An End-to-end Deep Learning Framework for Acoustic Emotion Recognition

Bachelor Thesis

by

Michael Sawin

Reviewer:	Prof. Dr. Dr.-Ing. Wolfgang Minker
Supervisor:	M.Sc. Denis Dresvyanskiy

Faculty of Engineering, Computer Science and Psychology
Institute of Communications Engineering
University of Ulm
17 April 2023

Version April 16, 2023

© 2023 Michael Sawin

This work is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Abstract

This work examines acoustic emotion recognition by a system using only an audio file containing a person's speech. This can be applied in several areas of human-computer-interaction (HCI). In this context, a system consisting of only a microphone and a processor is less expensive than other methods that use additional variants for emotion recognition. This thesis explores how automatic emotion recognition by a system using only speech as input works. This work provides insight into different technologies that can be applied for emotion recognition of a system.

Feature extraction and computation of Mel spectrograms from audio files was investigated and their usage as input to a machine learning algorithm or a neural network. A framework was developed using a model, which is well suited to classify an emotion for a short or longer audio clip.

The experiments of this thesis show the success that can be achieved using convolutional neural networks and Mel spectrograms as its input to classify an emotion based only on an audio file. This in combination with different training methods has results in a framework that can classify emotions from short but also longer audio files in a very short time.

In summary, automatic classification of emotions from a system is a very interesting topic, where multiple methods can lead to different successes. With the utilization of the results of this work, a further step towards improved HCI can be taken, as systems can adapt their action to the emotion state of the user.

Contents

List of Figures

List of Tables

1. Introduction

Emotions are playing an increasingly important role in human-computer interaction (HCI). It can find application in the areas of call centers and cars. A system that can detect a person's emotion based only on speech does not need cameras or sensors. All it needs is a microphone and a processor that can perform the calculations quickly. The recognition of emotions helps the system to adapt a more appropriate response or even action to the emotional state of a person.
There has been research on speech emotion recognition (SER) of a system for a while now. It started with constructing features that can be extracted from an audio file and then applying machine learning algorithms. More and more, neural learning is being used as an approach for SER [37]. This is because neural networks can recognize more complex patterns in certain data, learn them, and apply them afterwards.
The interest in this topic is based on building a system that can assign certain emotions based on acoustic signals only, i.e. an audio file. The tendency nowadays for such a framework is more and more to use neural networks for it [16, 27, 44, 55], furthermore it is a fast growing research area [37]. Preprocessing is required for the use of machine learning algorithms. Several static features are extracted from the audio track, which are helpful in determining the emotion. The resulting one dimensional vector is the input for an algorithm, based on which an approximation of the associated emotion can be made. Because emotions are temporary and the preprocessing takes longer, this variant tends to have a lower performance than the variant that uses neural networks [44]. The reason for this is that with the use of neural networks there is a tendency to use a resource saving preprocessing, so to speak the input is almost the raw audio track.
In this paper, we investigate how neural networks perform in contrast to machine learning algorithms. For this purpose, the machine learning algorithms *Support Vector Machine* (SVM), *Multi-layer Percepton Classifier* (MLP) und *Decision Tree Classifier* (DTC) are examined in more detail, including the associated preprocessing. In the context of neural networks, deep neural networks, convolutional neural networks and transfer learning are discussed in detail. The application of mel spectrograms and their computation, which serve as input for the experiments, is discussed. Important to mention here is the proposed framework that is developed in terms of chunking of audio files. Chunking splits an audio file into several parts of fixed size, in order to classify the emotion using them. With the help of chunking, many audio files no longer need to be padded with zeros if they have a longer playback time than the beforehand fixed size of the chunks.
Continuing with the outlook, everything that is covered in this paper. Section 2 summarizes related work and talks about similarities. Section 3 discusses what is important for a good dataset, which is suitable to build a SER system. In addition, it is mentioned what types of datasets there are. In section 4 follows the explanation of how feature

extraction works. This work refers to the toolkit of *openSMILE* [24], in combination with the feature set *ComParE 2016* [23, 60]. After that, section 5 explains the handcrafted machine learning algorithms SVM, MLP and DTC, which receive the *openSMILE ComParE 2016* features as input. In section 6, *Deep Neural Networks*, *Convolutional Neural Networks* and *Transfer Learning* are explained. In this work, the focus is on neural networks because a resulting model is applied in the end-to-end framework, of this work. In section 7 it is explained what possibilities exist in building this framework. In section 8, the experiments of this work are discussed in more detail. Discussion includes the used datasets, the steps of preprocessing, the results of the machine learning algorithms, and the results of the neural networks. Also, the implemented framework is presented and its speed in classifying emotions is tested. At the end in Section 9 follows a short summary and reflection of this work and recommendations on how to follow up with this work. In the appendix A the results of the applied machine learning algorithms SVM, MLP and DTC can be found in visual representation. In addition, the custom CNN architecture in PyTorch [56] notation can be found there as well. And there are graphs for each training of a neural network, how the results changed during the training.

2. Related Work

Comparable to this work is any work that trains CNN or uses transfer learning with existing CNN models on the data from the MSP-Podcast [46], ASVP-ESD [39] and CREMA-D [8] datasets for the purpose of classifying specific emotion classes. In the work of Landry et al. [39], the ASVP-ESD dataset is used to compute mel spectrograms from it as in this work and use that as input to a CNN. The difference here is that the work of Landry et al. [39] uses a model that after the CNN has a recurrent neural network for classifying the features from the CNN. The work of Mekruksavanich et al. [51] uses the data of the CREMA-D to classify positive and negative emotions with a CNN. The difference with this work is that not only positive and negative emotions are classified, but the available emotion classes from CREMA-D should be classified and that mel spectrograms and not MFCC features are used here. The work of Zeghidour et al. [74] works with the data from the CREMA-D dataset to be able to classify the emotion classes. A CNN architecture is also used there, and for this reason, the result is compared in chapter 8.
In addition, this work is about a working end-to-end framework built on an artificial model. This framework uses the technique of chunking to split the input in the form of an audio file into several parts of fixed size. Then, an emotion is classified for each chunk. This has similarities to the work of [44], as they are also experimenting on a framework that uses the principle of chunking. Also, there each chunk has a fixed size after which an audio file is divided into several parts. Thus, both frameworks aim to minimize the use of zero padding. Zero padding is a technique to extend an audio file or a mel spectrogram by adding zeros to the audio file or the mel spectrogram. In the case of an audio file, it is equivalent to a mute sound being added.

3. Datasets

3.1. Characteristics of datasets

Training with audio files enables the system to recognize emotions based on speech. The existing datasets have various characteristics.
Considering the dataset size is an important aspect. A larger dataset can lead to better results in terms of accuracy since the model has a larger basis to learn from [46].
Additionally, the dataset should have high diversity, with various emotions spoken by different individuals. This is due to the individuality of humans, which can affect the expression of emotions. The occurrence of emotions is often context-dependent. Therefore, it is advantageous for the dataset to include different speakers and contrasting situations. The diversity of a dataset allows the model to make generalized statements about emotional states [46].
Many individuals need to be consulted to classify the emotions of audio files [8, 46]. The majority decides which main emotion, or label, is assigned to each individual audio file. The process of labeling the audio files with emotions is an expensive operation, considering time and resources [40]. The number of labels may vary depending on the dataset, which can be an advantage as the trained model can distinguish between multiple emotions. However, it can also be a disadvantage as the emotions can become similar after preprocessing the data, leading to a higher level of inaccuracy in the output. Different datasets may contain different emotions and the number of emotions may vary represents a limitation. The accuracy of emotion recognition will be lower if the dataset used for validation has a higher number of emotions than the dataset used to train the model.
The data after preprocessing can be distorted due to the audio files containing loud background noises or too many interference signals. When creating such a dataset, aiming for high audio quality is important [8, 46], as this is critical for model training. The frequent occurrence of similar interference signals can be interpreted by the model as a pattern.
Additional information such as gender, cultural background, and age of the speaker is useful as these factors can influence the expression of emotions [8].
A uniform distribution of emotions in a dataset is advantageous for training a model. This prevents the model from developing a tendency towards a particular emotion.
The individuality of the speakers and their cultural background has already been mentioned. Additionally, the spoken language is another significant factor. Expressing emotions can differ from language to language. In order to train a model that can recognize emotions based on different languages, these must also be present in the dataset. In this regard, it is also important to ensure equal distribution. Otherwise, it is possible for the model to develop a bias towards a certain language or dialect, causing the accuracy in

recognizing emotions in another language to decrease.

3.2. Types of datasets

There are three main types of datasets, most of the datasets that can be used for SER come from professional speaker recordings. These make up almost 60% of the datasets. [37]. It may well be that a trained model with a particular type of dataset will give better results in terms of accuracy because it has been trained with such a particular type of dataset. Conversely, a model trained with one type may give worse results in terms of accuracy when validated with a dataset of a different type. All three types of datasets can be used to derive acoustic features that then serve as input to machine learning algorithms. Or used almost in raw form to train a neural network.

- Simulated dataset:
  Speakers are asked to repeat different sentences with different emotions. These are typically recorded in a controlled environment. A scripted situation is to be acted out by the speakers with a pre-determined emotion. These types of datasets are easy to create [37] because they can save a lot of resources and time, since the process of having many people assign an emotion to a particular audio file is eliminated here. The reason for this is that the professional speakers perform a specific emotion in advance. If there are any discrepancies regarding the recording and the associated emotion, then this can be recorded again. Also, you have control over the quality of the recording. However, there is a challenge here, and that is that the emotions, because they are acted out, tend not to be identical to the emotion as it might occur in a natural environment [46].

- Semi-simulated dataset:
  In this type of dataset, an artificial emotional situation is created in order to preserve the emotions in a semi-natural way [37]. In addition, care is taken to ensure that the recorded emotions are the result of natural dialogues between several people and are not monologues [7]. Especially not a monologue as in the simulated datasets, where professional speakers, or actors create an artificial emotion when speaking. A higher resource input is required because unlike the simulated datasets, a classification of emotions to the corresponding audio files is necessary.

- Natural dataset:
  This dataset contains the most natural emotions. This is because no artificial emotional situation is created here, as in the semi-simulated datasets. The recordings are also not acted out in a controlled environment by professional speakers, actors or actresses, as in the simulated datasets. Natural datasets use pre-existing voice recordings, snippets from interviews or call centers, or public conversations [37]. For this reason, such a dataset may contain a variety of emotions. The challenge in creating such a dataset is that it is very resource intensive and time-consuming. This is because several people have to assign emotions to all the audio files so that a model can be trained with this data.

4. Features

4.1. Handcrafted Features

Handcrafted Features are human-made extraction methods. Their goal is to extract information from an audio file so that a system can work with it. There are many kinds of features, this work focuses only on those that provide information about emotions. When creating such features, care should be taken that they provide suitable information to be able to assign them to different emotions. For this, preliminary work by individuals is necessary. Moreover, it is important that such features be well-chosen to train a machine learning model with their help.
Basically, the features of an audio file are obtained as shown in Figure 4.1. An audio file is divided into several frames. A frame should not be shorter than 10 ms, because people only perceive a sound when the length exceeds 10 ms [18]. *Time-domain features* are then calculated for each frame. These are then aggregated. This means that the features of all frames are combined to form complete features over the entire length of the audio file. This could be done using an average value, for example, but much more complex methods can be used to obtain information that describes the entire audio file. The computation and aggregation of features can be abstracted as *feature extraction*. Finally, a vector is obtained, full of features and their associated values extracted from an audio file. These can be used to train a machine learning model.
In this work, openSMILE [24] is used as a feature extraction toolkit. This toolkit can be used to extract speech-related features from an audio file, including features that help classify emotions. This audio file does not need to be edited in any way, but can be taken in raw form as input for feature extraction. OpenSMILE is divided into four components, *Data Source*, *Data Memory*, *Data Processors* and *Data Sinks* [24]. The Data Sources component, reads existing data from a device, interface or disk and stores it in the Data Memory component. Data Memory is the centerpiece of the whole construct. The data processors process the data by applying defined algorithms to the data in the data memory. After the calculations, the changes or results are written back to the data memory. The data sinks are responsible for storing the results of the audio file for the user. These can be stored, for example, as a vector in a CSV file [24].
For the classification of emotions of machine learning models, this work uses the feature set *ComParE_2016* [60]. This feature set has, 6373 static features, which were chosen for the classification of emotions [60]. Thus, each extraction of features from an audio file produces a vector with the dimension 6373.

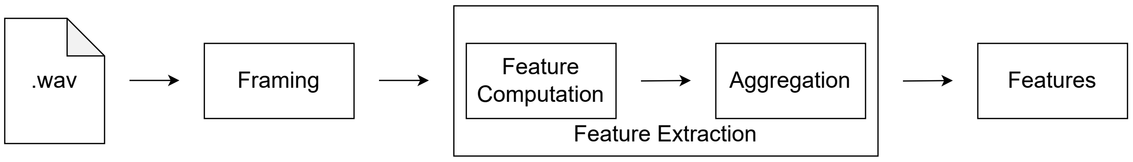


Figure 4.1.: Pipeline for feature extraction of an audio file.

4.2. Spectrograms

Spectrograms are two-dimensional images that visualize an audio track, where the color and brightness represent the strength of the frequency at a given time [19, 38, 73]. The scheme of the coordinate system of a spectrogram is constructed in such a way that it has the time on the x-axis and the frequency on the y-axis. Thus, a spectrogram provides the information at which time which frequencies are used [19, 38, 73].
Other than the handcrafted features, the spectrograms are used for training a *convolutional neural network* (CNN) as described in section 6.2. The advantage here is that, unlike machine learning algorithms, the training is not based on human-created features. This is because information from the audio track can be lost in the process of extracting handcrafted features, or simply not be used. A CNN trains directly with the spectrogram that represents the complete audio track. Thus, the CNN has the frequency information at any time of the audio track and can derive its own features or even recognize patterns. When applying the *fast fourier transform* (FFT) , information is obtained from the occurring frequencies, without temporal background information [19]. For this reason, a spectrogram is created with the method *gabor transform*, also known as *short-time fourier transform* (STFT). Here a Gaussian window is used, within which FFT is performed [6, 38]. This window can also have another form, for example a quadratic form. By repeatedly moving this window on the time axis and performing FFT in this area, frequency information is obtained at a point in time. The window is shifted with a certain value from the beginning to the end of the audio track. This is shown in figure 4.2. Here a point τ is set, from which the window extends in both directions on the time axis by the value a.
With this technique, a spectrogram is obtained which contains frequency information, depending on time. Such a simple spectrogram is presented in figure 4.4a. To obtain additional perceptually relevant frequency information, *mel spectrograms* are needed. The reason for this is that humans perceive frequencies in a logarithmic manner [19]. This means that people perceive a distance in Hz, in the lower frequency range more clearly than a distance of the same size in a high frequency range. Therefore, the *mel scale* is used, where the frequency range is perceptually the same.
To create such a mel spectrogram, it is necessary to have the simple spectrogram first, as described above. In addition, a mel filter bank is used to extract frequency bands [26, 48]. These frequency bands are designed in such a way that the distance from the start point to the end point of the band is perceived the same in the low frequency range as in the high frequency range. Figure 4.3 visualizes this. It can be seen there that the distance

from the starting point to the end point of a band increases significantly in the frequency range, but it is still the same in the mel scale and is therefore perceived by the human ear as the same distance. Each filter in the mel filter bank has the shape of a triangle. This extends on the axis of the amplitude from zero to one. Zero is equal to a mute sound and one is equal to the greatest power of the frequency. To calculate the individual filters in the mel filter bank, two formulas are needed for converting from hertz to mel scale and converting from mel scale back to hertz [26, 48, 49, 70].

$$m = 2595 \cdot \log_{10}(1 + \frac{f}{700}) \tag{4.1}$$

$$f = 700(10^{\frac{m}{2595}} - 1) \tag{4.2}$$

In the equations, m stands for the mel scale and f for the frequency in Hertz. Thus, 1000 Hertz corresponds approximately to 1000 mel [70]. Now it is necessary to convert the whole frequency range from Hertz to mel, for this equation 4.1 is used. Then divide this value by the number of filters desired. The result is the distance between the maximum point of one filter and the maximum point of its neighboring filter. If the value of the distance is divided by the value of two, the maximum point of the first filter is obtained. The starting point of the first filter starts at frequency equal to zero. And the end point of a filter is calculated by adding to the maximum value half the distance between the maximum points of the filters. These mel values of each point must then be converted into Hertz with equation 4.2. After using the equations, from the origin, i.e. the starting point of the filter, to the maximum point of the filter, to the end point of the filter, a triangle is formed, which is a filter band. For the next filter, its starting point is at the maximum point of the previous filter, with the amplitude value equal to zero.

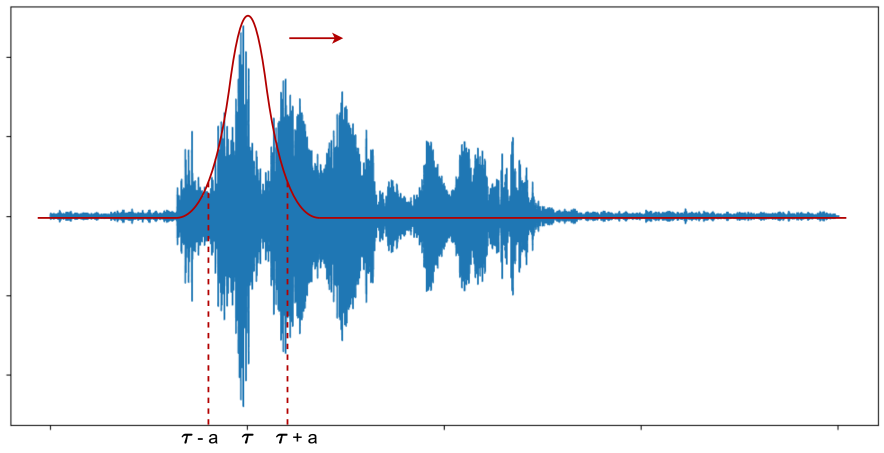


Figure 4.2.: Functionality of gabor transform. Audio visualization extracted from 1001_IOM_HAP_XX.wav (CREMA-D) [8].

