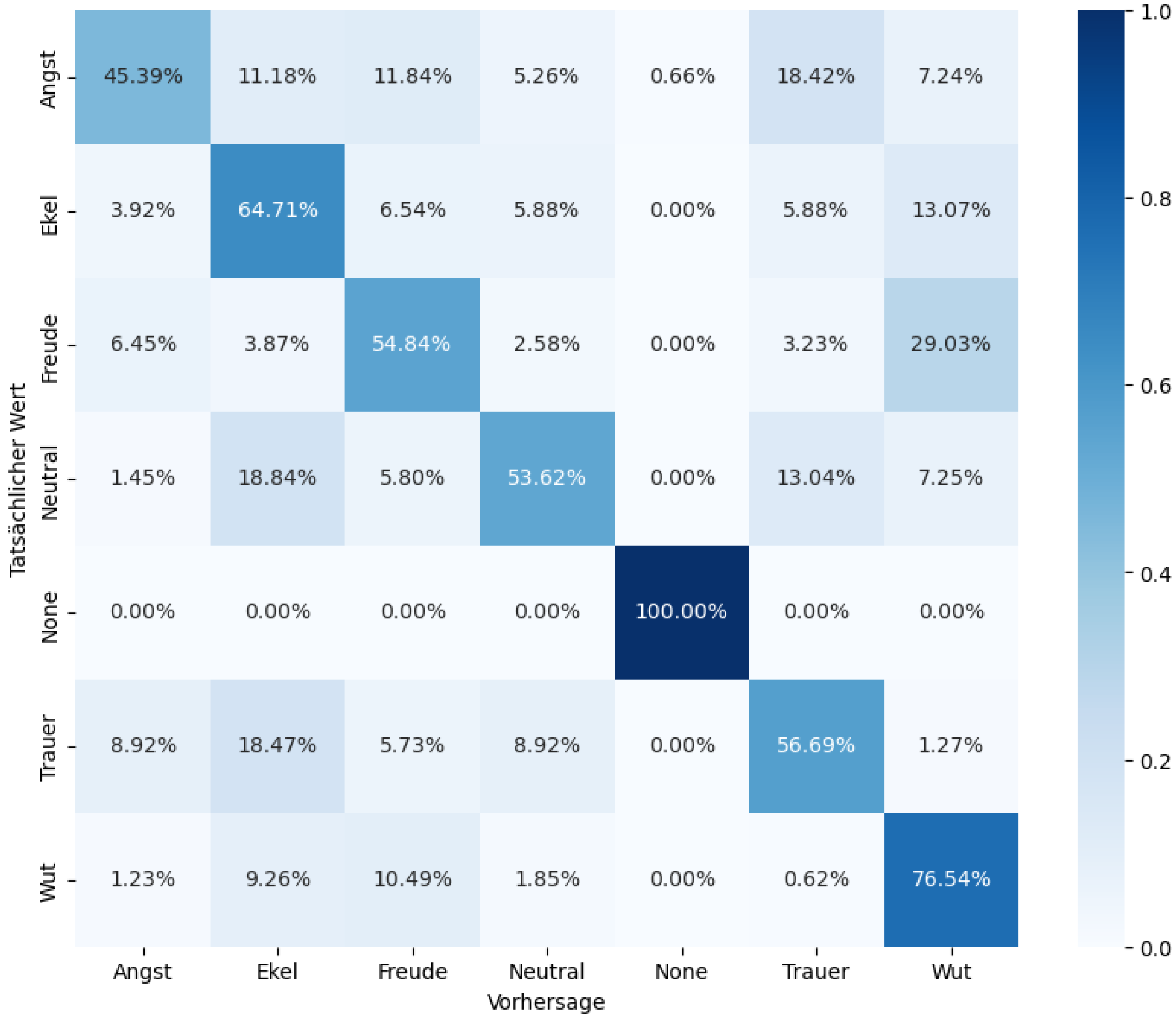


Figure 20 Confusion Matrix of 2D CNN Model with preprocessing outside of the model

3.5.7 Human-Labeled Emotion Accuracy

To contextualize the confusion matrix and the accuracy of our model, we conducted a small survey. This was not a scientifically rigorous survey but was intended to get a sense of how well humans can assign emotions to our training data. For this, six audio files, each representing a different emotion, were randomly selected, and ten participants were tasked with assigning the correct emotion to each file. These are the results:

Angst	60%	0%	0%	10%	0%	30%
Ekel	10%	30%	0%	30%	30%	0%
Freude	30%	0%	70%	0%	0%	0%
Neutral	0%	40%	0%	60%	0%	0%
Trauer	0%	0%	0%	0%	100%	0%
Wut	30%	0%	30%	0%	0%	40%
	Angst	Ekel	Freude	Neutral	Trauer	Wut

A correlation between the model and the data labeled by humans could not be discerned, however, it should be noted that the sample size of six audio files and ten participants is quite small. The overall accuracy of the human labeling was 57%. If one were to draw a comparison, our trained neural network is significantly more accurate.

Although not definitive, it shows that even for us humans, it is not always clear which emotion is being expressed, especially when it is through audio alone. Humans naturally also rely on facial expressions, gestures, and other cues.

3.6 Integrate TensorFlow Lite Model

In the project, TensorFlow Lite models were successfully integrated into an Xcode environment using CocoaPods. Initially, the TensorFlow model was thoroughly trained, tested, and converted into the TensorFlow Lite format, optimizing it for mobile device performance. In the converting process, the challenge was to find TensorFlow functions that are convertible to a TensorFlow Lite model. Initially, we encountered an issue when converting the model to TfLite. However, this was due to the combination of TensorFlow and Python versions not working together. Specifically, with Python version 3.12.0 and TensorFlow version 2.15.0, it worked successfully.

The integration process began with the setup of CocoaPods in the Xcode project. Once the Podfile was updated, the command `pod install` was run in the terminal within the project directory. This action prompted CocoaPods to fetch and install the TensorFlow Lite library, integrating it into the Xcode project. It was ensured that the `.xcworkspace` file was used thereafter, as it contained both the original project and the newly integrated CocoaPods dependencies.

The last step involved converting the trained TensorFlow model into TensorFlow Lite. The challenge encountered was that preprocessing functions, such as MFCC calculations, are not natively available in Swift. Calculating the coefficients manually would have been extremely time-consuming and prone to errors. Instead, preprocessing was handled within TensorFlow, and this step was incorporated as the first "input" layer within the model, thus being trained with it. The associated code can be found in Chapter 7.2.3, and the model structure running on the iOS app is detailed in Chapter 7.3.1.

The final stage of integration entailed embedding the TensorFlow Lite model within the iOS Player app. This included recording voice samples, saving them within the app, loading the model, preparing the input data as per the model's requirements, and running the model to generate predictions or results regarding emotions.

3.7 ChatGPT

In our project, we also set out to evaluate the utility of ChatGPT, specifically ChatGPT 4, in supporting our objectives. This investigation aimed to understand the strengths and limitations of ChatGPT in a practical development context.

ChatGPT proved to be an invaluable asset, particularly in time-saving aspects. It effectively reduced the hours that would typically be spent scouring through documentation or browsing through Stack Overflow posts. Often, the solutions provided by ChatGPT were successful on the first attempt, although there were instances where they did not work as expected. In such cases, it was crucial to pivot and suggest an alternative approach, which ChatGPT could then explore further. This highlighted the importance of precision in task formulation; the more specific the instructions given to ChatGPT, the more accurate and useful the provided solutions were. Also, in terms of “glowing up” written text, GTP4 is a very useful tool. [29]

However, it was also observed that ChatGPT has its limitations, especially when dealing with novel or highly specialized fields, such as those we ventured into in this project. In these scenarios, while ChatGPT could offer general guidance and support, its responses were sometimes less effective due to the cutting-edge nature of the work and the lack of extensive data in such new domains.

Following instructions helped us get the best answer from ChatGPT:

- Being as precise as possible with your instructions
- Don’t be shy to feed ChatGPT whole blocks of code to analyze and improve.
- There are moments where ChatGPT can’t seem to find a new approach, then it is your turn to show him an alternate way so he can work from there with your idea.

On the next page is an example prompt shown, how it was often encountered.