After creating the mel filter bank, we only need to apply it to the previously created simple spectrogram to obtain the mel spectrogram [26, 48].
By obtaining the mel spectrogram, in addition to the temporal frequency representation, a perceptual frequency information is now available. Such a mel spectrogram is shown in figure 4.4b. The difference between a simple and a mel spectrogram, are shown visually in figure 4.4. Mel spectrograms are usually applied, instead of simple spectrograms, to train a convolutional neural network [19]. In this work, they are used to train a convolutional neural network model. The use of simple spectrograms has been avoided due to the extra information that a Mel spectrogram provides.

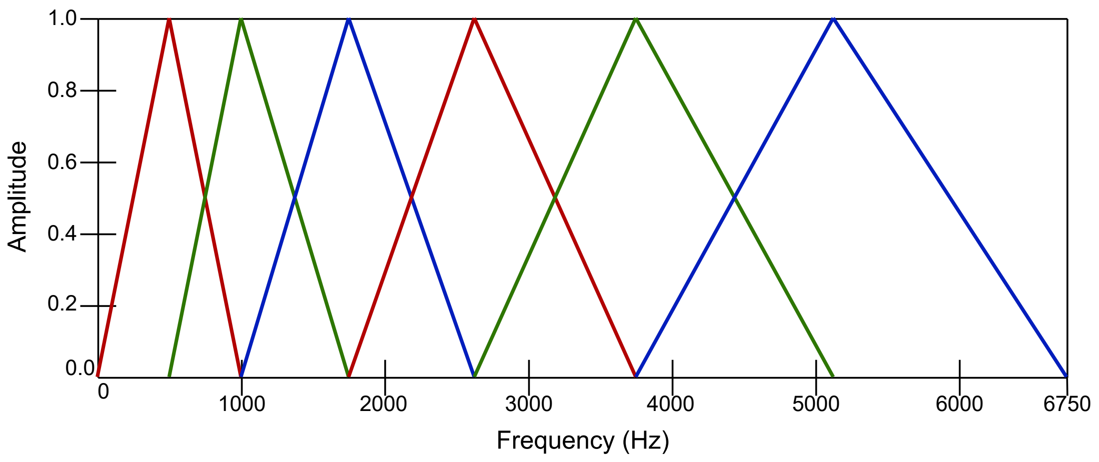
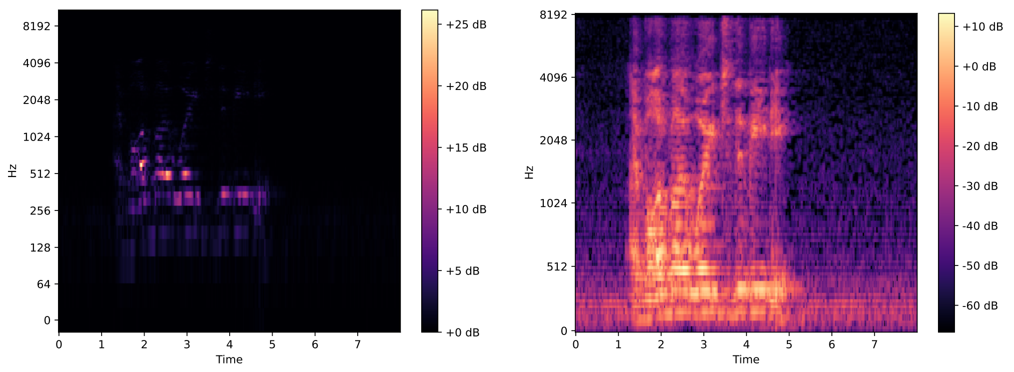


Figure 4.3.: Filter bank with six mel filter bands..



(a) Simple spectrogram with window size 512.
(b) Mel spectrogram with window size 512 and 128 filter bands.

Figure 4.4.: Difference between simple and mel spectrogram visualized in figures 4.4a and 4.4b.

5. Machine Learning Algorithms

5.1. Support Vector Machine

Support Vector Machine (SVM) is a simple machine learning algorithm. SVM works with a hyper plane. This is there to separate the existing categories. The hyper plane is determined by calculations with the input. Because SVM is a supervised learning algorithm [3], the data must each have a label indicating which category it belongs to. This algorithm tries to determine the best fitting hyper plane to separate the points of the different categories as good as possible. Thereby, these mentioned points, are vectors that represent the input data. The input data must be vectors so that the SVM algorithm can perform its calculations with them. This means that before using SVM, preprocessing must take place, which extracts various features from the audio files. As mentioned in section 4, these features must be suitable for the recognition of emotions.
The hyper plane is chosen so that it has the greatest possible distance to the supporting vectors. Such a supporting vector is the vector of a category, which is closest to the hyper plane [3]. This is illustrated in figure 5.1, which shows that all points of a category should be on the right side of the hyper plane.

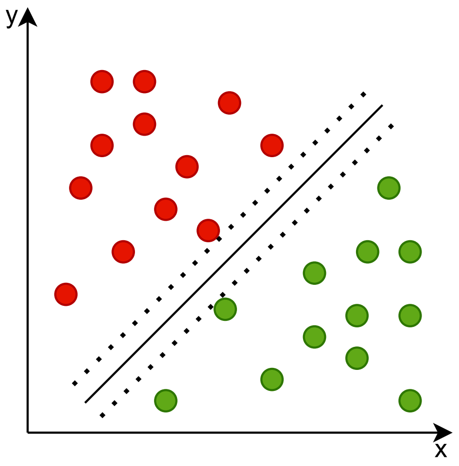


Figure 5.1.: 2D hyper plane visualization. Different color stands for diefferent emotion classes.

Nevertheless, there is a problem with this variant of SVM. It may well be that one category is completely enclosed by another, as can be seen in figure 5.2. In such a case SVM cannot

determine a hyper plane. For this reason the vectors are transformed. Each category has a different mathematical scheme according to which its vectors are transformed. This is helpful because with the addition of another dimension to each vector, a clear dividing line can be drawn between the categories. By adding one more dimension, SVM can now determine a hyper plane, as seen in figure 5.3.
To solve this problem, SVM proceeds as follows:

(i) Increases the vectors of the input data by one more dimension.

(ii) Determines a hyper plane in the space, which has been extended by one dimension to separate the vectors of the different categories.

(iii) After the categories are separated, the dimension is decreased by one.

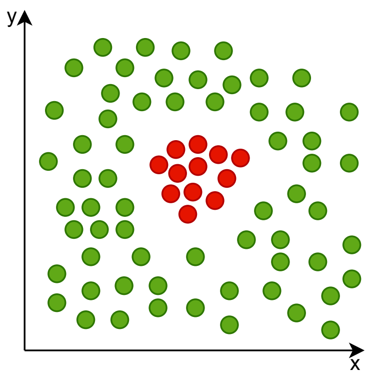


Figure 5.2.: Problem of surrounding classes.

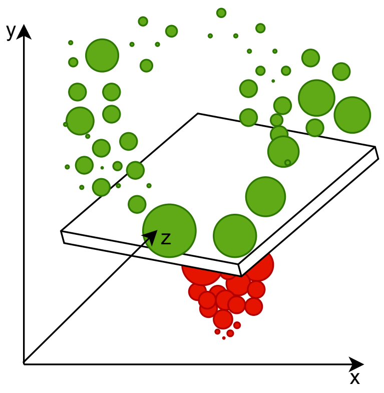


Figure 5.3.: Transformation from 2D as shown in figure 5.2 to a 3D space.

Applying a function to a vector, i.e. the mentioned transformation, represents a higher computational effort. This can be avoided, with the use of kernel functions. Furthermore, in practice, these three steps do not have to be taken care of because the kernel functions already take care of that [62]. Additionally, there is not this increased computational overhead because using the kernel functions, SVM does not need to know what each vector looks like after the transformation, in other words $x \longrightarrow f(x)$ [3]. Only SVM needs to know how each vector differs to every other vector after they have been transformed. The linear kernel function, or just kernel as it is called in in the work of Baştanlar et al. [3], is the product of a transformed vector x with another transformed vector $K(x, \hat{x}) = f(x)^T f(\hat{x})$ [3]. In addition to the linear kernel function, the polynomial, the sigmoid and the radial basis function (RBF) are also used as kernel functions in practice [3].
SVM is designed to be able to distinguish and classify two classes, two categories, as can be seen in figures 5.1 and 5.3. This is a problem when working with multiple classes,

which is the case when recognizing emotions because usually there are multiple of them. Without changing SVM, this problem can be solved by creating $n \cdot (n-1)/2$, where n represents the number of classes. Each of them is trained with the data of two classes [62]. Thus, it is possible to apply SVM for classifying emotions.

5.2. Multi-layer Perceptron Classifier

Multi-layer perceptron (MLP) is a neural network architecture and belongs to the subcategory of *feed-forward networks* [28]. When creating artificial neural networks, inspiration was taken from the human brain and attempts are being made to model it, so a system can work with it [53].
The most important component of MLP is the perceptron. This has one or more inputs. With these, the perceptron performs calculations and forwards it to one or more outputs. As the name indicates, MLP consists of many interconnected perceptrons. MLP always follows the same structure. It has an *input layer*, one to several *hidden layers* and an *output layer*, as can be seen in figure 5.4. In addition, it can be seen that an n-dimensional input vector can be given to the model, where n stands for the number of handcrafted features extracted, as discussed in section 4.1. The output of MLP is an m-dimensional vector, where m represents the number of emotions to be classified. The index with the highest value of the output vector indicates the predicted emotion. Further, the perceptrons in this section are referred to as nodes.
In the experiments of this paper, MLP is used only to classify emotions from handcrafted features. The input layer consists of n nodes, where n is the number of dimensions of the input vectors, thus the number of features of the input. This is then followed by the hidden layers. The number of layers in this area can be freely chosen. And also the number of nodes per layer can be chosen freely. The only limitation for the number of layers and their nodes, is the memory of the system, the bigger the created model, the more memory is needed. In the hidden layer area, the features are processed to make a prediction in the output layer afterward. The outputs of the last hidden layer are forwarded to the output layer. This consists of m nodes. Where m is the number of emotions that the model has to classify.
Each node of the previous layer is linked to each node of the next layer. Each link has a weight, which is calculated when processing the data from one layer to another. When such a neural network is initialized, the weights are set with random numbers between -1 and 1 [65]. These weights lead to the result that, given an input, the model can make a classification. In the training phase of the model with MLP, the prediction of the model is compared with the real emotion in the output layer. An error is calculated. This error has to be minimized in the training phase and for this the procedure *gradient descent* can be used. This procedure tries to determine the absolute minimum of the error in a space [28].
In the training phase, MLP follows an algorithm that can be briefly summarized into the steps of guess, change, and repeat. In the guessing stage, when a new input vector is re-

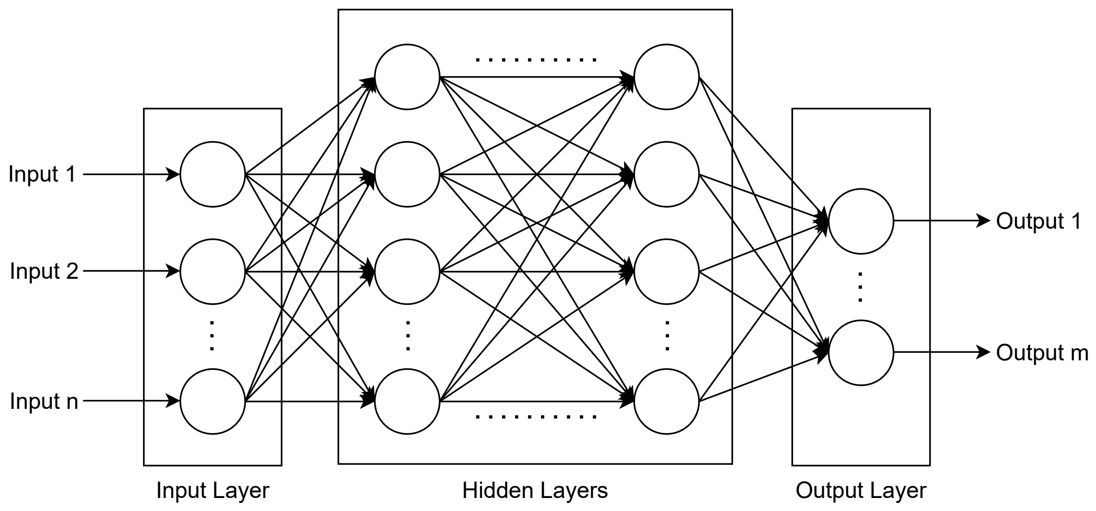


Figure 5.4.: MLP architecture with n for incoming vector size and m for outgoing vector size.

ceived, it is a matter of setting the various weights of the individual nodes that determine the output in the final effect. Then, the error is calculated, that is, the difference between the prediction and the real emotion. When optimizing the weights, in the process of *back propagation*, the weights of the nodes are adjusted with respect to the error. As the name implies, this time we start from the back, thus from the last layer and go up to the first layer. This is followed by the next step of repeating, where the first steps are performed with the next input vector. The execution of these steps, with all input vectors, is called an epoch. The use of multiple epochs, can lead to a better performance of the model, in terms of classification of emotions. Nevertheless, after a certain point, the performance may decrease again, due to overfitting. This can also happen if a neural network is too large, with many hidden layers and too many nodes. This can also lead to overfitting, because the neural network has more opportunity to learn all the patterns of the training dataset [4, 28]. And for this reason it is necessary to have a validation dataset, which is not shown to the model during the training phase.
In the work of Gardner et al. [28] these summarized three steps are separated into seven detailed steps. These are presented in the following in modified form using six steps.

1. When the neural network is initialized, the weights of each node are set randomly.

2. This is followed by input in the form of an input vector, which passes through the entire neural network to make a prediction of an emotion.

3. After the output layer, the predicted emotion is compared with the real emotion and the error is calculated.

4. This is followed by the back propagation, which passes this error from the last layer to the first layer. In relation to this error, the weights of the connections between

the nodes are adjusted in order to minimize the error.

5. The steps 2 to 4 will be repeated for each further input vector, this is called an epoch. The goal of MLP is that the error after an epoch is as small as possible.

6. Another epoch begins and steps 1 to 5 are repeated.

Another important point are the activation functions. A neural network without the usage of activation functions cannot recognize complex relationships between the data [65] and thus it would not be useful for a model that is supposed to classify emotions because emotions can have complex characteristics that need to be recognized. With the addition of an activation function, more complex correlations can be recognized.
ReLU is an often used activation function in neural networks. It is also very performant, because it does not activate all nodes at the same time [65]. If the input of ReLU is less than 0, the node is not activated, because the threshold of 0 was not crossed. Only if the input is greater than 0, the threshold of ReLU is exceeded and the node is activated. The mathematical formula for ReLU is $f(x) = max(0, x)$ [65]. Thus, all values less than 0 are set to 0 and all positive values remain at its previous value. The activation function ReLU is also often used in emotion recognition. The output layer, of a model that classifies emotions, has SVM or Softmax as its activation function to assign probabilities to the neural network outputs [37]. The emotion with the highest probability is the result of the prediction of the neural network. In this work, Softmax is used, which will be discussed later.

5.3. Decision Tree Classifier

Decision Tree Classifier (DTC) is a machine learning algorithm in which, as the name suggests, a tree architecture is generated for classifying classes, in this case emotions. The tree structure consists of an origin node, several decision nodes, several leaves, and the edges connecting the nodes. Figure 5.5 shows an example tree structure and the different colors stand for different emotions. It can also be seen that each node that is not a leaf makes a decision D. The leaves are responsible for classifying emotions. In addition, the duration of the training of DTC is much shorter than that of MLP [59].
The DTC algorithm makes different decisions $\{D_1, D_2, ..., D_n\}$ for the selection of a condition of a node, from which the decision with the highest information gain for this node is taken [20, 59]. DTC tries to maximize the information gain. The decisions made within a node lead to a separation of the data. In this work, the data is presented as a set of vectors. Each vector represents one emotion. The number of emotions depends on the dataset used in the training phase.
In this work, the layers of a tree are continued as levels. With each further level, the impurity of the data decreases. A higher impurity of the data means that vectors corresponding to different emotions are together in one set. In contrast, a lower impurity means that almost all the vectors in the remaining set represent the same emotion. Ideally, a leaf contains a subset that contains only vectors corresponding to a single emotion.

DTC follows the approach to divide a complex decision into several decisions [59] and therefore a tree structure is used. Because here a decision is always based on a decision made before. In DTC, each node, except the leaves, makes several decisions on which scheme to use to split the dataset in subsets. If the vectors have been successfully split, then a leaf contains only vectors of one emotion. When a vector is given into this tree structure, then it is routed by the nodes' specified decisions until it reaches a particular leaf. Based on the leaves, DTC can assign the emotion to which each vector belongs. This can be seen in figure 5.6. The decisions, which are marked with blue lines in the figure, divide a space into subspaces. Now, when a new vector is presented to the tree structure created by the DTC algorithm, it is placed in this space and depending on which subspace it is in, an emotion is assigned to this vector.

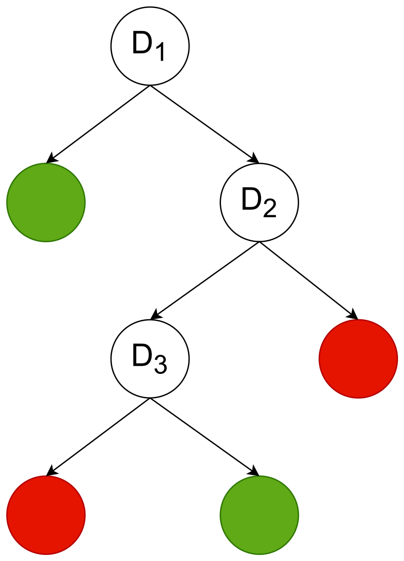


Figure 5.5.: Tree structure that could be generated by DTC.

Each node, except the leaves, creates several decisions, which are made using different schemes. From this set of decisions $\{D_1, D_2, D_n\}$, the best decision D_i is selected. For this, it is looked which decision brings the highest information gain. The number of nodes that make a decision based on a condition is proportional to the number of times the incoming dataset is split. Thus, as can be seen in Figure 5.6, DTC can divide a linearly inseparable set of vectors, with several linear operations.
To calculate the information gain, it is necessary to find out the amount of information contained in the set. This can be calculated with the *entropy*, or the *gini index*. Therefore, we can use the formulas from [20, 59].

$$Entropy = -\sum p_i \cdot log(p_i) \tag{5.1}$$

$$Gini\ index = 1 - \sum p_i^2 \tag{5.2}$$

Here, p_i stands for the probability of occurrence of an emotion i in a set of vectors. The calculation of the gini index is more performant than the calculation of the entropy. This is due to the use of the logarithmic function in the entropy formula. Depending on the use of the entropy or the gini index, it will be built into the formula of the information gain [20].

$$information\ gain = Entropy(S) - \sum \frac{|S_i|}{|S|} \cdot Entropy(S_i) \tag{5.3}$$

$$information\ gain = Gini\_index(S) - \sum \frac{|S_i|}{|S|} \cdot Gini\_index(S_i) \tag{5.4}$$

In equations 5.3 and 5.4, S stands for the set of vectors, which is divided into subsets S_i. $|S|$ denotes the cardinality of the set S, that is, the number of vectors in the set S. $|S_i|$ denotes the cardinality of the subset S_i, that is, the number of vectors in the subset S_i. Each decision D partitions the set S into subsets. The highest information gain is equivalent to the best division. This means that the decision of a node is set if the information gain of it is the highest of all calculations so that the decision can split the set into the best constellation of subsets.

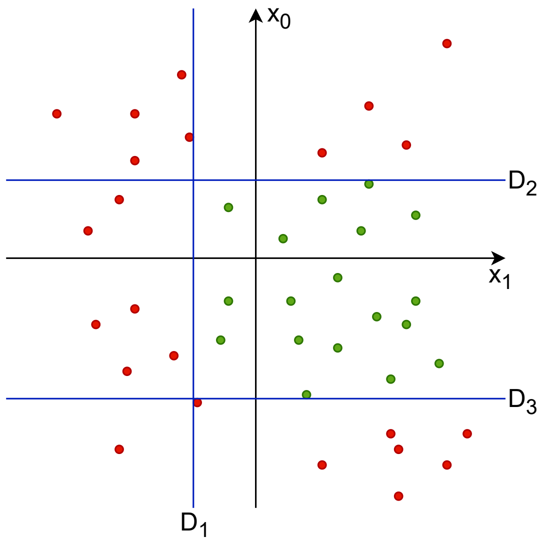


Figure 5.6.: Distribution of vectors that are not linearly separable and their seperation by multiple decisions.

The depth of the tree structure created by DTC can be limited in advance with a fixed value. If this is not the case, then the depth is variable and expands until all the leaves contain vectors of only one emotion, unless it reaches the number of vectors at which the set is no longer separated [64]. If the depth is fixed in advance, it may happen that vectors of different emotions are contained in one leaf. In this case, DTC, has chosen less good conditions in the decision nodes for distinguishing the features of a vector. It may also happen that certain emotions in a multidimensional space overlap in regions because some features of different emotions are similar. This can happen not only if the depth is too shallow, but also because possibly the tree structure is too complex and overfitting to the training data occurs. Another issue is the data itself, it could contain noise or similar. DTC is a greedy algorithm because it takes the best value at each time. As a result, all further separations of the dataset, depend on the decisions of the previous nodes. And given this, it is impossible to guarantee that DTC will provide the optimal set of decisions. In addition, DTC has no back propagation to modify the decisions of the previous nodes. These are reasons why DTC is performant. Because at any time, DTC takes the best value of information gain to set the corresponding decision for a node. And it doesn't change any decisions that have already been set.

6. Neural Networks

6.1. Deep Neural Networks

Deep Neural Networks (DNN) is a topic of much research these days [17, 22, 37]. DNN is a subcategory of machine learning [36, 72]. These are inspired by the structure of the human brain. A DNN is characterized by the fact that it does not need features of emotions extracted beforehand, as is the case with machine learning. This is because a DNN is presented with images as input, from which it derives features itself, without the use of human intervention. For training a DNN model, a huge amount of data is needed. [72]. These images referred to in this work are mel spectrograms, as presented in section 4.2. These are two-dimensional images, which is equivalent to a two-dimensional matrix. This two-dimensional matrix must be converted to a one-dimensional vector for the input layer of the neural network. This must be done because the value of each pixel is the input of a neuron in the input layer, as shown in figure 6.1. Thus, the number of neurons in the input layer must be equal to the number of pixels in the mel spectrogram. Because the number of neurons in the input layer is fixed, each image that is used as input must be the same size. Often the audio files have a different playback time and for this reason, they have to be resized to the same size using the zero padding technique, which is also part of this work and used in the experiments.

The most important component of a DNN are the neurons. These are responsible for information processing. A linear composition of neurons corresponds to a layer. In principle, a DNN consists of an input layer, several hidden layers and an output layer. The number of hidden layers must be greater than two, so that the DNN is not a simple neural network [36, 72]. The number of neurons in the output layer corresponds to the number of emotions that the DNN should classify. The rule is that all neurons in the previous layer are connected to every neuron in the following layer. Each individual connection is weighted. The input of a neuron of the next layer is made up of the sum of the product of each value of a previous neuron with the weight of the connection to the following neuron. The value of the following neuron, the bias, is then added to this sum. The formula for this is mentioned in the work of Svozil et al. [66] and is presented here in customized form.

$$y_j = B_j + \sum_{i=1}^{n} x_i \cdot g_{ij} \tag{6.1}$$

j is the index of the neuron of the following layer for which the input is calculated using equation 6.1. y_j is the value of the input to the neuron with index j. n represents the number of neurons of the previous layer, where they are labeled with index $i \in [1, n]$. The value of the output from a neuron with index i is denoted by x_i. g_{ij} is the value of the

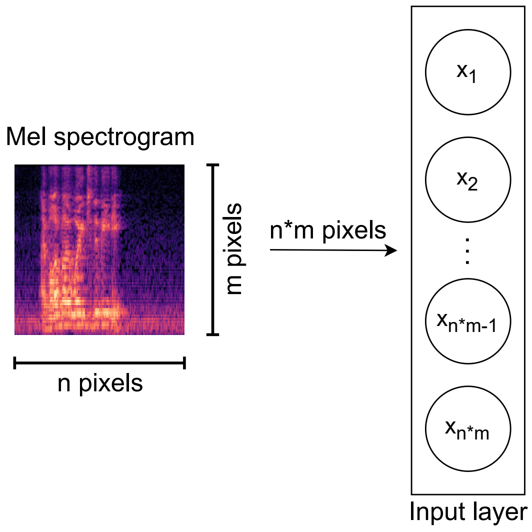


Figure 6.1.: A mel spectrogram flatten to a one dimensional input vector.

weight of the connection between a neuron with index i and the neuron with index j. B_j is the value of the neuron with index j, this value is also called bias.
The value of the input of a neuron in the hidden layer is calculated with equation 6.1. This value is then processed by an activation function. The result of such an activation function determines whether the neuron is deactivated or activated. Activated neurons pass data to the neurons of the following layer, all the way to the output layer. The neuron with the highest value in the output layer decides the emotion assigned to the mel spectrogram. This described process, as also shown in figure 6.2, is called *forward propagation* [28, 47].

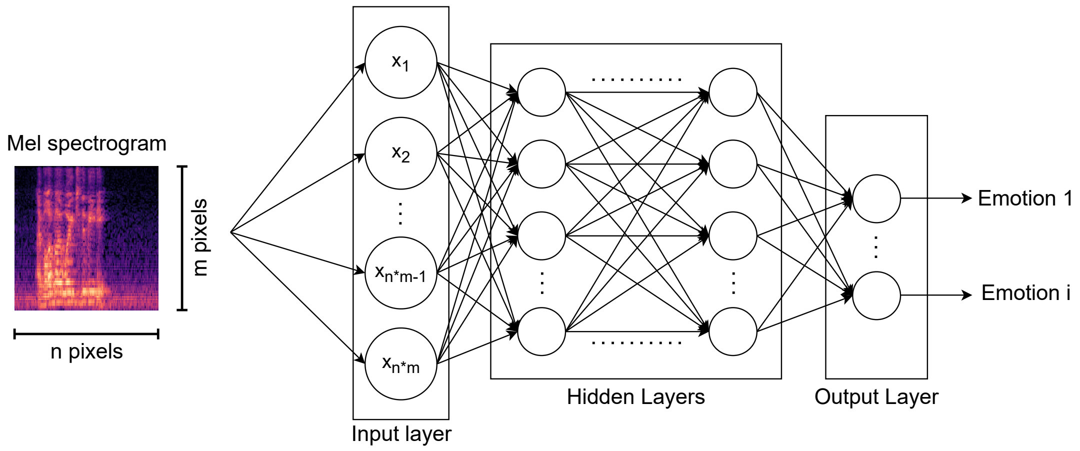


Figure 6.2.: Mel spectrogram as input to a neural network architecture.

Now let's take a closer look at activation functions, specifically the ReLU function because this is used in the experiments of this work and is generally a very widely used activation function [65]. As mentioned earlier, the input of a neuron is fed into an activation function. This then decides whether that neuron is deactivated, or activated. The ReLU function sets all negative values to zero. All positive values remain unchanged. This is achieved with the equation $f(x) = max(0, x)$ [65]. This can be seen in figure 6.3. In the ReLU function, all neurons that have a value of zero are deactivated. All other neurons with their unchanged positive values remain activated.

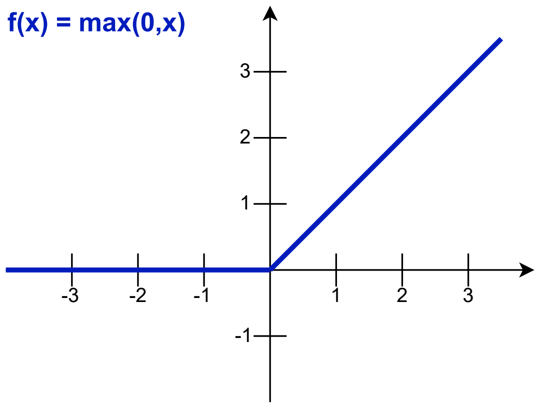


Figure 6.3.: Visualisation of the ReLU activation function.

After the forward propagation, the error is calculated, therefore the result of the prediction of the DNN is compared with the actual value. Now an unclarity arises here. Emotions, such as joy and fun, have no value. Due to this because the DNN works with numbers, it is necessary to assign a natural number $i \in \mathbf{N}$ starting with the value zero and increasing by one to all emotions that should be classified. After comparing the predicted and the actual value, an error is calculated.
Before the calculation of the error is discussed further, the softmax function is discussed first, because it is used in this work to create another layer after the output layer. This is done to normalize the values of the output layer. By applying softmax to the values of the neurons of the output layer, for each emotion, a value between zero and one is obtained. Adding all the values after applying softmax results in one. Accordingly, the values of the softmax layer, can be seen as predicted probabilities for the classification of the emotions. Figure 6.4 illustrates visually the application of the equation 6.2 to the output values of the output layer neurons. It is also recognizable that at the end, the predicted probabilities $p_{Emotion_j}$ for the classification of the corresponding emotion are obtained. The following formula is used to calculate softmax [65, 71].

$$Softmax(\vec{o})_i = \frac{e^{o_i}}{\sum_{j=1}^{n} e^{o_j}} \tag{6.2}$$

The equation 6.2 calculates for the value of the neuron, with index i, the possible probability $Softmax(\vec{o})_i$, which the DNN assumes to be emotion i. $\vec{o}$ denotes the vector with the values of the output layer. i is the index of the neuron for whose value softmax is applied. The index $j \in [1, n]$ goes over all neurons of the output layer. n represents the number of neurons in the output layer. Softmax is calculated for all values of the output layer, and the emotion corresponding to the highest value is presented by the DNN as the result.

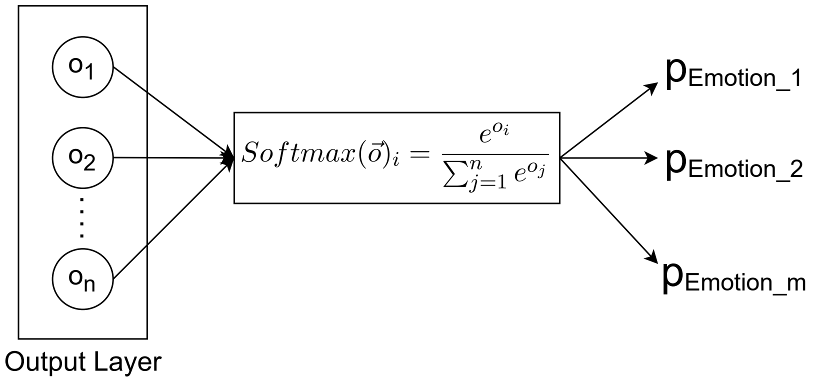


Figure 6.4.: Applying Softmax to the output vector for the predicted probabilities of emotions.

After obtaining the results of the output layer and applying softmax, the error is calculated. For this, the predicted emotion is compared with the actual emotion. In this work, a closer look is taken at the *cross entropy loss function* and the *focal loss function* for the calculation of the error. Cross entropy is often used when softmax was applied before and values between zero and one are left. The original equation of cross entropy is thus presented [32, 69].

$$CE(p_i) = -\sum_{i=1}^{n} y_i \cdot log(p_i) \tag{6.3}$$

The equation 6.3 is the original form of how to calculate the cross entropy $CE(p_i)$ of the predicted probability p_i of an emotion i. The sum is performed for all emotions from one to n. Here, y_i describes the actual emotion i that the DNN is trying to classify. This equation has a weakness because it performs unnecessary computations. The reason is that in this work only emotions are considered. The actual emotion always returns a value of one, and all others return a value of zero. Multiplication by zero can be avoided because the result is always zero, so the equation 6.3 is replaced by the following [43].

$$CE(p_i) = -log(p_i) \tag{6.4}$$

With this equation, we get the same effect. Here we calculate the cross entropy error $CE(p_i)$ between the predicted and the actual emotion. Here p_i with $i \in [0, 1]$ stands for the predicted probability to be emotion i. The cross entropy can be additionally weighted if the dataset has a non-equivalent distribution of emotions [43].

$$CE(p_i) = -\alpha_i \cdot log(p_i) \tag{6.5}$$

With this, a weight α_i is multiplied on the error of an emotion i.
However, the cross entropy function has a problem. If the data in the dataset are not balanced, the calculated error of a specific emotion may be high because only few data represent this emotion. The error is large, but unfortunately does not have a large effect on the overall error if other emotions are represented in high numbers in the dataset. If the DNN can classify them well, then they will lead to a lower error. But the problem lies in the amount of data that is difficult and easy to classify. For example, if we have a 1:10000 ratio of the occurrence of one emotion to another emotion in the dataset, then the error of the emotion with the low occurrence in the dataset is not critical enough. In addition to this, the low represented emotion may also be harder to classify. This can be seen in figure 6.5. The red graph in the figure maps the cross entropy function. It can be seen that when the probability of classification is less than 50%, the error becomes larger and larger. Nevertheless, this does not matter much if most mel spectrograms have a probability of classification above 50% because these have a much more decisive total error. This problem is to be bypassed with the focal loss function [43]. The focal loss function is based on the equation 6.4 of the cross entropy and shall counteract this problem [43].

$$FL(p_i) = -(1 - p_i)^{\gamma} \cdot log(p_i) \tag{6.6}$$

$FL(p_i)$ stands here for the result of the calculation of the focal loss function with the predicted probability p_i that it is emotion i. The γ parameter decides how close the cross entropy function is pushed to the origin. A γ value of zero, corresponds to the cross entropy function from the equation 6.4. A higher value of γ results in counteracting the problem of unevenly represented emotions in the dataset and emotions that are easy and difficult to classify. Thus, the calculated errors of the much represented emotions have a lower impact if the classification turns out to be correct. And the errors of the low represented emotions have a higher impact on the total error if they are classified incorrectly. This can be seen in figure 6.5. The green and blue graphs in the figure represent respectively the focal loss function with different values for γ. The higher the chosen value for γ, the less meaningful the high represented emotions are on the total error. Again, weighting can be added to the equation 6.6 of the focal loss calculation to further counteract the uneven distribution of emotions [43].

$$FL(p_i) = -\alpha_i(1 - p_i)^{\gamma} \cdot log(p_i) \tag{6.7}$$

Here, as already in the equation 6.5 of the cross entropy with weighting, a weight α_i of the emotion i was additionally multiplied to it. In the experiments, the two loss functions 6.5 and 6.7 are used for the calculations of the weighted error.
Following the forward propagation and the calculation of the total error, the *back propagation* is performed. In this process, the calculated total error and an optimization function are used to optimize the weights and biases of the neurons in a DNN. These values are

optimized so that the calculated total error is lower than before the back propagation. The optimization function tries to find the minimum of the total error. For this purpose, such functions use gradients to determine in which direction the slope of the error goes, thus the value of the calculated total error decreases. In this direction, the optimization function takes a step with a certain step size. The step size of the optimization function also depends on the learning rate. This is multiplied by the calculated gradient of each neuron. In the experiments of this work, AdamW is used as the optimization function. AdamW investigates for the further gradients to determine whether they differ little from each other, or strongly. Accordingly, AdamW adjusts the step size of the optimization because these are not a fixed value in AdamW, but vary [45]. If the gradients differ rapidly, then AdamW takes small steps toward lowering the error. If the gradients differ only slightly, then AdamW makes a large step [31]. Thus, the global minimum of the total error can be found more quickly.

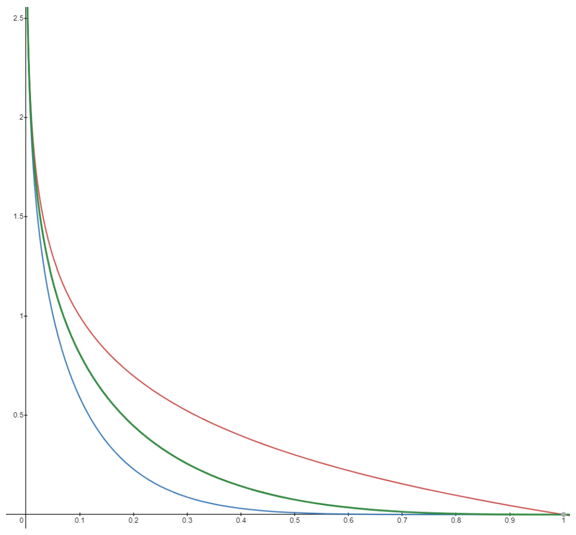


Figure 6.5.: Red graph is the cross entropy function. Green and blue graphs are the focal loss with $\gamma = 2$ and $\gamma = 5$.