V **You**

Please build me a class in swift/xcode to record an audio file and save it in the documents:

i found this as base:

```
class AudioRecorder: ObservableObject {
  var audioRecorder: AVAudioRecorder?
  @Published var isRecording = false

  func startRecording() {
    let session = AVAudioSession.sharedInstance()
    do {
      try session.setCategory(.playAndRecord)
      try session.setActive(true)

      let url = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
[0]
      let filename = "myRecording_\(Date().timeIntervalSince1970).m4a" // Unique
filename
      let fileURL = url.appendingPathComponent(filename)

      let settings = [
        AVFormatIDKey: Int(kAudioFormatAppleLossless), // Audio Format
        AVSampleRateKey: 44100,          // Sample Rate in Hz
        AVEncoderBitRateKey: 320000,
        AVNumberOfChannelsKey: 2,          // Number of Channels (2 for stereo)
        AVEncoderAudioQualityKey: AVAudioQuality.high.rawValue // Audio Quality
      ]

      audioRecorder = try AVAudioRecorder(url: fileURL, settings: settings)
      audioRecorder?.record()
      isRecording = true
    } catch {
      print("Audio Recorder setup failed: \(error)")
    }
  }

  func stopRecording() {
    audioRecorder?.stop()
    isRecording = false
    if let fileURL = audioRecorder?.url {
      print("Recording stopped. File saved to: \(fileURL.path)")
    }
  }
}
```

ChatGPT

Your base class `AudioRecorder` is a good start for creating a Swift class to record audio and save it in the documents directory using Xcode. To make it fully functional, a few enhancements and checks can be added. Here's an improved version of your class:

swift Copy code

```swift
import Foundation
import AVFoundation

class AudioRecorder: ObservableObject {
    var audioRecorder: AVAudioRecorder?
    @Published var isRecording = false

    func startRecording() {
        let session = AVAudioSession.sharedInstance()
        do {
            try session.setCategory(.playAndRecord, mode: .default)
            try session.setActive(true)

            let url = FileManager.default.urls(for: .documentDirectory,
            let filename = "myRecording_\(Date().timeIntervalSince1970)
            let fileURL = url.appendingPathComponent(filename)

            let settings = [
                AVFormatIDKey: Int(kAudioFormatMPEG4AAC), // Changed to
                AVSampleRateKey: 44100,                    // Sample Rat
                AVEncoderBitRateKey: 320000,
                AVNumberOfChannelsKey: 2,                  // Number of
                AVEncoderAudioQualityKey: AVAudioQuality.high.rawValue
            ]

            audioRecorder = try AVAudioRecorder(url: fileURL, settings:
            audioRecorder?.record()
            isRecording = true
        } catch {
            print("Audio Recorder setup failed: \(error)")
        }
    }

    func stopRecording() {
        audioRecorder?.stop()
        isRecording = false
        if let fileURL = audioRecorder?.url {
            print("Recording stopped. File saved to: \(fileURL.path)")
        }
    }
}
```

Key Changes and Enhancements:

1. **Audio Format Change**: Changed the audio format to `kAudioFormatMPEG4AAC` for better compatibility across various devices and platforms.
2. **Session Mode Setting**: Added a mode setting to the `AVAudioSession` configuration for more explicit session management.
3. **Error Handling**: The existing error handlir ints the error in case of failure, which is good for debugging.

Message ChatGPT...

The Question was if it was possible to determine at which simple the player is exactly at any moment. Even though ChatGPT answers with Yes, the answer is still just an estimation as GPT states in second last paragraph.

ChatGPT

Yes, in iOS development using the AVFoundation library, you can determine which sample is currently being played in various ways, depending on the context and the specific AVFoundation class you are using.

For instance, if you are using `AVAudioPlayer`, you can access the `currentTime` property to get the current playback time in seconds. This doesn't directly give you the sample number but allows you to calculate it based on the sample rate. Here's a basic way to estimate the current sample:

1. Get the current playback time in seconds from `currentTime`.
2. Get the audio file's sample rate (usually in samples per second) from the audio file format.
3. Multiply the playback time by the sample rate to get the approximate current sample number.

Here's a simple example in Swift:

swift Copy code

```swift
let currentTimeInSeconds = audioPlayer.currentTime
let sampleRate = audioFile.processingFormat.sampleRate
let currentSample = Int(currentTimeInSeconds * sampleRate)
```

In this code:

- `audioPlayer` is an instance of `AVAudioPlayer`.
- `audioFile` is an instance of `AVAudioFile` or similar, used to get the sample rate of the audio.

Remember, this is an approximation and may not be perfectly accurate for every frame, especially in compressed formats or when dealing with complex audio processing.

If you are using a different class in AVFoundation, such as `AVPlayer` or lower-level APIs like `AVAudioEngine`, the approach might differ slightly. These classes might provide different methods or properties to access the current playback time or the current audio frame.

[29]

4 Results

4.1 Audio Player

Throughout this project, several objectives were achieved. A key milestone was the ability to manipulate audio samples by dividing an audio file into smaller segments and applying real-time effects. This became a crucial component of the final application. However, we faced challenges in accurately determining the exact sample being played at any given time. This led to the development of a method to estimate the player's position, resulting in an approximation of time rather than precise tracking.

4.2 Neural Network

The results of our Neural Network demonstrate that we can successfully recognize emotions using our CNN model with an accuracy of more than 75% on test data. It's worth noting that even humans performed worse with these training data in this small survey. In the preprocessing of the model, we followed a series of steps, including normalizing the audio, converting it into Mel coefficients, and generating a Mel coefficient spectrogram. The model's architecture comprises several layers, each designed to process MFCC spectrogram data for emotion recognition. The confusion matrix reveals that our model excels in identifying 'None' data and the emotion 'Anger.' Finally, we successfully integrated the TensorFlow Lite model into the Xcode environment for use in a mobile application, enabling mobile audio emotion recognition. This model, along with its effective confusion matrix as shown in figure 21, has now been successfully integrated into our iOS app, making it accessible for emotion recognition on mobile devices.

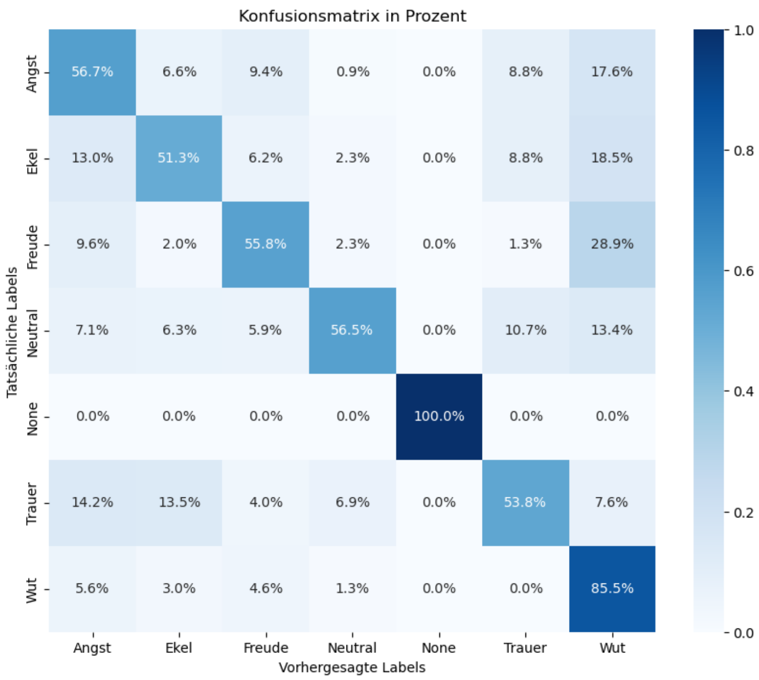


Figure 21 Confusion Matrix of TensorFlow Lite Model

4.3 iOS Application

The application has a simple design and is easy to use. It allows users to record one or more memos and offers playback functionality, with the option to apply various audio effects. The key feature is the integration of a neural network for emotion recognition. By selecting the 'tell me how I feel' option, users can submit their active memo for analysis. The system processes the memo and identifies potential emotions, categorized into seven distinct types. The algorithm then determines the predominant emotion and represents it with a corresponding Emoji. This output, along with a textual notification stating, 'I sense you feel <emotion>', is displayed on the user's screen, providing an intuitive interface for emotional feedback, as shown in figure 22.