The whole training process of a DNN consists of the forward propagation, the calculation of the total error and the back propagation, the optimization of the weights of the connections between the neurons and their bias values. This combined process is performed for each batch. A batch is a pre-determined number of data from the dataset that is given to the DNN in one go. There is one case where the size of a batch is different from all others. This concerns the last batch. If all other batches have already taken data from the dataset and there is less data left than the batch size, then the last batch has less data. If the last batch has only one date, then this can be a problem because there are components that need at least two dates to work with. For example, this affects the component that is responsible for batch normalization [10]. For this reason, when loading a dataset, it is possible to drop the last batch that is not full [14], thus ensuring that all batches have the same amount of data. Once all batches have gone through the training process, this is

called an epoch. As a general observation, multiple epochs lead to better performance on classifications of a DNN. Nevertheless, overfitting of the DNN can appear. For this, after each epoch, the model can be saved with its weights and biases, as done in experiments in this work, or early stopping can be used [5]. Early stopping aborts the training phase when a model's results start to degrade.
There are many components that can be used to build a DNN. Thus, the ones that are used in the experiments will be discussed.
Linear [12] is one of the most important components in building a DNN. It creates a layer of neurons. It is mandatory to specify the parameters, input size and output size. The number of neurons that are initialized corresponds to the output size. The input size corresponds to the number of incoming connections of a neuron. This component performs a linear transformation of the incoming data [12]. This corresponds to the equation 6.1.
ReLU is a component that can be placed after a linear component. This provides the execution of the activation function ReLU, as discussed earlier, on the incoming data of a neuron.
Dropout is a component that deactivates neurons in a layer at a pre-determined probability [11, 52]. These neurons are not utilized during forward propagation and their weights are not changed during the back propagation optimization process either. These neurons are only temporarily deleted. After training with one batch, deleted neurons are restored and before starting training with the next batch, random neurons of a layer to which dropout is applied are temporarily deleted again. One might think that it would be disadvantageous for training a DNN to apply dropout. As discussed in the work of Nielsen [52], it is possible to think of training multiple neural networks because the internal structure of the neural network changes at each training phase. This has as an advantage that the other neurons of the layer that are not deleted have to compensate for the deleted neurons, and thus the neural network learns in a more generalized way. Another advantage is that it avoids overfitting to the training data.
Batch norm [10] is a component that takes care of the normalization of the batch. This component increases the speed of the training phase [35]. For this purpose, the output values of the neurons of a layer are normalized to be within a certain scale. Thus, when using batch normalization, the DNN ensures that the data, when passing through the neural network, is within a scale. This component can be placed after each layer to ensure that the values remain normalized and do not make large jumps. In the Mel spectrograms used, the time unit is much smaller as opposed to the frequencies. Thus, the optimization function may have trouble making large steps because the time unit is smaller in contrast to the frequency. Normalization brings these two different scales to a common scale. Thus, it is possible that the optimization function can make larger steps towards the global minimum of the error [35].
As mentioned earlier, the training process consists of forward propagation, the computation of the error when comparing the predicted and the actual emotion, and back propagation where the weights of the connections between the neurons and their biases are adjusted to lower the error. In the following, the pipeline of forward propagation is described step by step as it is applied to the experiments.

1. Initialize scheduler that adjusts the learning rate.
2. Start of the epoch.
3. Batches are created.
4. All gradients are set to zero.
5. Data is loaded into the neural network.
6. The predicted emotions of the DNN, are compared with the actual emotions. The total error of the batch is calculated.
7. Gradient with respect to the error is calculated for each neuron.
8. Each parameter is recalculated with application of the corresponding calculated gradient.
9. Scheduler is increased by one.

This is the pipeline of the training phase for one batch of the epoch. This must be repeated for all batches, and then another epoch can start. The scheduler that is initialized ensures that the learning rate decreases by a certain value after a specified number of epochs. After each epoch, the DNN is validated. In the validation phase, a part of the dataset not previously presented to the DNN is used. The validation is intended to provide information about the accuracy and distribution of the classifications, so that we as humans can judge how well the DNN can classify emotions. For this, the *accuracy* and the *recall (macro)* are used as results of the validation. The accuracy can be calculated with the formula $Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Negatives}$ [29]. The model has an accuracy of 100% if everything was classified correctly. The recall is calculated with the formula $Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$ [30]. The model has a recall of 100% when it has correctly recognized all emotions. The macro recall is calculated with $Recall\ (macro) = \frac{Recall_1 + Recall_2 + ... + Recall_n}{n}$ [68]. Where n is the number of recalls. Thus, the macro recall gives us the average of correctly recognized emotions. A high macro recall indicates that the model can distinguish the different emotions well. In the following, the pipeline for the validation process is presented.

1. Batch of the validation dataset is created.
2. Data is loaded into the neural network.
3. The accuracy, between the actual emotions and those predicted by the DNN, is calculated.
4. The macro recall, between the actual emotions and those predicted by the DNN, is calculated.

At the end of each epoch, the validation process is performed and provides information about the accuracy and the macro recall. Considering this, it is important to split the dataset in advance. In this work, a value of 10% of the whole dataset is taken for the validation.

6.2. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are used in the field of image recognition. As reported earlier, the experiments in this work use mel spectrograms, as discussed in section 4.2, as input to a neural network. CNNs are good at recognizing patterns in images that contribute to the classification of an emotion. Moreover, linear layers with many neurons are not necessary because in CNN, the input data is processed in advance and, accordingly, when the data is passed to a linear layer, it is smaller and, therefore, fewer neurons are needed. Furthermore, the two-dimensional matrix behind the mel spectrogram does not need to be converted into a one-dimensional vector beforehand because a convolutional layer can take the image in its original form as input. Due to pattern recognition, this work particularly relies on the use of CNN because emotions can have differences from person to person and static input data like handcrafted features could have a problem with it. In contrast, patterns can help to overcome this effect partially. Figure 6.6 presents how a Mel spectrogram is fed as input to the first linear layer of a DNN. Here, each pixel corresponds to a value in the input vector. For large images with many pixels, one needs accordingly many neurons in the input layer. This leads to an increased computational effort. In addition, each pixel is interpreted separately by the DNN. A CNN can capture the dependencies from one pixel to neighboring pixels.

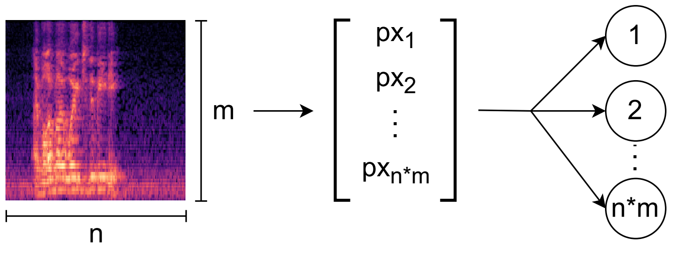


Figure 6.6.: Each pixel of the mel spectrogram corresponds to one value in the input vector.

A *Convolution Layer* has a so called filter, or kernel, as its component [1, 21, 41, 54]. This is a matrix that is initialized in advance with a fixed size. This filter, starting in the top left corner, moves over the image of a mel spectrogram as shown in Figure 6.7. The matrix of the filter contains a value at each position. Using these, the filter performs the calculation of the Hadamard product [33] with the underlying pixels of the mel spectrogram. All values of the resulting matrix are subsequently summed. This value is then stored in the output matrix of the convolutional layer. This is also called *feature map* [21, 41] or *activation map* [54]. After computing a value for the feature map, the filter, moves on by a certain number of pixels on the mel spectrogram. This is called *stride* [1, 21, 41, 54]. This value must be set when creating a convolutional layer. It can also be a tuple, in which case the first value corresponds to how many pixels moving to the right and the second to how many pixels moving down. The direction of the filter is always from left to right. Arriving at the end of the mel spectrogram, the filter jumps to the left side of the mel spectrogram and shifts its position downward by the number of pixels corresponding to the stride value. In addition to the stride, the filter of a convolutional layer can be given padding as an attribute [1, 21, 41, 54]. This is a value or tuple that determines how many pixels are added around the mel spectrogram. Each of these added pixels represents a value of zero. After the feature map is fully computed, this is the output of the convolutional layer. The matrix of the feature map is smaller in dimensions than the matrix of the input mel spectrogram. Of course, it has to be considered how many dimensions the image has in depth, by this is meant if it is a colored image (RGB) and accordingly has 3 dimensions for the colors. Nevertheless, the application is equivalent because the filter has the same dimension in depth as the input of the convolutional layer. The filter calculates the value for each dimension separately and adds them up in the end and adds the value to the feature map. A convolutional layer can have more than one filter. For each filter, a feature map is created, which together are used as output. The goal of the convolutional layer is the recognition of patterns and the resulting extraction of features and the reduction of the input for the upcoming linear layers of the neural network. On top of the output matrix of the convolutional layer, an activation function can still be placed, such as the ReLU activation function as discussed earlier in section 6.1.

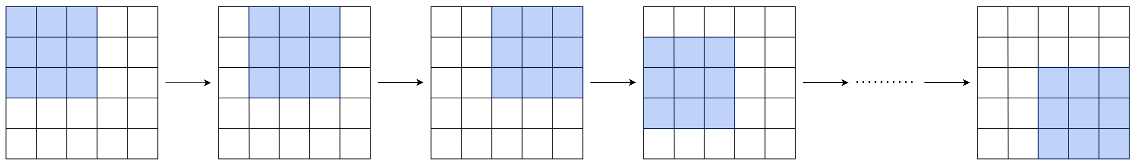


Figure 6.7.: Movement of the filter with stride equals to one.

Another thing is used mostly in the connection, to the convolutional layer and the possible activation function following it. This is about *pooling* [1, 21, 41, 54]. A distinction is made between *max pooling* and *average pooling* [21, 41]. Here, a surface, with a predefined size, moves over the calculated feature map of the convolutional layer. It has similar properties as the filter of the convolutional layer because this area moves like the filter and the stride parameter can be given to it. The max-pooling layer performs the max operation on the values located below the surface. Thus, the maximum value is taken and written to the output matrix of the max-pooling layer. Similarly, the average-pooling layer calculates the average of the values that are below the surface. After performing pooling on the whole feature map, the output is a matrix smaller in dimensions. This reduces the computational effort for the next layers. Here, the max-pooling layer is the one that is often used in practice [21] and an example of max pooling is presented in the figure 6.8. However, when creating a max-pooling layer, attention must be paid to how the size and value of the stride is set, as information can be lost or corrupted [54].
It is possible to use several convolutional layers and several pooling layers. After the convolutional layers and the pooling layers, the output is given to a neural network with linear layers to classify the emotions. For this, an operation must be performed beforehand so that the matrix is transformed into a vector. In the experiments of this work,

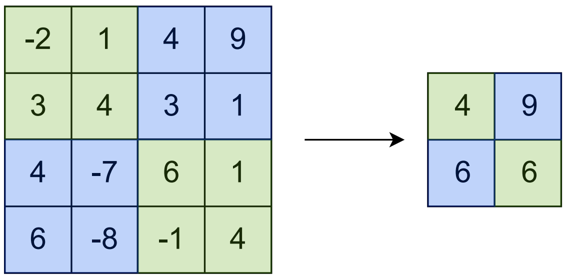


Figure 6.8.: Application of max pooling on the feature map.

there are two possible approaches used for this. Once, the component *flatten* [13], which combines the values of the different dimensions into one dimension. This gives a vector as a result, which can be used as input for the first linear layer. And the other way is to use the component *adaptive average-pooling* [9]. This component actually does the job of creating an average-pooling layer, but unlike what was discussed, this component uses as a parameter what dimension the output should have. Thus, this component automatically adjusts the area of the pooling so that the output has the required dimension. If this parameter is set to one, the output will be a vector of dimension one, which can be used as input for the first linear layer. However, with this method it is important to note that the output matrix from the layers before the first linear layer is reduced in dimension because with average-pooling the average of the values of the pixels below the area of the pool is calculated, as already described.
After extracting features from the mel spectrogram by the convoutional layers and transforming them into a one dimensional vector, classification of an emotion can take place from linear layers as discussed in section 6.1.

6.3. Transfer Learning

Transfer Learning describes the process of using an already trained model and adding some components to it. This can be helpful if these models follow a similar training strategy or were trained with a similar dataset. In addition, the advantage is that the model is already trained and the computational effort is reduced by not having to optimize many parameters. In advance, it is also necessary to obtain information about what exactly the model can classify, what the input of the model needs to look like and how well the model performs. This work deals with models for image classification, since in the experiments CNN are trained with mel spectrograms. Thus, a model is used that can already recognize patterns in images.
An already trained model still needs to be adjusted to the intended use. The output of the trained model often does not match the number of emotions to be classified. Accordingly, it is necessary to add a linear layer, which has a set of neurons corresponding to the size of the output. Of course, it is possible to build a more complex DNN according to the already trained model, that has more chance to adapt parameters for the specific

application domain.
Another possibility is to freeze only a certain percentage of the already trained model, and a part experiences an optimization of the weights. Thus, the whole model has one more chance to optimize the parameters in such a way that it gives a better result in the classification of emotions. However, this still increases the computational cost because more parameters need to be trained.
One thing to watch out for when using already trained models is that they expect a certain input size. This means that the data has to be adjusted to fit that model. The trained models, used for transfer learning in the experiments, require three dimensions for color because these models were trained with RGB images. The problem is that a mel spectrogram has only one dimension for the color. Thus, it is necessary to artificially extend the mel spectrograms to three dimensions for the color. This is done in the experiments by copying the values of the original dimension for each dimension of a color.

7. End-to-end Deep Learning Framework for acoustic emotion recognition

The framework presented in this paper is designed to classify an emotion based only on an audio track. Depending on the use of a dataset, this framework can recognize a certain number of emotions. In doing so, the framework handles the entire process after receiving an audio track. It processes the audio track into a matching mel spectrogram. This serves as input to the trained model. After classification, by the trained model, the framework returns the predicted emotion. This is presented in figure 7.1.

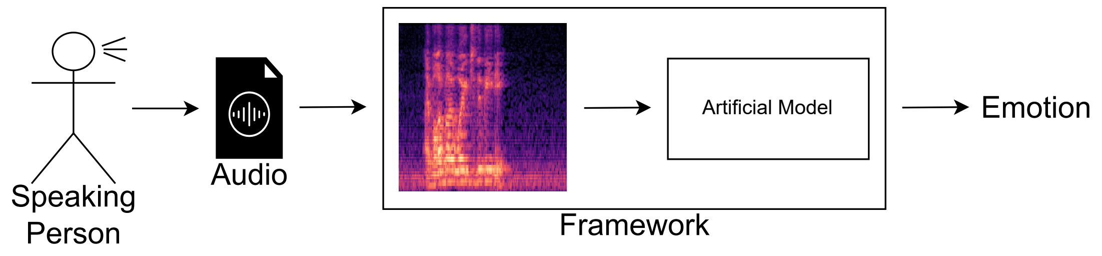


Figure 7.1.: Pipeline of the framework.

A working framework needs two components. Once, the preprocessing of the received audio file into a mel spectrogram. And it needs a model that has been trained to classify emotions. Such a model should have a certain accuracy in classifying emotions. Therefore, several methods are used in training neural networks in order to use the model with the best result in terms of accuracy and macro recall for the framework.
Another thing that the framework has to take care of is the variable length of the audio tracks that the framework receives as input. This is because the trained model only accepts a single size as input. The framework takes care of this by checking the size of the audio track and the associated mel spectrogram in advance. If it is too small for the trained model, it is resized to the desired size. To achieve this, the audio track is extended by the missing time with a mute sound. If the audio track is too long, chunking is used to split the audio file into several chunks of fixed size. After splitting, an emotion is classified for each chunk. When doing so, the output of the framework, is the most predicted emotion of all the chunks. One advantage of doing this is that, over the course of a conversation, one can repeatedly pass smaller audio snippets to the framework to have an emotion classified without having to use zero padding if each snippet corresponds to the size of the chunk. This option for the framework is an excellent choice, considering that emotions can change quickly.

Due to the possible sizes of chunks, several sizes were tried in the experiments. In addition, the experiments look to see if chunking gives better results with variable audio lengths. Chunks allow us to take smaller input sizes, yet the issue here is that with longer audio tracks there could be a tie in the occurrence of different emotion. The framework must also take an action for this. If such a case occurs, then the framework outputs that no emotion was classified. If the experiments indicate that the results are better with the use of zero padding, then the framework must be restructured in which it brings each incoming audio track to the desired length. If the incoming audio file is longer, the framework can either choose the area of the audio track with the most frequency occurrence, or it simply takes the exact center of the audio track each time and computes a mel spectrogram from it, which is then classified by the model in the framework. The problem with this option is that information could be lost if only a fixed interval of the longer audio track is used.

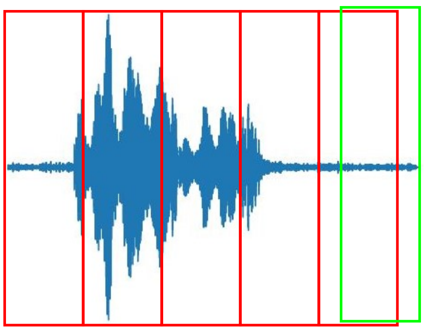


Figure 7.2.: Chunking of an audio file.

In the following is explained how to work with the method of chunks and how to prepare the data for training a neural network. The size of the chunk is chosen in advance and therefore it is fixed. That means that the audio track will be divided into several parts. For this, we use equation 7.1. After that, all audio tracks shorter than the size of the chunk are filtered out. And after that, the audio track is split into the set of chunks calculated with the equation 7.1. In the process, the last chunk may stick out over the end of the audio track. The end of this chunk is placed at the end of the audio track. Thus, the last two chunks overlap, as shown in figure 7.2, where the chunk with the green frame is the last chunk that leads to the overlap.

$$n = \left\lceil \frac{t_a}{t_c} \right\rceil \tag{7.1}$$

In the equation 7.1, n denotes the number of chunks, t_a the length of the audio file in seconds, and t_c the size of the chunk in seconds.
Besides filtering out audio tracks that are shorter than the size of the chunk, there is also the option of allowing audio tracks above a certain size. These audio tracks are then extended to the length of the chunk size with a mute sound. This silent sound is added at the end of the audio file. Thus, these can also be included in the training dataset. This is because they are effectively split into one chunk. With this method, the neural network gets familiar with silent tones.

8. Experiments

All the preprocessing of the data and the execution of the experiments was done on one system. The most relevant components are the processor. This is an Intel® Core^{TM} i7-10700K. The graphics card, which was used for training the neural networks, is an Nvidia GeFroce RTX 3080. The system memory is 32 GB. Python is the programming language used for development. Python packages that were used are, scikit-learn [57], for the application of SVM, MLP and DTC. Librosa [50] for the processing of the audio files and generation of the mel spectrograms and PyTorch [56] for the development of the neural networks.

8.1. Datasets used in this work

In the experiments of this thesis, a total of three different datasets are used. Each dataset consists of audio files, each of them already having an emotion associated with it. The datasets are processed differently, have different numbers of audio files, and also differ in the number of emotions represented in each dataset.
Experiments are performed on the dataset *MSP-Podcast Corpus* [46]. The data was collected from podcasts. The MSP podcast dataset has 73042 audio files at the time of writing this work. This dataset contains ten different labels for emotions. These consist of angry, sad, happy, surprise, fear, disgust, contempt, neutral, other, and no agreement. The emotions were assigned using crowd-sourcing [46]. In this sense, it corresponds to a natural dataset as discussed in section 3.2. The language of the audio files is English.
Another dataset which is used is *ASVP-ESD* [39]. The ASVP-ESD dataset has 12625 audio files at the time of writing this work. This dataset contains 13 different labels for emotions. These consist of boredom, neutral, happy, sad, angry, fearful, disgust, surprised, excited, pleasure, pain, disappointment, and breath. The first, 5105 audio files were annotated by five people according to their feeling. These assigned emotions were then voted. The emotion with the most votes was assigned to the audio file. In the event of a tie, a random emotion from this was assigned to the audio file [39]. The emotion assignment for the later added audio files was using the same procedure, but this time from three people. The data was collected from movies, TV shows, YouTube videos, and from other websites [39]. In this sense, it corresponds to the category of semi-structured datasets as discussed in section 3.2. This is because some of the emotions are real, but it was taken in more of an artificially created emotional situation. The language of the audio files ranges from Chinese, English, French, Russian and other languages [15].
As the last dataset *CREMA-D* [8] was used. The CREMA-D dataset has 7442 audio files at the time of writing this work. This dataset contains six different labels for emotions.

These consist of anger, disgust, fear, happy, neutral, and sad. The emotions were played by experienced speakers. The audios were then classified by participants according to emotion and their strength. This is because even when a piece is recorded, it is possible to tell if the right emotion was played. These audios were recorded in a planned manner and sentences were performed by experienced speakers. In this sense, this dataset corresponds to the category of simulated datasets as discussed in section 3.2. The language used for the audio files is English.

8.2. Preprocessing of the datasets

All three datasets must be preprocessed in different ways. This is because each dataset follows a different architecture in compiling the data. Because of the differences, the preprocessing of each dataset is discussed separately below.
First, we discuss the preprocessing of the MSP-Podcast dataset. For this dataset, the labels of the emotions of all audio files are in a separate JSON file. From this JSON file, the individual emotions were each extracted into a CSV file. Each CSV file has the same name as the corresponding audio file. This aspect is important for the later extraction of the features and when creating the mel spectrograms. Because of the enormous amount of audio files, only a part of the audio files were used for the experiments so that the experiments do not require too much time. Therefore, the audio files were filtered out, which correspond to the label of the emotion 'other' and 'no agreement'. This is because these do not provide any meaningful information for the desired model, which should only classify emotions. In addition, all audio files that exceed the length of six seconds were removed. Due to the differences between the number of different emotions, the remaining audio files were reduced again. For each of the eight emotions, 1000 audio files were left, and thus the MSP-Podcast dataset contains a total of 8000 audio files after preprocessing. This was chosen to be similar to the amount of audio files in the CREMA-D dataset. Thus, the training time of a neural network is reduced and thus more results are obtained in a shorter time. After preprocessing the MSP-Podcast dataset, the longest playback time of an audio file is 6 seconds and the shortest, rounded to two decimal places, is 1.12 seconds. The average playback duration, rounded to two decimal places, is 4.2 seconds. The distribution of emotions can be seen in the figure 8.1.
Now for the preprocessing of the ASVP-ESD dataset. Each audio file follows a naming scheme. In this scheme, a number at a certain position in the name corresponds to a certain value of a category. First, all audio files were filtered out that correspond to the label of the emotion 'breath'. Therefore, all audio files are identified, which contain the value '13' in the third position of the filename. This is because this does not appear directly as an emotion. In addition, all audio files corresponding to the Chinese language were filtered out in the same step. Therefore, all files are identified, which have the value '01' at the ninth place in the name. After preprocessing the ASVP-ESD dataset, 3079 audio files remain. After preprocessing the ASVP-ESD dataset, the longest playback duration, rounded to two decimal places, is 32.24 seconds and the shortest is 0.28 seconds. This

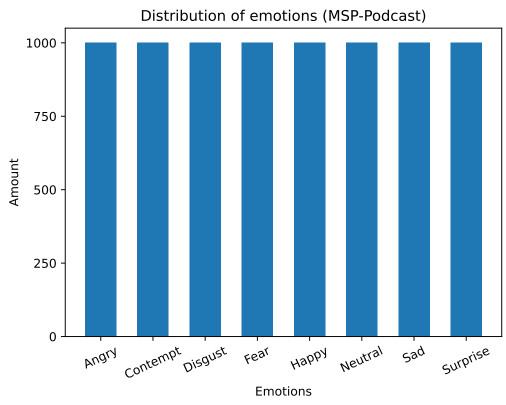


Figure 8.1.: Distribution of emotions of the preprocessed MSP-Podcast dataset.

difference in playback duration results in very short audio files being padded to the length of the largest audio file using the zero padding method. The average playback duration, rounded to two decimal places, is 6.11 seconds. The distribution of emotions can be seen in the figure 8.2.

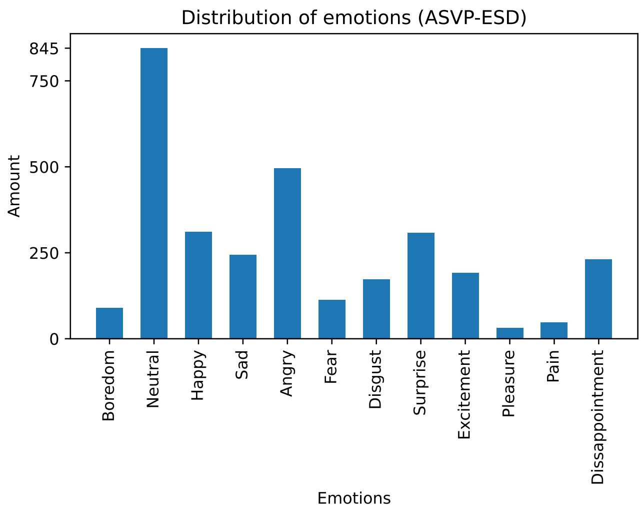


Figure 8.2.: Distribution of emotions of the preprocessed ASVP-ESD dataset.

For the CREMA-D dataset, there was no preprocessing in terms of reducing the dataset because this dataset contains only, 7442 audio files. The number of different emotions is additionally reasonably balanced. Similar to the ASVP-ESD dataset, the names of the audio files of the CREMA-D dataset are also divided into segments. Here, the third segment of each audio file contains the identifier for the emotion. After preprocessing the CREMA-D dataset, the longest playback duration, rounded to two decimal places, is 5.01

seconds and the shortest is 1.27 seconds. The average playback duration, rounded to two decimal places, is 2.54 seconds. The distribution of emotions can be seen in the figure 8.3.

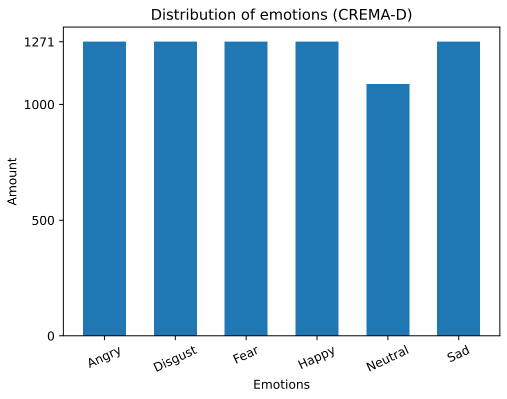


Figure 8.3.: Distribution of emotions of the preprocessed CREMA-D dataset.

For running the machine learning algorithms as discussed in section 5, extracted features are needed from the audio files as discussed in section 4.1. For this purpose, the experiments use the toolkit *openSMILE* [24]. Explicitly, *ComParE 2016* [60] is used as the feature set. Here, *Functionals* was used for the feature level. This will extract 6373 features from each audio file [2] which will be useful for classifying emotions. The extracted features for each audio file were saved into a CSV file for further processing. The same name of the associated audio file was used for each CSV file, except for the MSP-Podcast dataset. For naming the files for the extracted features of the audio files of the MSP-Podcast dataset, the corresponding emotion was additionally added to the end of the filename from the corresponding CSV files. This simplifies the further procedure because the emotion does not have to be additionally taken from the related CSV file when training. When creating each CSV file, the columns 'file', 'start' and 'end' were discarded before saving, so that only the features in each file are retained. Further, for each dataset, the files containing the features were split into a training dataset and a validation dataset. In this process, 90% of the files for training and 10% of the files for validation are split randomly. After this split, all individual CSV files for the training dataset and the validation dataset are concatenated into a single CSV file. These are used for training and validation.
For performing the neural network experiments in this work, the Python package *librosa* [50] is used. For this, each audio file is iterated over and it is loaded by librosa each time. When loading, the audio file is obtained in digital form as an array and the sampling rate. After that, the mel spectrogram is calculated with librosa. Here 512 is taken for the length of the FFT window as discussed in section 4.2. This value is taken because it is written in [42] that this value is usually used for speech processing. For the spacing between FFT segments, 256 is taken, which is half the value of the length of a single FFT.

This results in having an overlap, which prevents information from being lost. For the number of mel filter bands, as discussed in section 4.2, the default value of librosa is used, which equals 128. After the mel spectrogram is calculated, it is saved into a TXT file for further processing. In each case for the filename, the name of the associated audio file is selected, with exception for the files of the data of the MSP-Podcast dataset. For this, as before for the features, the emotion is taken from the corresponding CSV files and added to the name of the TXT file of the mel spectrogram.
For chunking, as discussed in chapter 7, two different sizes of chunks are tried in the experiments. Once the size is two seconds and once it is four seconds. If the size of the chunks is two seconds, all audio files that have a duration shorter than two seconds are filtered out, the same is done for the size of the chunks of four seconds. Again, each audio file in a dataset is iterated over and each of these are loaded with librosa. With the playback length of the audio file t_a determined in seconds, the equation 7.1 is then used to determine how many chunks are generated for each audio file. For this, the size of the chunk t_c is used, which is two and four seconds. After that, the mel spectrogram is calculated and saved for each chunk. When saving, the value of the index of the chunk is added to the end of the filename. For the last chunk, the mel spectrogram is always created from the end of the audio file and dragged with the size of the chunk towards the beginning of the audio file, as discussed in chapter 7. Thus, in most cases, there is an overlap of the last two chunks of an audio file. The unique feature of a chunk size of four seconds is that all audio files whose playback length is longer than two seconds but shorter than four seconds are processed before a mel spectrogram is computed. A mute sound is added to the end of the affected audio files until they contain a playback length of four seconds, which in this constellation corresponds to exactly one chunk. Thus, these are also included in the dataset.

8.3. Machine Learning Algorithms

This section presents the results of applying the three machine learning algorithms presented in 5. These machine learning algorithms use as input the features of an audio file that were extracted using openSMILE, as discussed in section 8.2. For now, these results should only provide a baseline for the further experiments. The goal here is that the results, with the application of DNN, will be better in the experiments.

8.3.1. Hyperparameter Search

Before presenting the results, a clarification of what *hyperparameter search* is and which values and for which machine learning algorithms it is used is given first.
The hyperparameter search executes a machine learning algorithm with a wide variety of parameters. After the algorithm has been executed with all parameters, the best parameters with the corresponding results of the execution of a machine learning algorithm are retrieved. Due to the variety of parameters that are relevant for training a machine

learning algorithm in the experiments of this work, the hyperparameter search method is applied. Specifically, it finds application for the machine learning algorithms SVM, MLP and DTC as discussed in sections 5.1, 5.2 and 5.3.
For SVM, a hyperparameter search is constructed, with values of 1 and 10 for the C parameter, and with values of 1, 0.1, 0.01 for the *gamma* parameter. The C parameter decides the strictness of setting the hyper plane [62]. The higher the value chosen for C, the stricter SVM tries to avoid misclassifying emotions. The *gamma* parameter decides how well the hyper plane adapts to the data [62]. The higher the value for *gamma*, the more SVM tries to fit the hyper plane to better separate the different groups of emotions. And for the *kernel* parameter [62], *RBF* is used in the experiments as explained in section 5.1. Due to the six different variations of the composition of the parameters of hyperparameter search, SVM is executed six times.
For MLP, hyperparameter search is used only for the *alpha* parameter [63]. The values used for the *alpha* parameter are 0.1, 1.0 and 1.9. The *alpha* parameter normalizes the weights, which helps in avoiding overfitting of the model [61]. Additional parameters that do not directly affect the hyperparameter search are, the hidden layers with the value (150,100,50). Each value of this corresponds to the number of neurons of a hidden layer. As activation function *ReLU* is used. *Adam* is used as the optimization function. And the number of iterations corresponds to the value 1000. Thus, MLP is executed three times because of the three different variations, which is related to the number of values for the *alpha* parameter.
For DTC, hyperparameter search is executed only for one parameter. Namely, for the parameter with the function that measures the quality of a split of the data at a node. Section 5.3 discusses this, specifically the functions entropy and the gini index. Other than that, there are no other parameters for using DTC because this algorithm is not supposed to have any constraints. This shall generate a tree, with a variable depth and also a variable number of leaves. The generated tree shall contain in each leaf only the data corresponding to exactly one emotion.

8.3.2. Results

In this section, the results produced by SVM, MLP and DTC are presented for each dataset. The accuracy and macro recall, as described in section 6.1, are used to measure how successfully a model is trained.
Starting with the MSP-Podcast dataset. The SVM algorithm with hyperparameter search returns as a result that the best parameters are *C=1*, *gamma=1* and *kernel=RBF*. With these parameters, rounded to two decimal places, an accuracy of 11.63% and a macro recall of 22.25% are obtained. The macro recall is low, which indicates that the model does not distinguish well between the different emotions, as can be seen in the confusion matrix of the figure A.1. This confusion matrix contrasts the predicted emotions with the correct emotions.
The MLP algorithm with hyperparameter search gives as a result that the best value for alpha is 0.1. All other parameters are the same as described in 8.3.1. The result is an accuracy of 13.5% and a macro recall rounded to two decimal places of 3.23%. The

macro recall is very low, which means that the emotion predicted is almost completely misinterpreted by the model. This can be seen by observing the confusion matrix from figure A.2.
The DTC algorithm with hyperparameter search gives the result that the best function is gini. The result from running DTC with gini, rounded to two decimal places, is an accuracy of 26.38% and a macro recall of 26.94%. This is the highest macro recall of the three machine learning algorithms for the MSP-Podcast dataset. Here, the model tries to distinguish the different emotions as seen in the confusion matrix from figure A.3. However, many emotions are still misclassified. DTC creates a tree for the MSP-Podcast dataset with a depth of 27 and 1881 leaves.
In summary, the results of all three machine learning algorithms run on the MSP-Podcast dataset are not good. The resulting models can hardly distinguish different emotion and classify most of it as one or two emotions, except for the model trained with DTC. In this case, many emotions are misclassified, but the model still distinguishes between all emotions. The model trained with DTC gives better results in terms of macro recall and accuracy.
Now to the results with the ASVP-ESD dataset. Following the same order, starting with SVM. The SVM algorithm with hyperparameter search returns as a result that the best parameters are *C=1*, *gamma=1* and *kernel=RBF*. Thus, rounded to two decimal places, we get an accuracy of 32.57% and a macro recall of 2.97%. In the confusion matrix of figure A.4 we see that the model classifies all but one of the audios as neutral emotion. For this reason, the macro recall is low. The accuracy is 32.57% because most of the emotions in the validation dataset correspond to the neutral emotion. Furthermore, it can be seen that the validation dataset does not contain the emotion 'pleasure'. If it were included and if it were also classified as a neutral emotion, then the accuracy would be lower.
The execution of MLP gives as a result that the best value for alpha is 0.1. Rounded to two decimal places, an accuracy of 32.25% and a macro recall of 4.33% is retrieved. Again, the macro recall is so low because the trained model classifies almost all audios as one of two emotions, with a few exceptions. This can be seen by looking at the confusion matrix from figure A.5. Again, as with the model that was trained with SVM, most audios are classified as a neutral emotion.
The DTC algorithm with hyperparameter search gives the result that the best function is entropy. Applying DTC with entropy, rounded to two decimal places, gives an accuracy of 34.85% and a macro recall of 23.43%. The confusion matrix of the figure A.6 shows that the model has a high accuracy when it comes to the neutral emotion. Otherwise, many emotions are misclassified. DTC generates a tree of depth 17 and 593 leaves when trained with the ASVP-ESD dataset.
In summary, the results of the applied algorithms on the ASVP-ESD dataset are not good. Here, the model trained with DTC brings better results for macro recall and accuracy.
Lastly, the results for the CREMA-D dataset. Using SVM with hyperparameter search, the best parameters are *C=1*, *gamma=1* and *kernel=RBF*. The result rounded to two decimal places is an accuracy of 15.84% and a macro recall of 2.64%. This macro recall

Algorithm	Dataset	Accuracy (%)	Recall (macro) (%)
SVM	MSP-Podcast	11.63	22.25
MLP	MSP-Podcast	13.50	3.23
DTC	MSP-Podcast	26.38	26.94
SVM	ASVP-ESD	32.57	2.97
MLP	ASVP-ESD	32.25	4.33
DTC	ASVP-ESD	34.85	23.43
SVM	CREMA-D	15.84	2.64
MLP	CREMA-D	16.91	2.82
DTC	CREMA-D	37.99	38.40

Table 8.1.: Results of the used machine learning algorithms.

is very poor and the reason can be seen by inspecting the confusion matrix of figure A.7. All emotions are classified as happy. Thus, this model does not provide any insightful information either.
Using MLP, rounded to two decimal places, an accuracy of 16.91% and a macro recall of 2.82% are obtained. Here, the best value for alpha is 0.1. Again, looking at the confusion matrix from figure A.8, it can be seen that all emotions are classified as disgust. Here, the model of MLP could also not be trained to distinguish different emotions.
The DTC algorithm with hyperparameter search gives the result that the best function is entropy. The result by applying DTC with entropy, rounded to two decimal places, contains an accuracy of 37.99% and a macro recall of 38.40%. Looking at the confusion matrix from figure A.9, it can be seen that the model tries to classify the emotions. It can also be seen that more than half of the audios, with the associated emotion angry, are also recognized as such. DTC, when trained with the CREMA-D dataset, creates a tree with depth 16 and 1029 leaves.
Applying the three algorithms to the CREMA-D dataset shows that training a model with DTC gives better results in terms of macro recall and accuracy. All the results of the experiments with the machine learning algorithms used are collected in table 8.1 for quick reference.