<u>Initial state:</u>

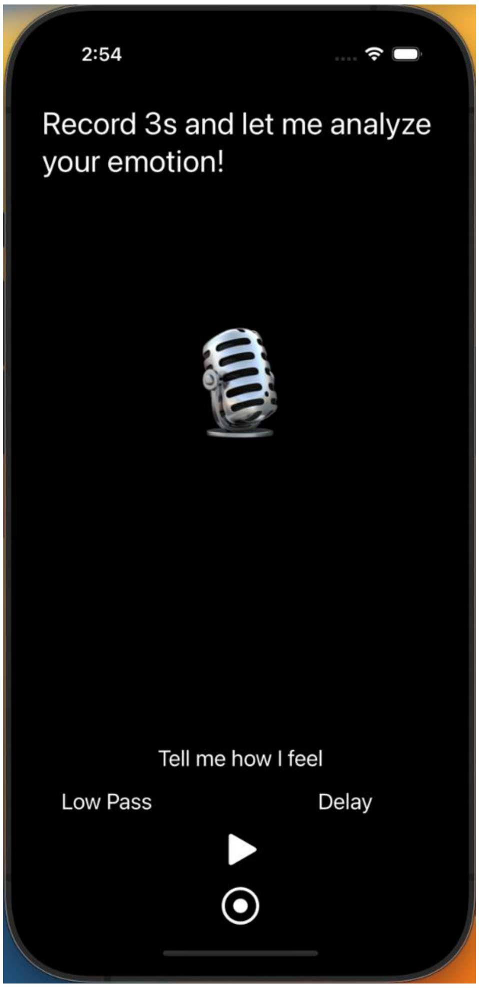


Figure 23 iOS UI Initial state

<u>Memo recorded:</u>

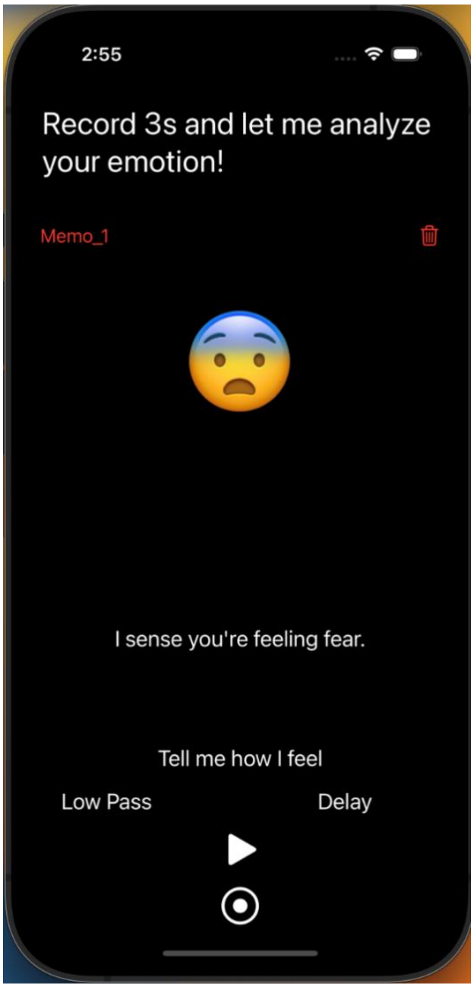


Figure 22 iOS UI Memo with fear recorded

5 Discussion

Taking on the challenge of incorporating real-time audio manipulation into an iOS app, our project pushed the limits of what’s possible with the AVFoundation library. It became clear that to achieve the level of precision we wanted for on-the-fly audio adjustments, we had to dig deeper into low-level programming.

Our decision to analyze a wider range of frequencies let us pick up on the finer details of emotional expression in speech, suggesting a promising direction for emotion detection technology. Such advancements could lead to more responsive and emotionally aware applications.

It was interesting to see our model sometimes do better than people in recognizing emotions, showing that technology might one day assist in areas like mental health support.

By bringing this technology into a mobile app, we're moving closer to having our devices understand and react to our emotions, making our interactions with them more natural and personal.

Our work also shows that with platforms like TensorFlow Lite, it's possible to run complex machine learning models on everyday devices, potentially making powerful computing tools widely available.

Looking forward, we're excited about the prospect of creating models that not only understand our current emotions but can also track and adapt to our emotional patterns over time. This could open up new possibilities for monitoring and supporting emotional health.

6 Outlook

Looking forward there are several promising avenues to explore for our Emotion Recognition project:

Firstly, Data Enrichment is a priority. Gathering more real-world data would be beneficial to improve our model's accuracy. Additional data would enable us to train the model more effectively and enhance its performance across various scenarios.

Next, we should explore Neural Network Techniques in more detail. Experimenting with parameters like stride length or batch normalization can help us refine the model and make it more efficient.

One exciting prospect is Real Time Emotion Recognition. Modifying our Swift app to perform real-time analysis during live conversations without the need to prerecord audio could significantly enhance user experience.

Incorporating Speech Recognition capabilities is another avenue to consider. This addition would allow our application to comprehend spoken language, adding depth to our emotion analysis. Furthermore, the integration of Language Models like ChatGPT holds potential. These models could provide valuable insights into a person's emotional state based on text or speech.

Collaborating with experts in psychology and emotion recognition for research collaborations could validate and enhance our model's accuracy.

Lastly, exploring opportunities in various industries, including mental health, customer service, and entertainment, could broaden the application of our technology.

By pursuing these possibilities, we could further enhance and diversify our Emotion Recognition project, making it an even more valuable tool with a broader range of applications.

References

[1] "Convolutional neural networks pretrained on large face recognition datasets for emotion classification from video," ar5iv. Accessed: Dec. 18, 2023. [Online]. Available: https://ar5iv.labs.arxiv.org/html/1711.04598

[2] Y. Zhou, W. Han, X. Yao, J. Xue, Z. Li, and Y. Li, "Developing a machine learning model for detecting depression, anxiety, and apathy in older adults with mild cognitive impairment using speech and facial expressions: A cross-sectional observational study," *Int. J. Nurs. Stud.*, vol. 146, p. 104562, Oct. 2023, doi: 10.1016/j.ijnurstu.2023.104562.

[3] "Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN." Accessed: Dec. 18, 2023. [Online]. Available: https://www.hindawi.com/journals/cin/2020/7251280/

[4] Colin, "Audio Recording 101 for Home Studio Musicians," E-Home Recording Studio. Accessed: Dec. 18, 2023. [Online]. Available: https://ehomerecordingstudio.com/audio-recording/

[5] "Basic Audio Recording Techniques | The Definitive Guide 2023." Accessed: Dec. 18, 2023. [Online]. Available: https://mixingmonster.com/audio-recording-techniques/

[6] "From Source to Output: Audio Signal Flow for Videographers - Videomaker." Accessed: Dec. 20, 2023. [Online]. Available: https://www.videomaker.com/what-is-signal-flow/

[7] "Waves, Spectra and Sounds | Inner Magazines." Accessed: Dec. 20, 2023. [Online]. Available: https://www.inner-magazines.com/audiophilia/waves-spectra-and-sounds-fourier-analysis/

[8] A. Inc, "Xcode 15," Apple Developer. Accessed: Dec. 18, 2023. [Online]. Available: https://developer.apple.com/xcode/

[9] A. Inc, "Swift - Apple Developer." Accessed: Dec. 18, 2023. [Online]. Available: https://developer.apple.com/swift/

[10] A. Inc, "AVFoundation," Apple Developer. Accessed: Dec. 18, 2023. [Online]. Available: https://developer.apple.com/av-foundation/

[11] "TensorFlow." Accessed: Dec. 18, 2023. [Online]. Available: https://www.tensorflow.org/

[12] "What's coming in TensorFlow 2.0." Accessed: Dec. 20, 2023. [Online]. Available: https://blog.tensorflow.org/2019/01/whats-coming-in-tensorflow-2-0.html

[13] C. Bishop, "Pattern Recognition and Machine Learning," Jan. 2006, Accessed: Dec. 18, 2023. [Online]. Available: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/

[14] 2023 ZHAW, "F. Wadehn, „DNN2. unveröffentlichte Vorlesung der Lektion 3 an der ZHAW School of Engineering", am 21.12.2023. Winterthur."