8.4. Deep Neural Networks

This section presents the results obtained when applying a self-designed CNN architecture and using transfer learning as presented in chapter 6. The neural networks receive as input mel spectrograms, as discussed in section 4.2. The exact way these mel spectrograms are created is discussed in section 8.2.
Each experiment consists of 100 epochs and a batch size of 16. Once weighted cross entropy and once weighted focal loss with $gamma = 5.0$ are used as loss functions. For the creation of the weights of the individual emotion classes for the loss functions, all emotions of the training dataset are taken. This is then used to calculate a weight for

each emotion class that balances the inequality of the distribution of emotions. The optimization function used in all experiments is AdamW, with a starting learning rate of 0.01. A scheduler is used which decreases the learning rate for the training phase by 10% after every 20 epochs.
In addition, the datasets are split into a training dataset and a validation dataset. A random split is used to split the dataset for training and validation. For the different experiments, a generator with the seed 42 is used to ensure that the split is always the same. This is done using a random split function and a generator [14], which are included in PyTorch [56].

8.4.1. Convolutional Neural Networks

First, experiments are performed on a self-designed convolutional neural network model. These are divided into training with the whole mel spectrograms, which must be padded to the maximum length of the biggest audio file, mel spectrograms computed from two second audio chunks, and mel spectrograms built on audio chunks of the length of four seconds. As explained in 8.2, audios longer than two seconds but shorter than four seconds are padded.
First, the applied architecture of the CNN is discussed. This can be read step by step in the table A.1. Roughly speaking, the architecture can be divided into five blocks. The first four blocks each contain two convolutional layers. After the first convolutional layer of a block, the activation function ReLU is applied. After the second convolutional layer of a block, the batch is normalized by a batch norm component as explained in section 6.1 and then the activation function ReLU is applied. At the end of the first four blocks, a max pool layer is added, with the size of the kernel 2x2. Each convolutional layer has several filters of size 3x3 with a stride of 1x1 and padding 1x1. The first convolutional layer has 32 filters, the second has 64, the third 128, the fourth 128, the fifth 256 and the last 256. Each following convolutional layer takes as input the number of feature maps from the previous convolutional layer. As explained in section 6.2, each filter creates a feature map, which is another dimension of the output. At the end of the fourth block, an adaptive average pool is added, which has 1x1 as the size of the output. The fifth block is then responsible for the classification of the features from the first four blocks. For this, first, the flatten component is used to create a one dimensional vector for the first linear layer. This is followed by a dropout component, as explained in section 6.1. This temporarily deletes 10% of the inputs for the following linear layer. The linear layer follows, which takes 256 inputs and has 128 neurons. Each of these neurons has a bias. This is followed by a batch norm component and then the application of the activation function ReLU. Then again the same scheme, dropout, linear layer with 128 inputs and 64 neurons, batch norm and ReLU. Finally, a linear layer takes care of the 64 outputs of the previous layer. This linear layer has as many neurons as there are emotions to classify. So for MSP-Podcast eight, for ASVP-ESD twelve and for CREMA-D six. This presented CNN architecture has 2,349,128 parameters and all of them are optimized in the back propagation of a training phase.
Six experiments are performed on each dataset using the CNN architecture described

above. First, mel spectrograms are computed, for each individual audio file, and these are brought to the length of the longest playback duration of the dataset. This is done by simply padding the end of the mel spectrogram with zeros until the total length is equal to the longest playback duration of the dataset. The second experiment is performed, with the use of the two seconds chunks as explained in section 8.2. And the last experiment is performed with four seconds chunks, this is also explained in the 8.2 section. Here, each possibility is performed once with the cross entropy as the loss function and once with the focal loss function with $\gamma = 5.0$. For all experiments, the optimization function AdamW is used, as discussed in section 6.1.
Now for the results, starting with those for the MSP-Podcast dataset. The results for the MSP-Podcast dataset, in the form of two graphs, one for accuracy and one for macro recall, can be found in Figures A.10 - A.15. Here, the gray circle in the graph shows that macro recall is highest at this point, and its value and associated accuracy are noted in the upper left of the image. There it can be seen that the best result for the training with the cross entropy as loss function and the four second chunks are comparatively the best. An accuracy of 42.31% and a macro recall of 41.62% is achieved. This means that less than half of the emotions are correctly classified. The possible reason for this is that the MSP-Podcast belongs to the category of natural datasets. These data may contain noise and multiple voices, which makes it more difficult for the model to detect patterns of different emotions.
Consequently, the results of the ASVP dataset are presented. In figures A.16 - A.21 the results are illustrated using an accuracy and a macro recall graph. Again, the gray circle records the highest macro recall and in the top left corner of each image is noted the value of it and also the corresponding accuracy. Again, training the CNN with the cross entropy as loss function and the four second chunks performs comparatively best. An accuracy of 50.27% and a macro recall of 51.01% is achieved. Thus, more than half of the emotions of the validation dataset are correctly classified. It should be mentioned here that the preprocessed ASVP dataset consists of twelve emotions to be classified. The work of Landry et al. [39] achieves an accuracy of 69.68% with the use of the whole dataset, meaning the speech-related and non-speech-related audios. This result is higher. Possible reasons are that there the whole dataset is used, not like in this work, and another reason could be due to the linking of the CNN architecture with a recurrent neural network [39].
Lastly, the results of the models trained with the data from the CREMA-D dataset are presented. The results can be seen in figures A.22 - A.27 using the accuracy and macro recall graphs. As mentioned in the other experiments, the gray circle indicates the highest macro recall achieved and its value can be read in the upper left of the image, with the corresponding accuracy. Comparatively, the best result is achieved when training with the cross entropy as loss function and the two second chunks. The result is an accuracy of 70.35% and a macro recall of 71.02%. This means that more than two thirds of the emotions of the validation dataset were correctly classified. The reason for this rather high result could be due to several reasons. It could be the data of the dataset, these are relatively good because belonging to the category of simulated datasets, the audios are recorded by professional speakers, or actors and are rather free from noise. The highest

accuracy obtained when classifying emotions with the CREMA-D dataset in the work of Zeghidour et al. [74] is 57.8 ± 2.4 and is quite a bit lower compared to the result of this work.
From the results, it can be seen that training with the cross entropy as loss function in most cases gives better results in the experiments. All the results of the experiments with the custom CNN model as described in table A.1, are collected in table 8.2 for quick reference.

Model	Dataset	Accuracy (%)	Recall (macro) (%)
Custom CNN (cross entropy)	MSP-Podcast - 4 seconds chunks	42.31	41.62
Custom CNN (cross entropy)	ASVP-ESD - 4 seconds chunks	50.27	51.01
Custom CNN (cross entropy)	CREMA-D - 2 seconds chunks	70.35	71.02

Table 8.2.: Best results for the custom CNN model, as described in table A.1, for each dataset.

8.4.2. Transfer Learning

For transfer learning, this work uses the models *EfficientNet-B4* [67] and *MobileNetV3* [34]. These are models for image classification and they were trained with the dataset *ImageNet* [58]. This dataset contains 1000 different classes of objects and all images are in RGB format. For this reason, the mel spectrograms are extended from one dimension to three by copying the data of the first dimension to the other dimensions. EfficientNet-B4 has 19,341,616 parameters and MobileNetV3 has 5,483,032 parameters.
These models are further extended to include a dropout layer, which temporarily deletes 10% of the input. A subsequent linear layer, which passes the 1000 outputs of both models to 256 neurons. After that, a batch norm component is used, followed by an application of the activation function ReLU. Finally, the 256 outputs of the previous linear layer are processed by n neurons. Here n corresponds to the number of emotions to be classified. All neurons have a bias. In addition, for training, the first 70% of the layers is frozen from the EfficientNet-B4 model and from the MobileNetV3 model. The parameters of the frozen layers are not optimized and the values always remain the same. The parameters of the remaining 30% of the layers of the models are free for optimization.
Now to the results, starting with the MSP-Podcast dataset. The results in the form of an accuracy and a macro recall graph are captured in figures A.28 - A.39. Here, the A.28 - A.33 figures show the results from the experiments with the EfficientNet-B4 model and the A.34 - A.39 figures show the results from the experiments with the MobileNetV3 model. The gray circle indicates the highest achieved macro recall of all 100 epochs and in the upper left corner of an image its value is recorded, as well as the corresponding accuracy. The best result is considered as the highest macro recall because it represents how well a model can distinguish the different emotions. Thus, the result of the figure A.32 is the best, with an accuracy of 28.40% and a macro recall of 27.89%. This is the EfficientNet-B4 model with cross entropy as the loss function and the four second chunks. The result from section 8.4.1 is better.

The results of the experiments with the ASVP-ESD dataset are captured in figures A.40 - A.51. The gray circle indicates the highest macro recall out of 100 epochs and its value is shown in the upper left corner of the figure, as well as the corresponding accuracy. Again, the best result is determined by the highest marco recall. Thus, the result from figure A.40, with the EfficientNet-B4 model and the cross entropy as loss function and no chunking, is the best result, with an accuracy of 48.21% and a macro recall of 38.41%. Also, for the transfer learning experiments with the ASVP-ESD dataset, the results are worse than the best result from section 8.4.1.
Finally, the results of the experiments with the CREMA-D dataset are presented. These are shown in the figures A.52 - A.63. Again, the gray circle indicates the highest macro recall out of 100 epochs and its value, as well as the corresponding accuracy, is included in the upper left of each image. The best result with the highest macro recall from the experiments with the CREMA-D dataset is shown in figure A.53. This is the EfficientNet-B4 model with the focal loss with $gamma = 5$ as the loss function and no chunking. The accuracy is 51.88% and the macro recall is 52.25%. The transfer learning results with the CREMA-D dataset are worse than the best result obtained for the CREMA-D dataset in section 8.4.1.
It is noticeable that the best results come only from training an EfficientNet-B4 model. This may be due to the reason that EfficientNet-B4 has many more parameters to train than the MobileNetV3 model. All the obtained results perform worse than the obtained results with the custom CNN architecture in section 8.4.1. All the results of the experiments with transfer learning are collected in table 8.3 for quick reference.

Model	Dataset	Accuracy (%)	Recall (macro) (%)
EfficientNet-B4 (cross entropy)	MSP-Podcast - 4 seconds chunks	28.40	27.89
EfficientNet-B4 (cross entropy)	ASVP-ESD - no chunks	48.21	38.41
EfficientNet-B4 (focal loss, $\gamma = 5.0$)	CREMA-D - no chunks	51.88	52.25

Table 8.3.: Best results from transfer learning for each dataset.

8.5. Framework

A framework was implemented in this work using Python, which used the model with the self-designed CNN architecture trained with CREMA-D, with two seconds chunks and the cross entropy as a loss function. The result of this model is presented in section 8.4.1. This is the model that has an accuracy of 70.35% and a macro recall of 71.02% on the validation dataset of CREMA-D. The framework can thus classify the six emotions, angry, disgust, fear, happy, neutral and sad because the model used was trained with the CREMA-D dataset. The framework was implemented as a console application. It takes a path to an audio file in .wav format through the console and then outputs the classified emotion in the same console. This can be done as many times as desired until the framework is turned off. In doing so, the framework can be packaged into different applications. Another possibility would be that a continuous audio input, for example

from music or movie, is given to the framework and thus an emotion can always be classified for the last two seconds.
Now let's look at how the framework of this thesis works. First, the model is initialized with its layers. Then the parameters, from a file, are loaded into the model. The audio file is divided into chunks of two seconds length, and for each chunk a mel spectrogram is computed. Audio files with a playback time shorter than two seconds have a mute sound added at the end. Each of these chunks gets an emotion classified by the framework. These are not output directly because they belong together. This means that the predictions are analyzed. The emotion that is classified the most of all chunks, this is output by the framework. If different emotions have the same number of occurrences, the framework outputs for this audio file that no emotion was classified.
The framework was tested with the whole CREMA-D dataset to be able to make a statement about how long the framework needs for the classification of an audio file. When running the experiment, the maximum time required to classify an audio file is 0.6438 seconds. The minimum time taken is 0.0708 seconds and the average time is 0.1125 seconds. All times are rounded to four decimal places. Of the 7442 audio files in the CREMA-D dataset, no emotion was classified for 690 audio files.

9. Conclusion

This work first presents the concepts behind feature extraction for machine learning algorithms and neural networks. These are static features for machine learning algorithms and the computation of mel spectrograms for neural networks. After presenting the different possibilities of a framework, a framework was implemented using as a base a trained CNN, which was trained on the CREMA-D dataset. This model was chosen because it provided good results. In addition, the framework had to be designed to divide an audio file into several two second chunks and classify the emotion from each one to determine the emotion of the whole audio file. With the results of the experiments, it can be judged that using neural networks is better than using the machine learning algorithms SVM, MLP and DTC. Moreover, the preprocessing consists only of chunking and calculating the mel spectrograms for each chunk. In the result of the experiment of the framework, it can be seen that the classification of an emotion based only on an audio file does not take much time and is less than one second. The average time of the classification is so small that it can be described and used almost as a real-time application.
For future work, the concept of chunking could be worked on, so that a variable length of the chunk could be chosen, which adapts to the playback time of the audio file. In addition, an attempt could be made to improve the neural network based on the CREMA-D dataset so that it provides even better results. First of all, it could be tested if the result improves if more than 100 epochs are chosen. Another possibility would be to try if adding a recurrent neural network (RNN) architecture, as discussed in the work of Khalil et al. [37], to the existing CNN architecture would improve the results. For example, in the work of Fan et al. [25], the approach is that the extracted features of the CNN are given to the sequential learning of an RNN. This approach in the work of Fan et al. [25] is video-based. Since it has been shown in this work that good results can be obtained with image recognition of a mel spectrogram, it could be experimented whether a CNN-RNN architecture improves the results.
From this work, it can be seen that neural networks can achieve good results when combined with mel spectrograms. In addition, the presented concept of chunking is an advantage because it requires almost no usage of zero padding.

A. Appendix

```
Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
ReLU()
Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
ReLU()
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
ReLU()
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
AdaptiveAvgPool2d(output_size=(1, 1))

Flatten(start_dim=1, end_dim=-1)
Dropout(p=0.1, inplace=False)
Linear(in_features=256, out_features=128, bias=True)
BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
Dropout(p=0.1, inplace=False)
Linear(in_features=128, out_features=64, bias=True)
BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU()
Linear(in_features=64, out_features=n, bias=True)
```

Table A.1.: CNN architecture, according to PyTorch convention, which is used for the experiments. n stands for the amount of emotions.

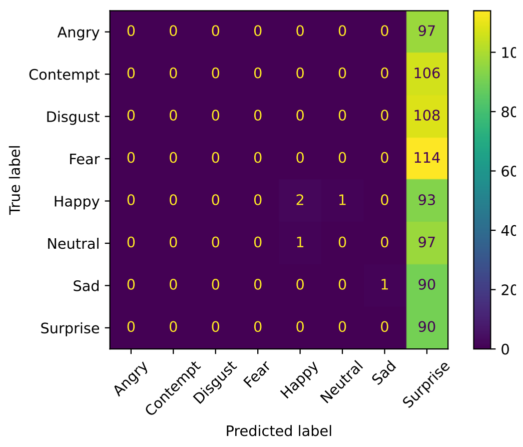


Figure A.1.: Confusion matrix after performing SVM on the MSP-Podcast dataset.

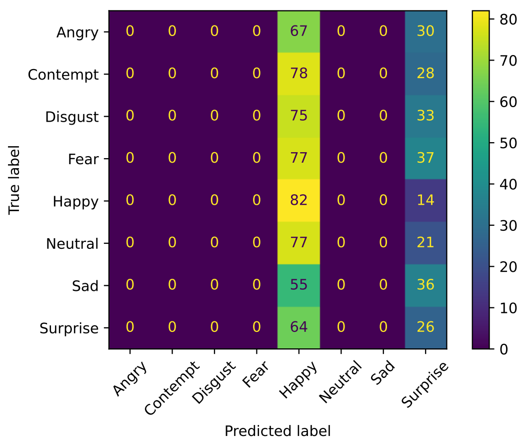


Figure A.2.: Confusion matrix after performing MLP on the MSP-Podcast dataset.

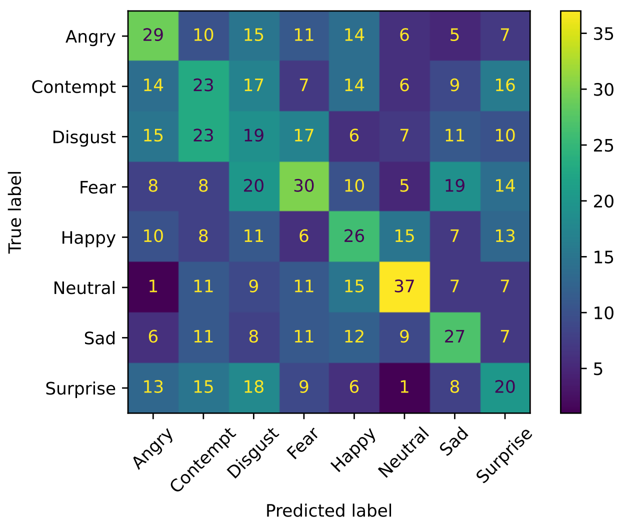


Figure A.3.: Confusion matrix after performing DTC on the MSP-Podcast dataset.

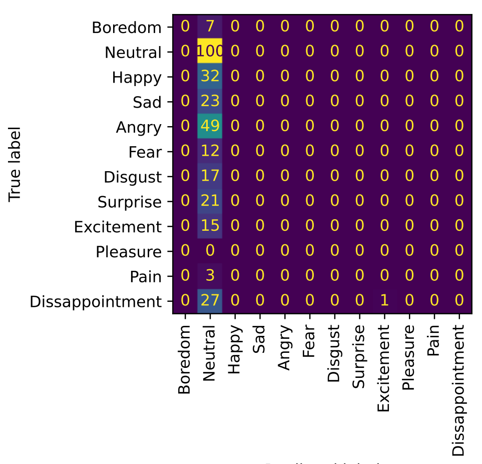


Figure A.4.: Confusion matrix after performing SVM on the ASVP-ESD dataset.