[15] 2023 ZHAW, "F. Wadehn, „DNN2. unveröffentlichte Vorlesung der Lektion 2 an der ZHAW School of Engineering", am 21.12.2023. Winterthur."

[16] "What are Recurrent Neural Networks? | IBM." Accessed: Dec. 18, 2023. [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks

[17] "– Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks," ar5iv. Accessed: Dec. 18, 2023. [Online]. Available: https://ar5iv.labs.arxiv.org/html/1909.09586

[18] S. G. Koolagudi, D. Rastogi, and K. S. Rao, "Identification of Language using Mel-Frequency Cepstral Coefficients (MFCC)," *Procedia Eng.*, vol. 38, pp. 3391–3398, Jan. 2012, doi: 10.1016/j.proeng.2012.06.392.

[19] Z. M. Chng, “Using Normalization Layers to Improve Deep Learning Models,” MachineLearningMastery.com. Accessed: Dec. 18, 2023. [Online]. Available: https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models/

[20] “Mel Frequency Cepstral Coefficient and its Applications: A Review | IEEE Journals & Magazine | IEEE Xplore.” Accessed: Dec. 18, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9955539

[21] “Planet Of Tunes - Time domain effects.” Accessed: Dec. 20, 2023. [Online]. Available: https://planetoftunes.com/sound-recording/time-domain-effects.php

[22] “MATLAB.” Accessed: Dec. 18, 2023. [Online]. Available: https://www.mathworks.com/products/matlab.html

[23] “Butterworth Filter: What is it? (Design & Applications) | Electrical4U.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.electrical4u.com/butterworth-filter/

[24] “CREMA-D.” Accessed: Dec. 18, 2023. [Online]. Available: https://www.kaggle.com/datasets/ejlok1/cremad

[25] “RAVDESS Emotional speech audio.” Accessed: Dec. 18, 2023. [Online]. Available: https://www.kaggle.com/datasets/uwrfkaggler/ravdess-emotional-speech-audio

[26] “Waveform - Introduction to Speech Processing - Aalto University Wiki.” Accessed: Dec. 21, 2023. [Online]. Available: https://wiki.aalto.fi/display/ITSP/Waveform

[27] J. Brownlee, “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks,” MachineLearningMastery.com. Accessed: Dec. 20, 2023. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[28] baeldung, “Training and Validation Loss in Deep Learning | Baeldung on Computer Science.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.baeldung.com/cs/training-validation-loss-deep-learning

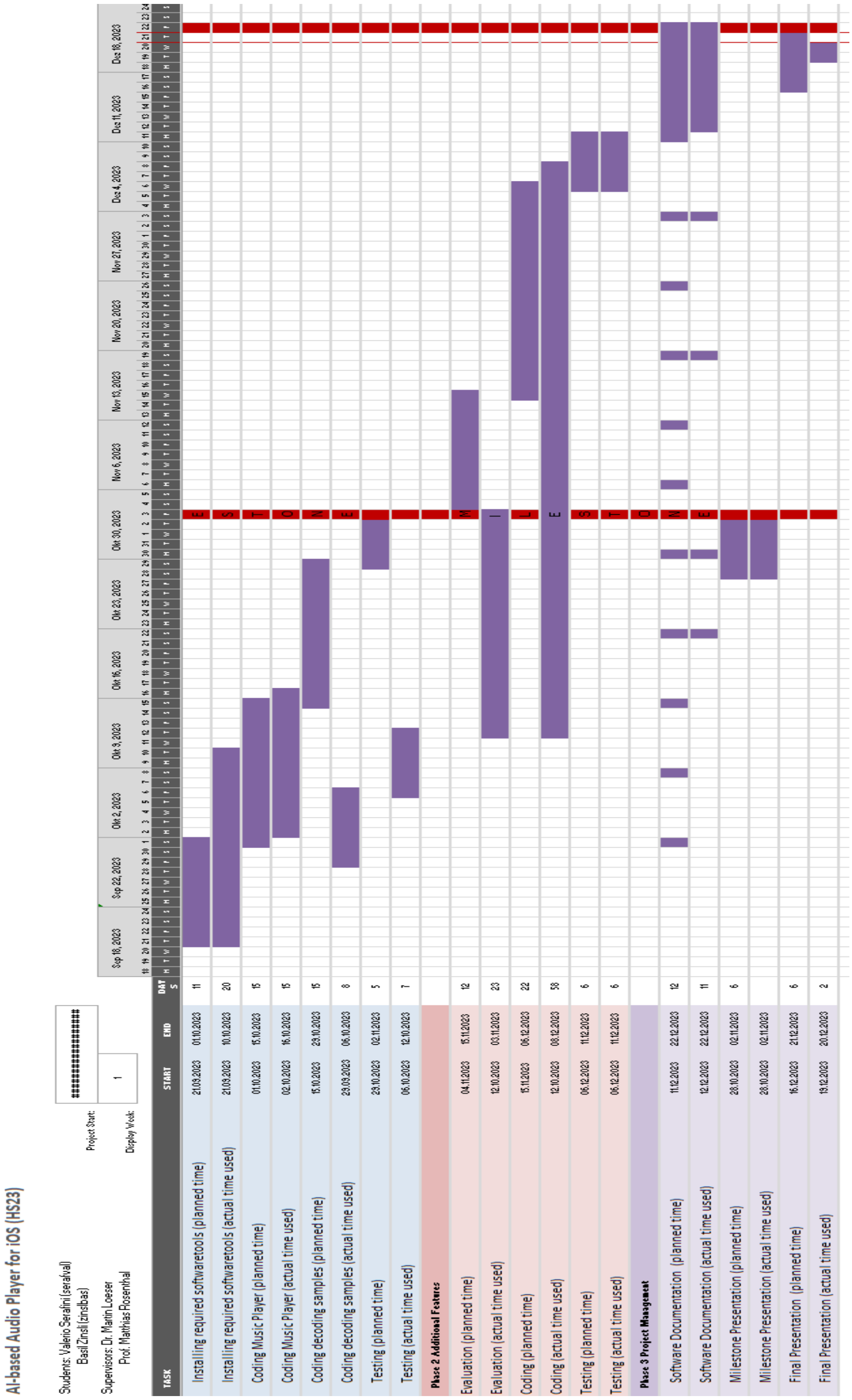
[29] “ChatGPT.” Accessed: Dec. 18, 2023. [Online]. Available: https://chat.openai.com

ChatGPT was used for proofreading the document.