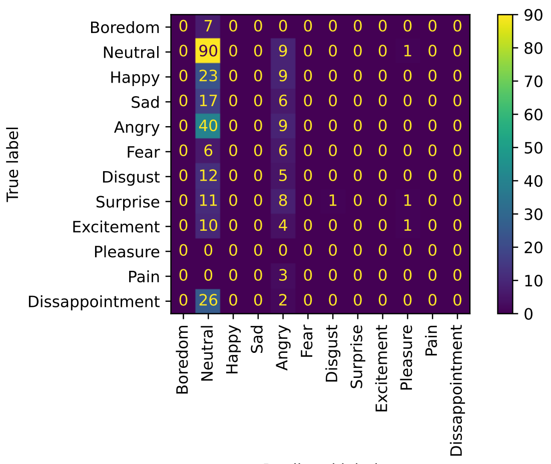


Figure A.5.: Confusion matrix after performing MLP on the ASVP-ESD dataset.

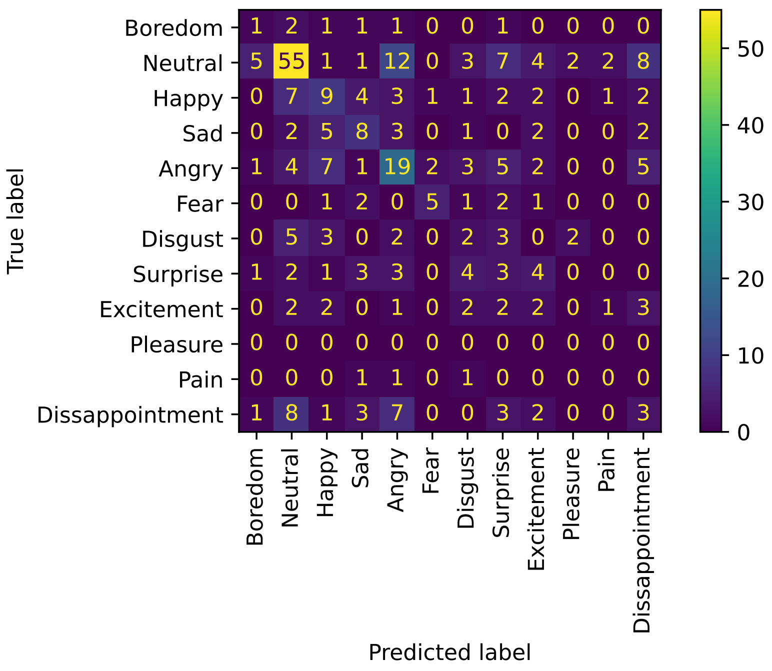


Figure A.6.: Confusion matrix after performing DTC on the ASVP-ESD dataset.

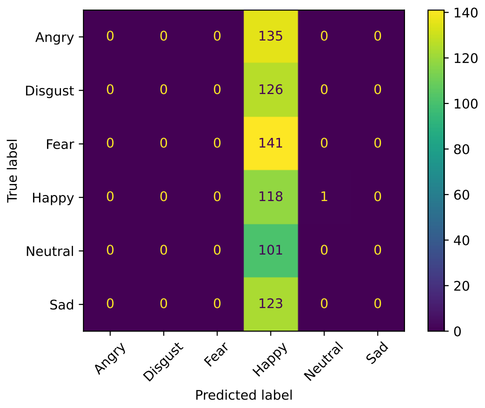


Figure A.7.: Confusion matrix after performing SVM on the CREMA-D dataset.

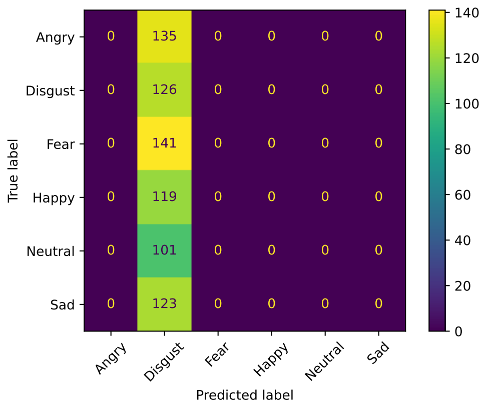


Figure A.8.: Confusion matrix after performing MLP on the CREMA-D dataset.

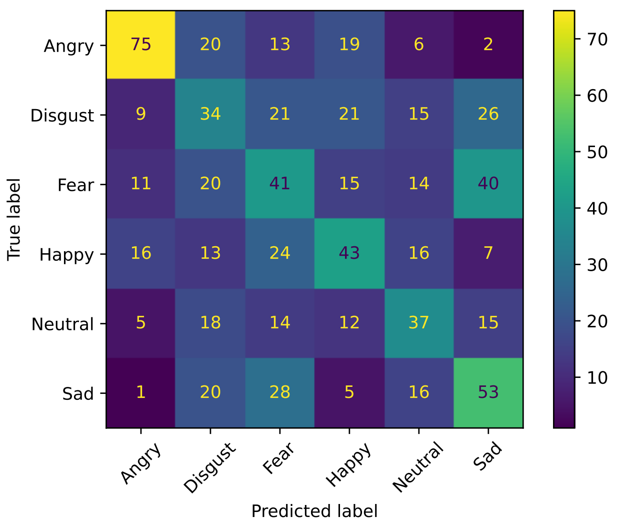


Figure A.9.: Confusion matrix after performing DTC on the CREMA-D dataset.

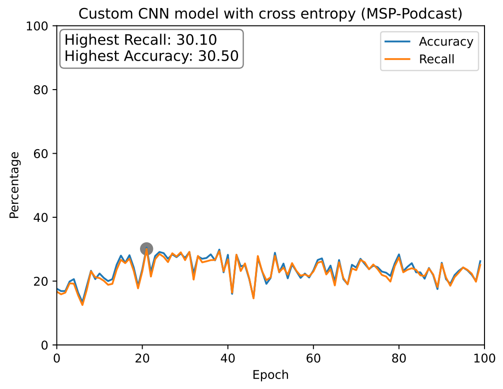


Figure A.10.: 100 epochs of training custom CNN model with cross entropy on MSP-Podcast dataset.

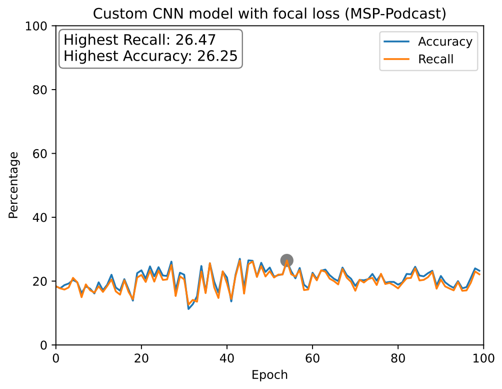


Figure A.11.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on MSP-Podcast dataset.

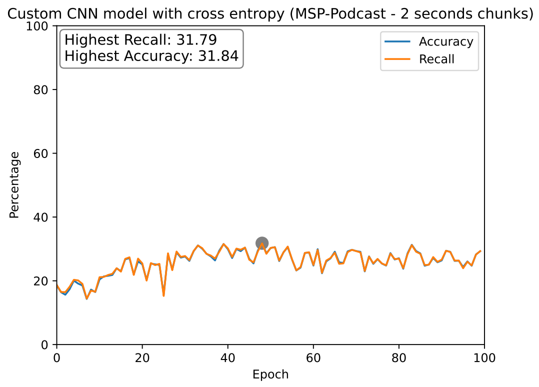


Figure A.12.: 100 epochs of training custom CNN model with cross entropy on MSP-Podcast dataset (2 seconds chunks).

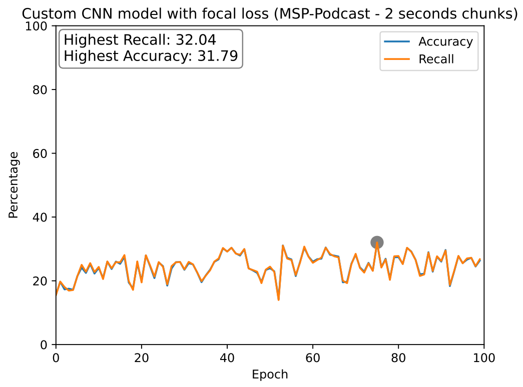


Figure A.13.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on MSP-Podcast dataset (2 seconds chunks).

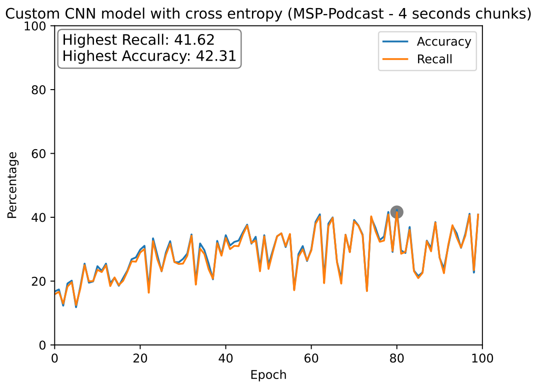


Figure A.14.: 100 epochs of training custom CNN model with cross entropy on MSP-Podcast dataset (4 seconds chunks).

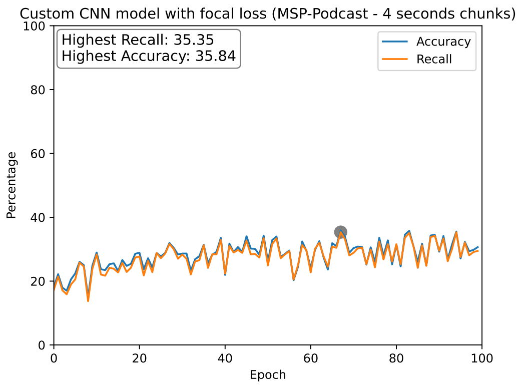


Figure A.15.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on MSP-Podcast dataset (4 seconds chunks).

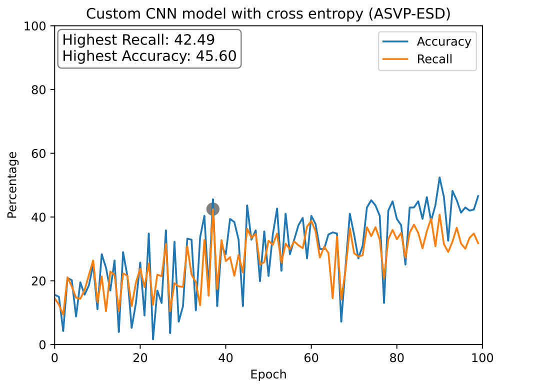


Figure A.16.: 100 epochs of training custom CNN model with cross entropy on ASVP-ESD dataset.

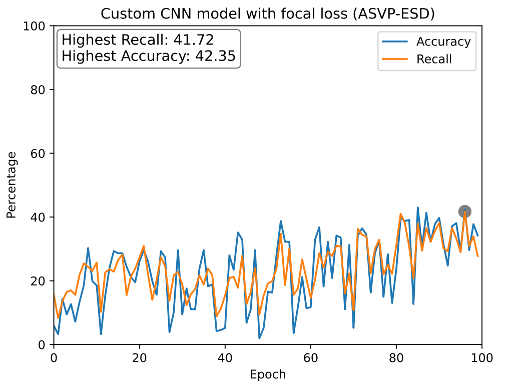


Figure A.17.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on ASVP-ESD dataset.

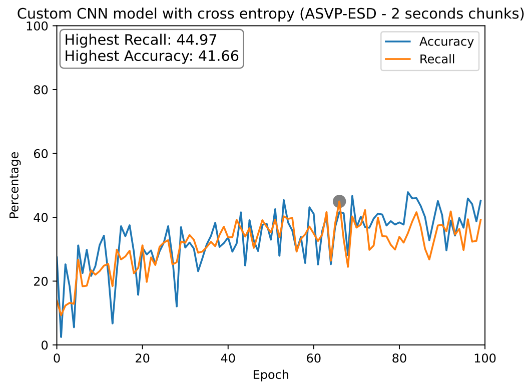


Figure A.18.: 100 epochs of training custom CNN model with cross entropy on ASVP-ESD dataset (2 seconds chunks).

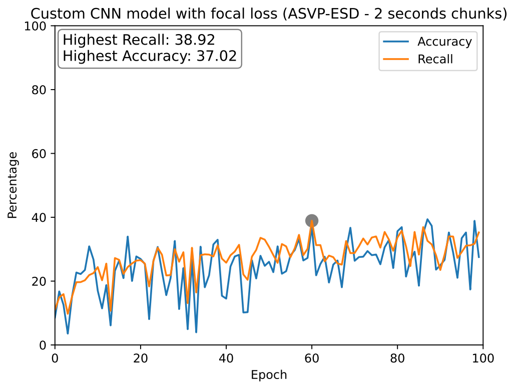


Figure A.19.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on ASVP-ESD dataset (2 seconds chunks).

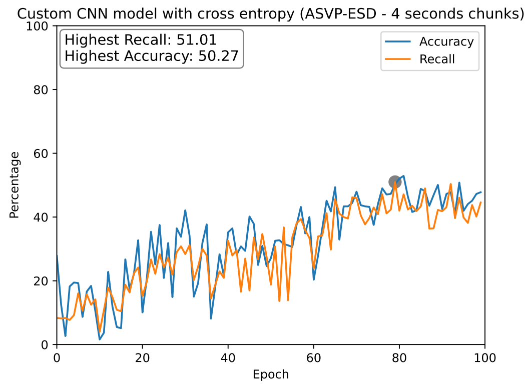


Figure A.20.: 100 epochs of training custom CNN model with cross entropy on ASVP-ESD dataset (4 seconds chunks).

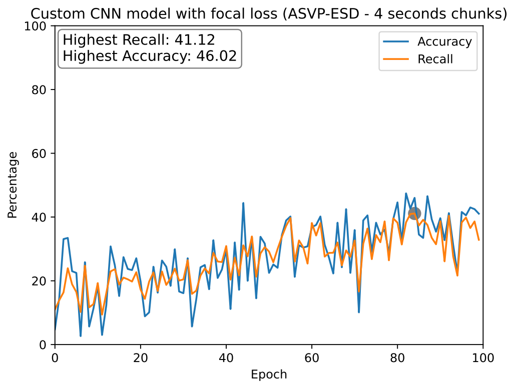


Figure A.21.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on ASVP-ESD dataset (4 seconds chunks).

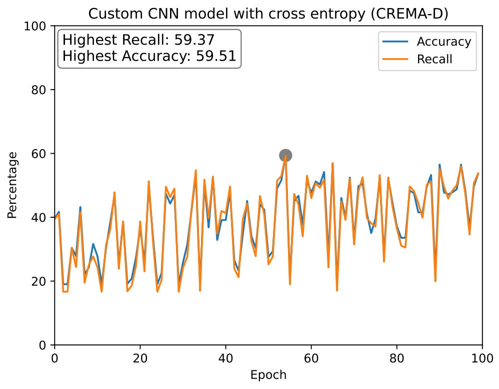


Figure A.22.: 100 epochs of training custom CNN model with cross entropy on CREMA-D dataset.

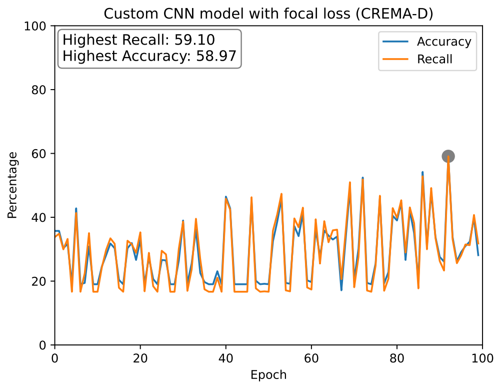


Figure A.23.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on CREMA-D dataset.

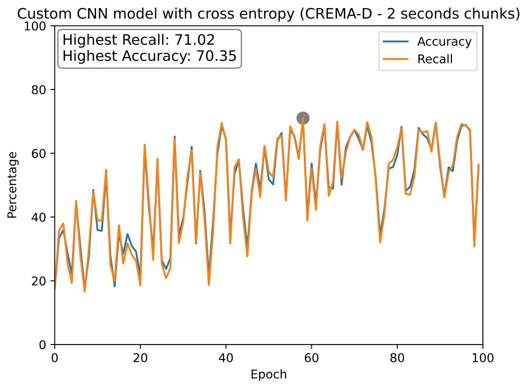


Figure A.24.: 100 epochs of training custom CNN model with cross entropy on CREMA-D dataset (2 seconds chunks).

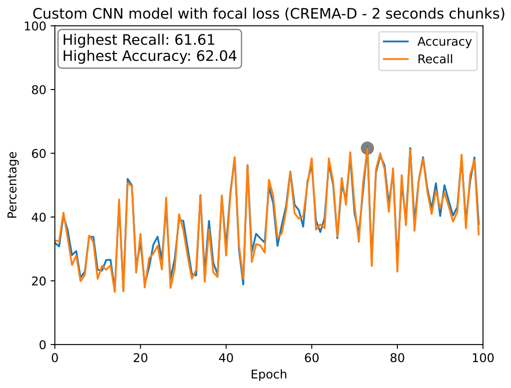


Figure A.25.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on CREMA-D dataset (2 seconds chunks).

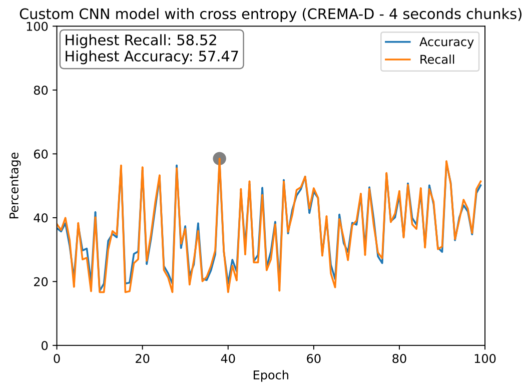


Figure A.26.: 100 epochs of training custom CNN model with cross entropy on CREMA-D dataset (4 seconds chunks).