List of Figures

7 Appendix

7.1 Project management



7.2 Python Model Code

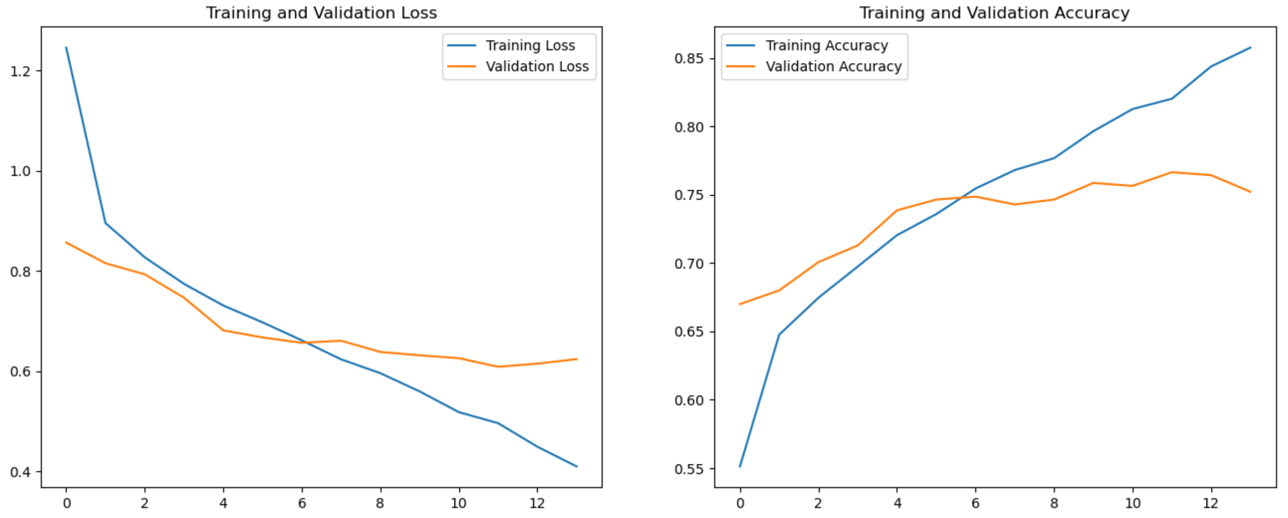
7.2.1 Preprocessing Outside of the Model

```
import os
import pathlib
import numpy as np
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers, models

# Pfad zur Datensammlung
DATASET_PATH = "C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\TensorflowModellTest\\PA
_Emotion_Modell\\Trainings und Testdaten"
data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
    raise Exception("Datenverzeichnis existiert nicht!")

# Laden der Daten
commands = np.array(['Angst', 'Ekel', 'Freude', 'Neutral', 'Trauer', 'Wut', 'None'])
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=200,
    validation_split=0.2,
    seed=42,
    output_sequence_length=48000,
    subset='both')

label_names = np.array(train_ds.class_names)
# Daten vorbereiten
def squeeze(audio, labels):
    return tf.squeeze(audio, axis=-1), labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)

# Funktion zur Normalisierung des Audios
def normalize_audio(audio):
    mean = tf.reduce_mean(audio)
    std_dev = tf.math.reduce_std(audio)
    return (audio - mean) / std_dev

# Funktion zur Berechnung der Mel-Spektrogramme
def get_mel_spectrogram(waveform):
    spectrogram = tf.signal.stft(waveform, frame_length=2048, frame_step=1024)
```

```
    spectrogram = tf.abs(spectrogram)
    num_spectrogram_bins = spectrogram.shape[-1]
    linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
        512, num_spectrogram_bins, 16000, 50.0, 6000.0)
    mel_spectrogram = tf.tensordot(spectrogram, linear_to_mel_weight_matrix, 1)
    mel_spectrogram.set_shape(spectrogram.shape[:-1].concatenate(linear_to_mel_weight_matrix.
shape[-1:]))
    return mel_spectrogram

# Funktion zur Berechnung der MFCCs
def get_mfccs(mel_spectrogram):
    log_mel_spectrogram = tf.math.log1p(mel_spectrogram)
    mfccs = tf.signal.mfccs_from_log_mel_spectrograms(log_mel_spectrogram)[..., :35]
    return mfccs

# Anwenden der Vorverarbeitung und Normalisierung auf die Datasets
def preprocess_dataset(ds):
    ds = ds.map(lambda audio, label: (normalize_audio(audio), label), num_parallel_calls=tf.data.AU
TOTUNE)
    return ds.map(lambda audio, label: (get_mfccs(get_mel_spectrogram(audio)), label), num_parall
el_calls=tf.data.AUTOTUNE)

train_ds = preprocess_dataset(train_ds)
val_ds = preprocess_dataset(val_ds)
test_ds = preprocess_dataset(test_ds)

# Modell bauen
def build_model(input_shape, num_labels):
    model = models.Sequential([
        layers.Input(shape=input_shape),
        layers.Reshape((45, 35, 1)),
        layers.Conv2D(32, (5, 5), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.175),
        layers.Conv2D(128, (5, 5), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='sigmoid'),
        layers.Dropout(0.15),
        layers.Dense(num_labels, activation='softmax'),
    ])

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

```
    return model

# Festlegen der Input-Shape und der Anzahl der Klassen
input_shape = (45, 35)  # Zeitframes und Anzahl der MFCC-Koeffizienten
num_labels = len(commands)  # Anzahl der Klassifikationskategorien

# Erstellen des Modells
model = build_model(input_shape, num_labels)

# Training des Modells
EPOCHS = 30  # Anzahl der Epochen
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2, verbose=1)],
    verbose=1
)

# Speichern des Modells im SavedModel-Format (.keras)
model.save('model_saved_model')

# Laden des SavedModel-Formats
loaded_model = tf.keras.models.load_model('model_saved_model')

# Konvertieren des geladenen Modells in TensorFlow Lite Format
converter = tf.lite.TFLiteConverter.from_keras_model(loaded_model)
converter.optimizations = []
tflite_model = converter.convert()

# Speichern des TFLite-Modells in einer Datei
tflite_model_path = "C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\TensorflowModellTest\\PA_
Emotion_Modell\\TFLite\\model.tflite"
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)

# Visualisierung des Trainingsverlaufs
import matplotlib.pyplot as plt

plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plt.plot(history.epoch, history.history['loss'], label='Training Loss')
plt.plot(history.epoch, history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history.epoch, history.history['accuracy'], label='Training Accuracy')
plt.plot(history.epoch, history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

# Modell evaluieren
y_pred = model.predict(test_ds)
y_pred = tf.argmax(y_pred, axis=1)
y_true = tf.concat([y for x, y in test_ds], axis=0)

confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
confusion_mtx = tf.cast(confusion_mtx, dtype=tf.float32)  # Konvertieren zu float für Division
sum_by_row = tf.reduce_sum(confusion_mtx, axis=1)  # Summe jeder Zeile
confusion_mtx_percentage = tf.divide(confusion_mtx, tf.reshape(sum_by_row, [-1, 1]))  # Normalisi
eren

plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx_percentage,
            xticklabels=label_names,
            yticklabels=label_names,
            annot=True, fmt='.2%', cmap='Blues')  # Formatierung als Prozent
plt.xlabel('Vorhersage')
plt.ylabel('Tatsächlicher Wert')
plt.show()loss: 0.4095 - accuracy: 0.8575 - val_loss: 0.6236 - val_accuracy: 0
.7521
```



7.2.2 Preprocessing in Model

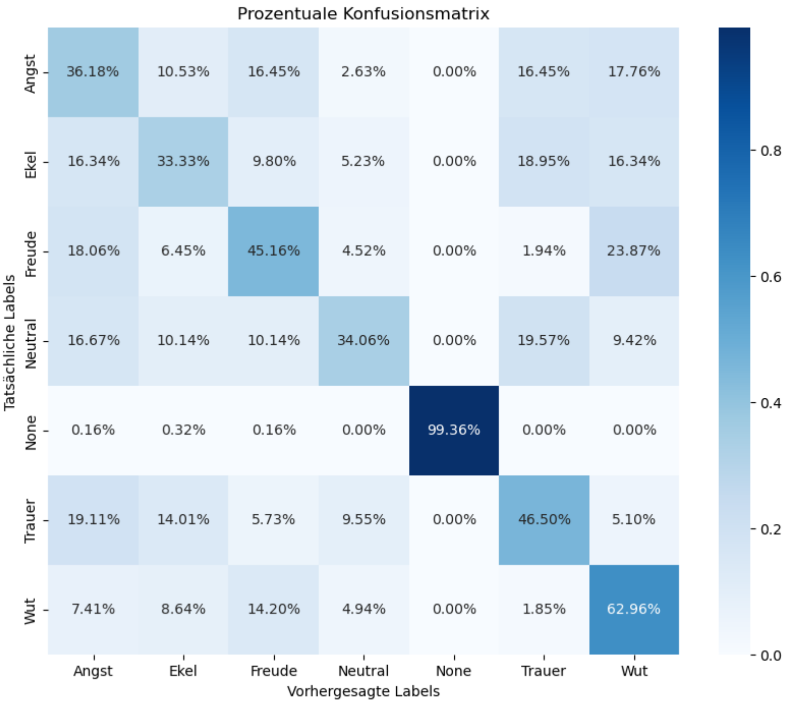
```
import os
import pathlib
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models

# Pfad zur Datensammlung
DATASET_PATH = "C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\TensorflowModellTest\\PA
_Emotion_Modell\\Trainings und Testdaten"
data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
    raise Exception("Datenverzeichnis existiert nicht!")

# Laden der Daten
commands = np.array(['Angst', 'Ekel', 'Freude', 'Neutral', 'None', 'Trauer', 'Wut'])
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=200,
    validation_split=0.2,
    seed=42,
    output_sequence_length=48000,
    subset='both')

# Daten vorbereiten
train_ds = train_ds.map(lambda audio, label: (tf.squeeze(audio, axis=-1), label))
val_ds = val_ds.map(lambda audio, label: (tf.squeeze(audio, axis=-1), label))
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

```
# Mel-Spektrogramm und MFCCs direkt im Modell berechnen
class AudioPreprocessingLayer(layers.Layer):
    def __init__(self):
        super(AudioPreprocessingLayer, self).__init__()

    def call(self, waveform):
        # Normalisierung
        mean = tf.reduce_mean(waveform)
        std_dev = tf.math.reduce_std(waveform)
        normalized_waveform = (waveform - mean) / std_dev

        # Mel-Spektrogramm
        spectrogram = tf.signal.stft(normalized_waveform, frame_length=2048, frame_step=1024)
        spectrogram = tf.abs(spectrogram)
        num_spectrogram_bins = spectrogram.shape[-1]
        linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
            512, num_spectrogram_bins, 16000, 50.0, 6000.0)
        mel_spectrogram = tf.tensordot(spectrogram, linear_to_mel_weight_matrix, 1)
        mel_spectrogram.set_shape(spectrogram.shape[:-1].concatenate(linear_to_mel_weight_matri
x.shape[-1:]))

        # MFCCs
        log_mel_spectrogram = tf.math.log1p(mel_spectrogram)
        mfccs = tf.signal.mfccs_from_log_mel_spectrograms(log_mel_spectrogram)[..., :35]
        mfccs = spectrogram[..., :35]
        return mfccs

# Modell bauen
def build_model(input_shape, num_labels):
    model = models.Sequential([
        layers.Input(shape=input_shape),
        AudioPreprocessingLayer(),
        layers.Reshape((45, 35, 1)),
        layers.Conv2D(32, (5, 5), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.175),
        layers.Conv2D(128, (5, 5), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='sigmoid'),
        layers.Dropout(0.15),
        layers.Dense(num_labels, activation='softmax'),
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
```

```
                  metrics=['accuracy'])
    return model

input_shape = (48000, )  # ursprüngliche Wellenformlänge
num_labels = len(commands)

model = build_model(input_shape, num_labels)

# Training des Modells
EPOCHS = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2, verbose=1)],
    verbose=1
)

# Vorhersagen auf dem Testdatensatz
y_pred = model.predict(test_ds)
predicted_classes = tf.argmax(y_pred, axis=1)

# Klassenindizes in Klassennamen umwandeln
class_names = commands  # oder train_ds.class_names, wenn es verfügbar ist

# Tatsächliche Labelnamen
true_label_names = np.array(class_names)[true_labels]

# Vorhergesagte Labelnamen
predicted_label_names = np.array(class_names)[predicted_classes]

# Konfusionsmatrix mit Klassenindizes erstellen
confusion_mtx = tf.math.confusion_matrix(true_labels, predicted_classes)

# Konfusionsmatrix in Prozent umwandeln
confusion_mtx_percentage = confusion_mtx / tf.reduce_sum(confusion_mtx, axis=1)[:, tf.newaxis]

# Konfusionsmatrix visualisieren mit Klassennamen
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx_percentage, annot=True, fmt='.2%', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Vorhergesagte Labels')
plt.ylabel('Tatsächliche Labels')
plt.title('Prozentuale Konfusionsmatrix')
plt.show()
```

```
# Konvertieren in TFLite-Modell
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Speichern des TFLite-Modells
tflite_model_path = "C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\TensorflowModellTest\\PA_
Emotion_Modell\\TFLite\\model.tflite"
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)
```

```
loss: 0.6793 - accuracy: 0.7482 - val_loss: 0.8628 - val_accuracy: 0.6686
```



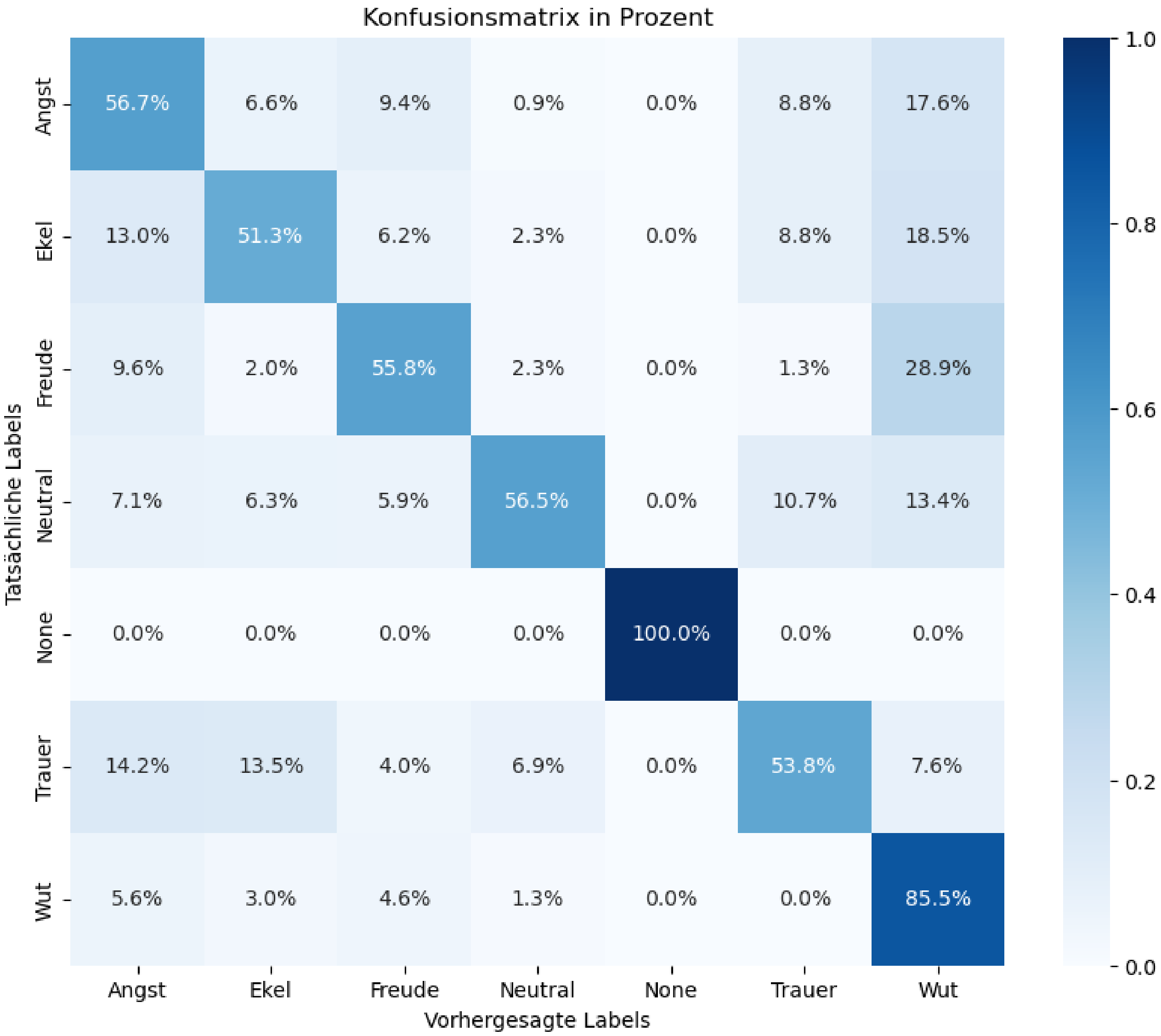
7.2.3 Testing TensorFlow Lite Model in Python

```
import os

import numpy as np

import tensorflow as tf

from sklearn.model_selection import train_test_split


# Pfad zur Datensammlung

DATASET_PATH = "C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\TensorflowModell-
Test\\PA_Emotion_Modell\\Trainings und Testdaten"

LABELS = ['Angst', 'Ekel', 'Freude', 'Neutral', 'None', 'Trauer', 'Wut']
```

```
def load_data(dataset_path, sr=16000, duration=3):
    all_waveforms = []
    all_labels = []
    max_length = sr * duration  # Maximale Länge der Waveform in Samples
    for label in LABELS:
        label_dir = os.path.join(dataset_path, label)
        for wav_file in os.listdir(label_dir):
            if wav_file.lower().endswith('.wav'):
                file_path = os.path.join(label_dir, wav_file)
                waveform, _ = tf.audio.decode_wav(tf.io.read_file(file_path))
                waveform = tf.squeeze(waveform, axis=-1)

                # Wellenform auf maximale Länge kürzen oder mit Nullen auffüllen
                waveform_length = tf.shape(waveform)[0]
                padding = tf.maximum(max_length - waveform_length, 0)
                zero_padding = tf.zeros([padding], dtype=tf.float32)
                waveform = tf.concat([waveform, zero_padding], 0)
                waveform = waveform[:max_length]  # Kürzen, falls länger als max_length

                all_waveforms.append(waveform)
                all_labels.append(LABELS.index(label))
    return np.array(all_waveforms), np.array(all_labels)

# Laden der Daten
waveforms, labels = load_data(DATASET_PATH)
X_train, X_test, y_train, y_test = train_test_split(waveforms, labels, test_size=0.4, ran-
dom_state=42)

# TensorFlow Lite-Modell laden
# TensorFlow Lite-Modell laden
interpreter = tf.lite.Interpreter(model_path="C:\\Users\\seraf\\Documents\\Schule\\Module\\PA\\Ten-
sorflowModellTest\\PA_Emotion_Modell\\TFLite\\model.tflite")
interpreter.allocate_tensors()
```

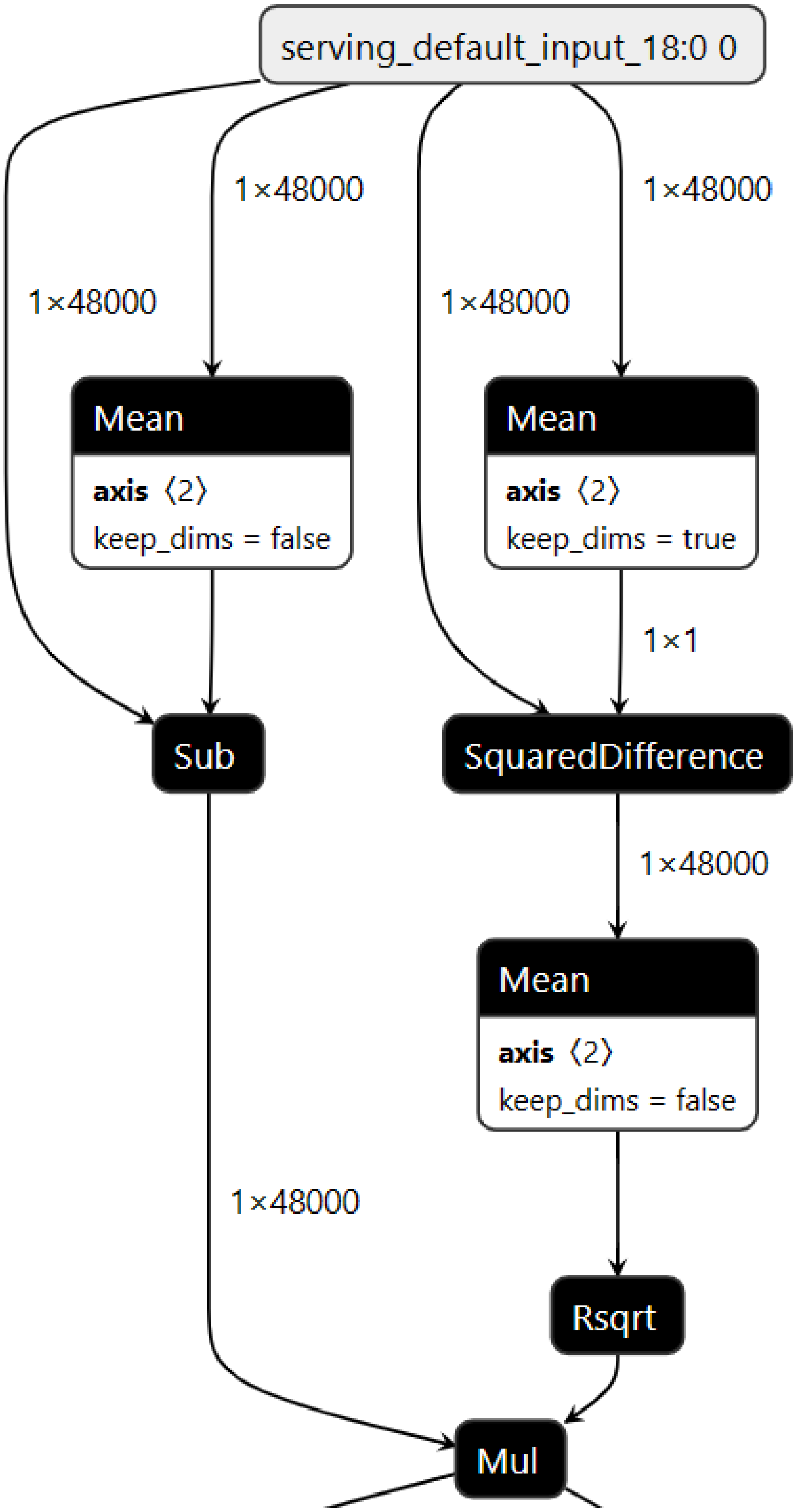
```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Vorhersagen auf den Validierungsdaten
predicted_labels = []
for waveform in X_val:
    # Sicherstellen, dass die Wellenform die richtige Länge hat
    if len(waveform) > 48000:
        waveform = waveform[:48000]
    elif len(waveform) < 48000:
        waveform = np.pad(waveform, (0, 48000 - len(waveform)), 'constant')

    # Erweitern der Dimension, um die Batch-Größe zu repräsentieren
    waveform = np.expand_dims(waveform, axis=0)

    # Sicherstellen, dass die Wellenform die richtige Form hat
    if waveform.shape == (1, 48000):
        interpreter.set_tensor(input_details[0]['index'], waveform)
        interpreter.invoke()
        output_data = interpreter.get_tensor(output_details[0]['index'])
        predicted_label = np.argmax(output_data, axis=1)
        predicted_labels.extend(predicted_label)
    else:
        print("Fehlerhafte Wellenform-Form:", waveform.shape)

# Berechnung der Konfusionsmatrix
cm = confusion_matrix(y_val, predicted_labels)
cm_percent = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Plot der Konfusionsmatrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm_percent, annot=True, fmt=".1%", cmap="Blues", xticklabels=LABELS, ytick-
labels=LABELS)
```

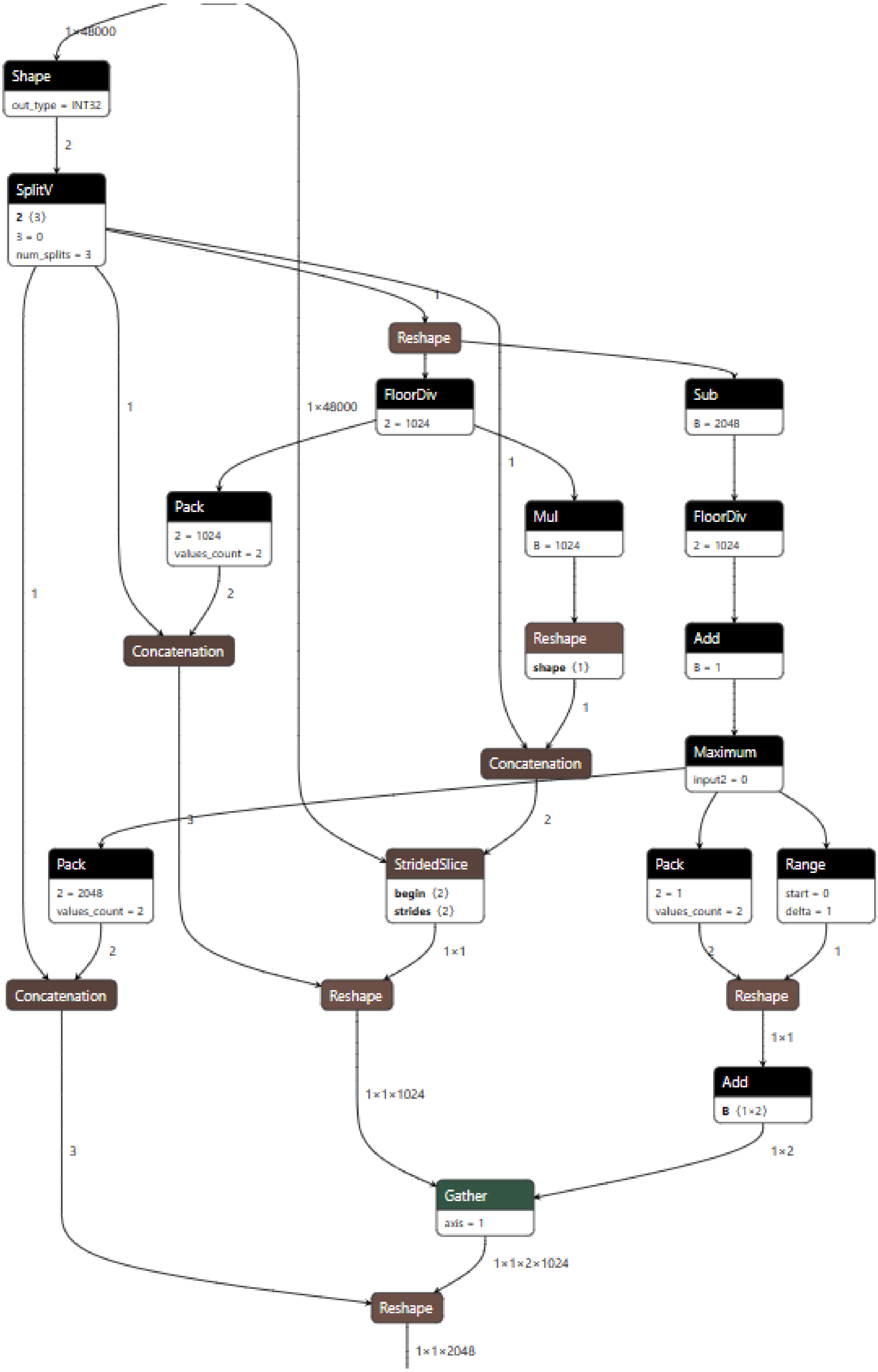
```
plt.xlabel('Vorhergesagte Labels')
plt.ylabel('Tatsächliche Labels')
plt.title('Konfusionsmatrix in Prozent')
plt.show()
```

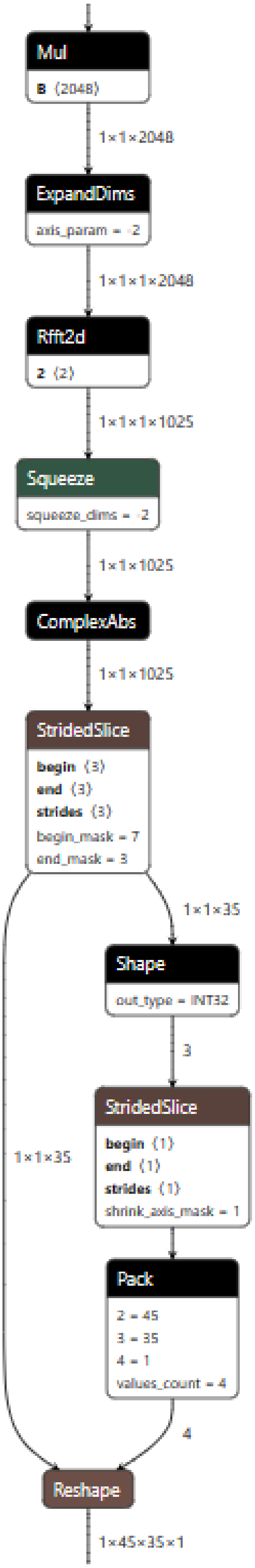


7.3 Model Structure

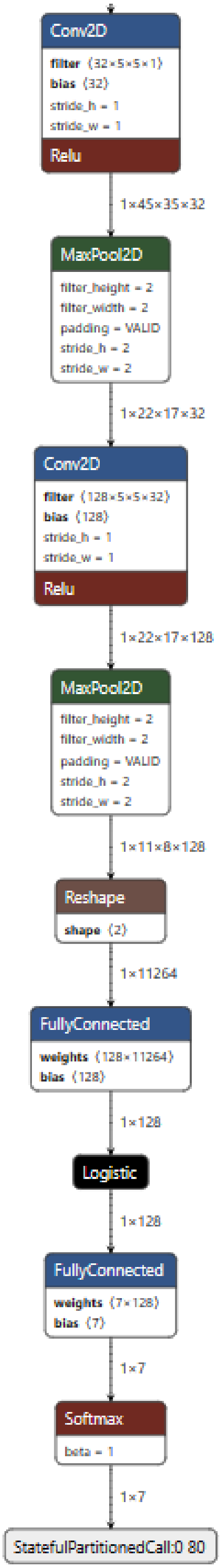
7.3.1 Model Structure TensorFlow Lite Model


serving_default_input_18:0 0
1×48000
1×48000
1×48000
1×48000
Mean
axis ⟨2⟩
keep_dims = false
Mean
axis ⟨2⟩
keep_dims = true
1×1
Sub
SquaredDifference
1×48000
Mean
axis ⟨2⟩
keep_dims = false
1×48000
Rsqrt
Mul


1×48000
Shape
out_type = INT32
2
SplitV
2 ⟨3⟩
3 = 0
num_splits = 3
1
Reshape
FloorDiv
2 = 1024
Sub
B = 2048
1
1×48000
1
Pack
2 = 1024
values_count = 2
Mul
B = 1024
FloorDiv
2 = 1024
1
2
Concatenation
Reshape
shape ⟨1⟩
Add
B = 1
1
Concatenation
Maximum
input2 = 0
3
2
Pack
2 = 2048
values_count = 2
StridedSlice
begin ⟨2⟩
strides ⟨2⟩
Pack
2 = 1
values_count = 2
Range
start = 0
delta = 1
2
1×1
2
1
Concatenation
Reshape
Reshape
1×1
Add
B ⟨1×2⟩
1×1×1024
3
1×2
Gather
axis = 1
1×1×2×1024
Reshape
1×1×2048



Mul
B ⟨2048⟩
1×1×2048
ExpandDims
axis_param = -2
1×1×1×2048
Rfft2d
2 ⟨2⟩
1×1×1×1025
Squeeze
squeeze_dims = -2
1×1×1025
ComplexAbs
1×1×1025
StridedSlice
begin ⟨3⟩
end ⟨3⟩
strides ⟨3⟩
begin_mask = 7
end_mask = 3
1×1×35
Shape
out_type = INT32
3
StridedSlice
begin ⟨1⟩
end ⟨1⟩
strides ⟨1⟩
shrink_axis_mask = 1
1×1×35
Pack
2 = 45
3 = 35
4 = 1
values_count = 4
4
Reshape
1×45×35×1


Conv2D
filter ⟨32×5×5×1⟩
bias ⟨32⟩
stride_h = 1
stride_w = 1
Relu
1×45×35×32
MaxPool2D
filter_height = 2
filter_width = 2
padding = VALID
stride_h = 2
stride_w = 2
1×22×17×32
Conv2D
filter ⟨128×5×5×32⟩
bias ⟨128⟩
stride_h = 1
stride_w = 1
Relu
1×22×17×128
MaxPool2D
filter_height = 2
filter_width = 2
padding = VALID
stride_h = 2
stride_w = 2
1×11×8×128
Reshape
shape ⟨2⟩
1×11264
FullyConnected
weights ⟨128×11264⟩
bias ⟨128⟩
1×128
Logistic
1×128
FullyConnected
weights ⟨7×128⟩
bias ⟨7⟩
1×7
Softmax
beta = 1
1×7
StatefulPartitionedCall:0 80