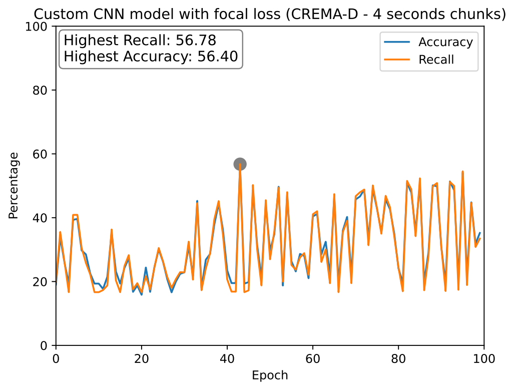


Figure A.27.: 100 epochs of training custom CNN model with focal loss ($\gamma = 5$) on CREMA-D dataset (4 seconds chunks).

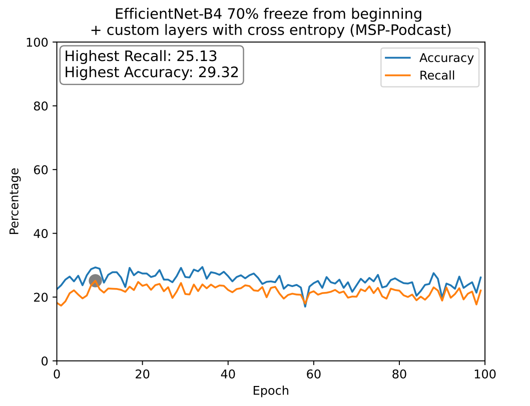


Figure A.28.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset.

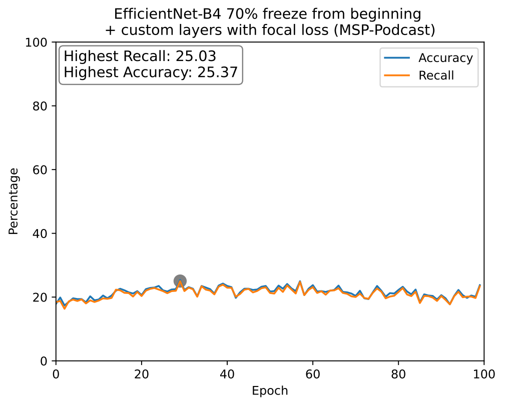


Figure A.29.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset.

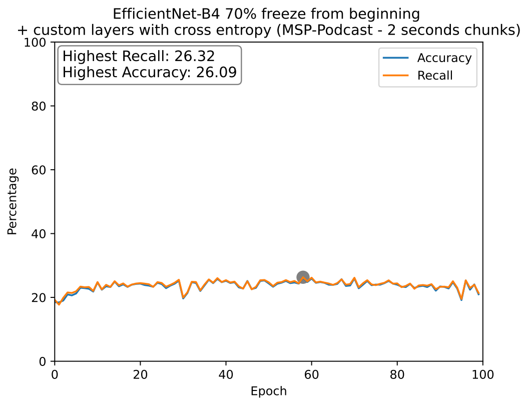


Figure A.30.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset (2 seconds chunks).

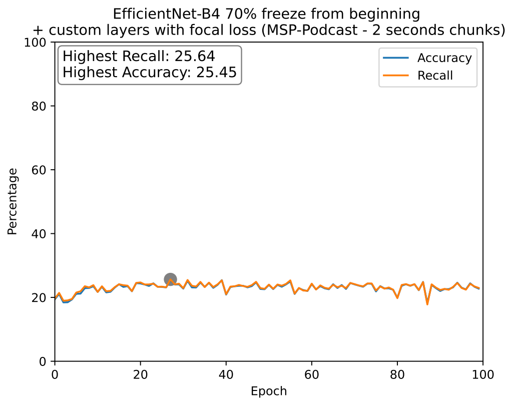


Figure A.31.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset (2 seconds chunks).

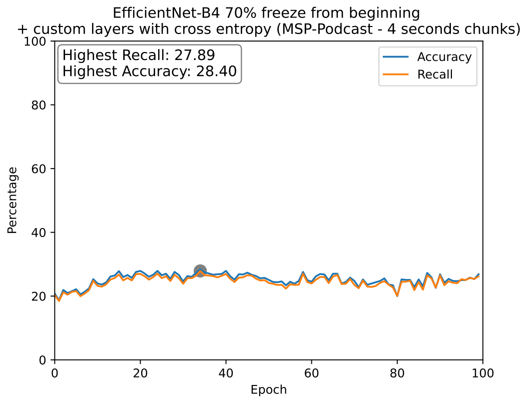


Figure A.32.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset (4 seconds chunks).

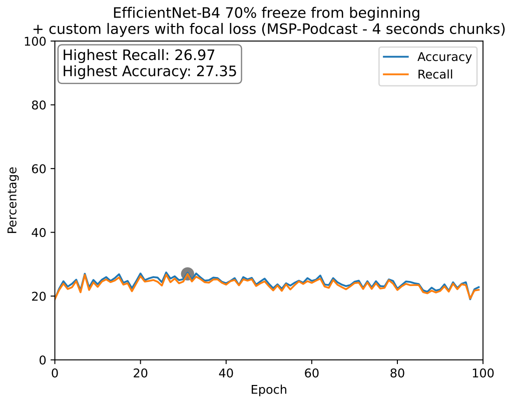


Figure A.33.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset (4 seconds chunks).

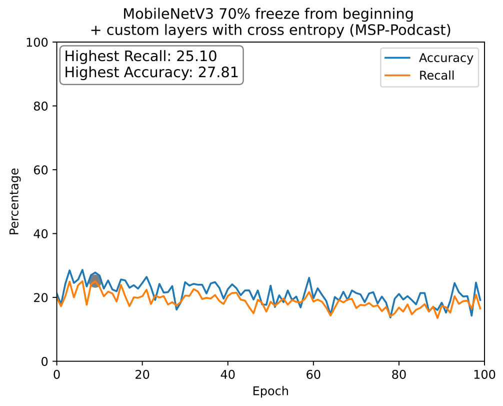


Figure A.34.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset.

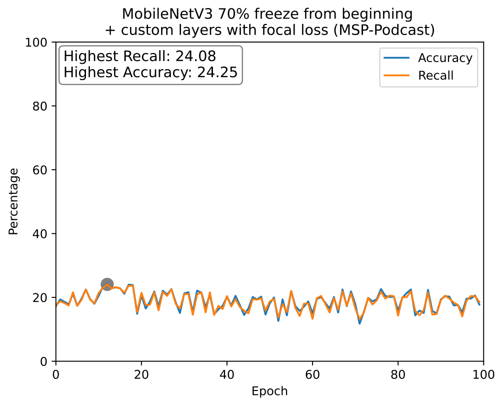


Figure A.35.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset.

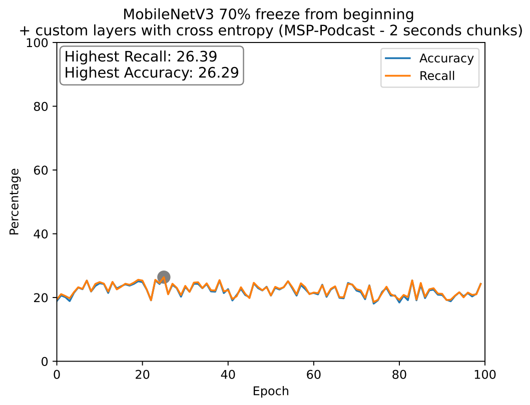


Figure A.36.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset (2 seconds chunks).

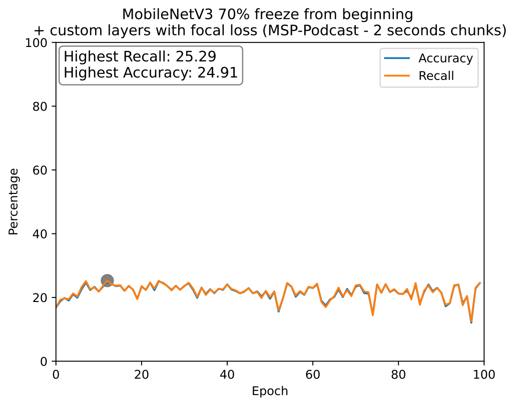


Figure A.37.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset (2 seconds chunks).

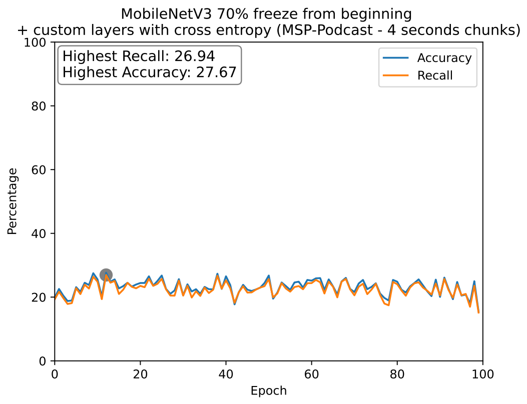


Figure A.38.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on MSP-Podcast dataset (4 seconds chunks).

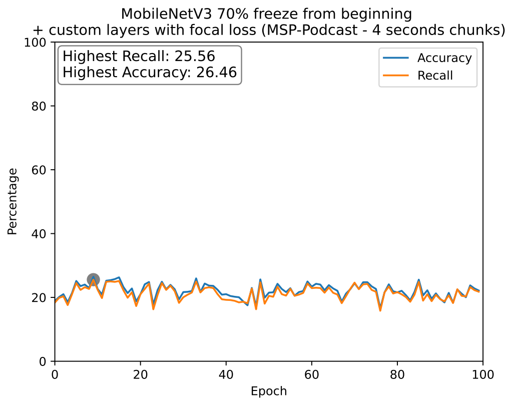


Figure A.39.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on MSP-Podcast dataset (4 seconds chunks).

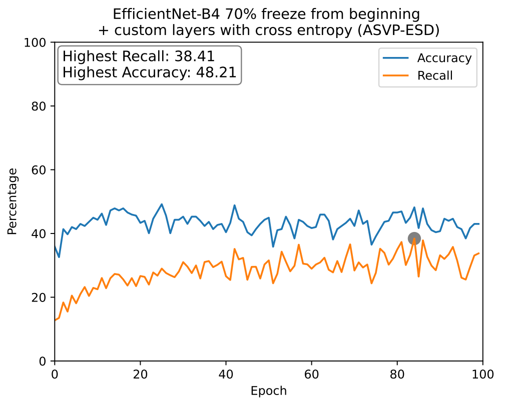


Figure A.40.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset.

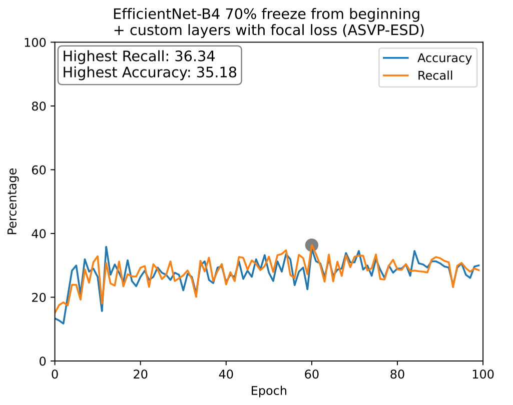


Figure A.41.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset.

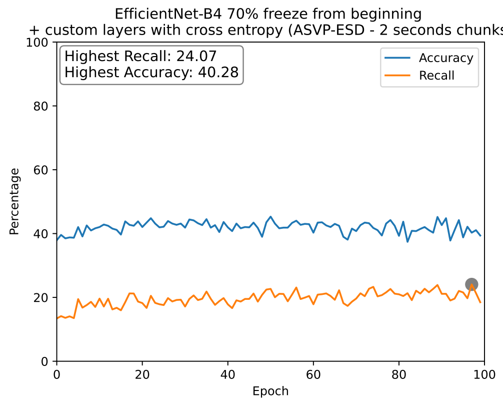


Figure A.42.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset (2 seconds chunks).

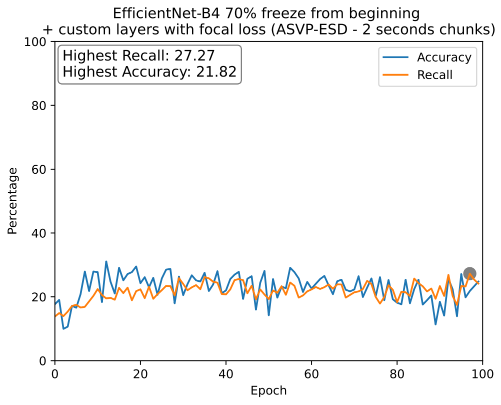


Figure A.43.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset (2 seconds chunks).

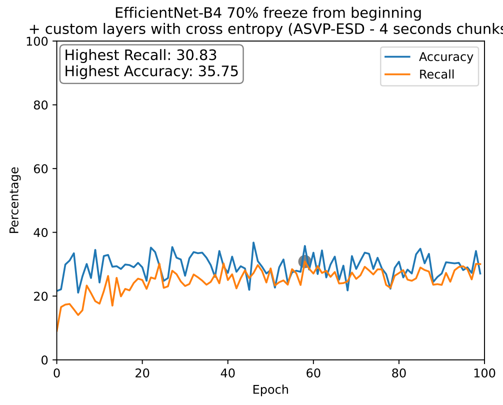


Figure A.44.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset (4 seconds chunks).

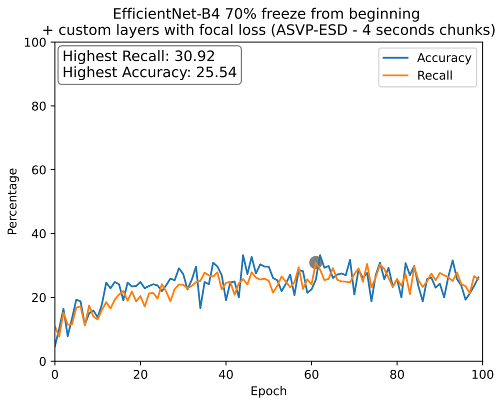


Figure A.45.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset (4 seconds chunks).

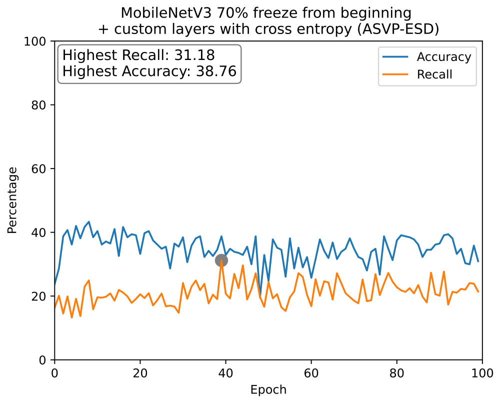


Figure A.46.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset.

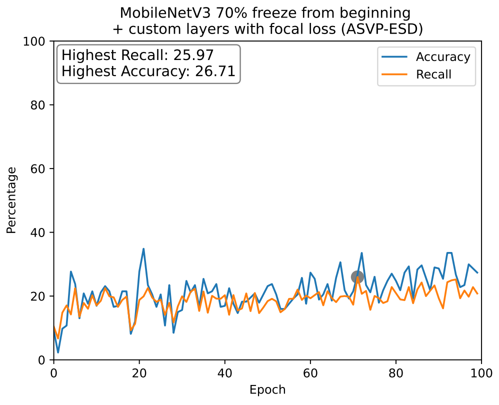


Figure A.47.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset.

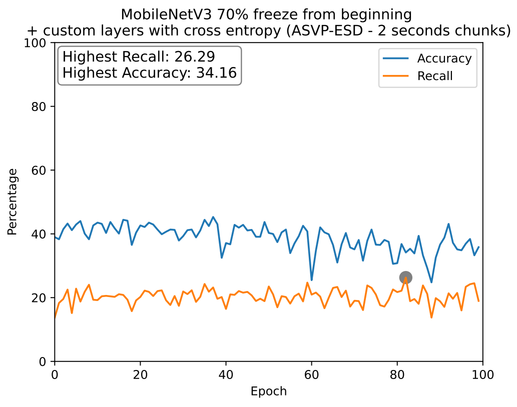


Figure A.48.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset (2 seconds chunks).

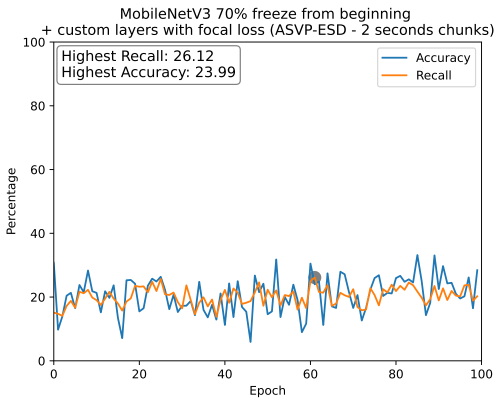


Figure A.49.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset (2 seconds chunks).

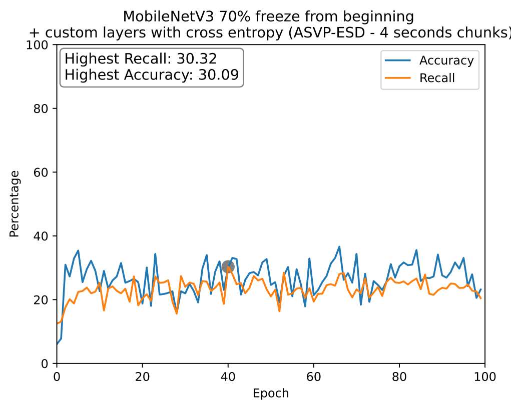


Figure A.50.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on ASVP-ESD dataset (4 seconds chunks).

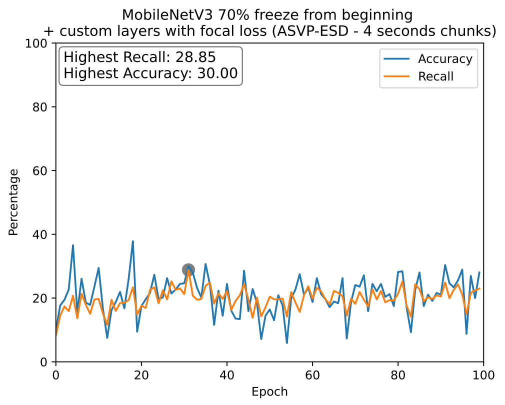


Figure A.51.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on ASVP-ESD dataset (4 seconds chunks).

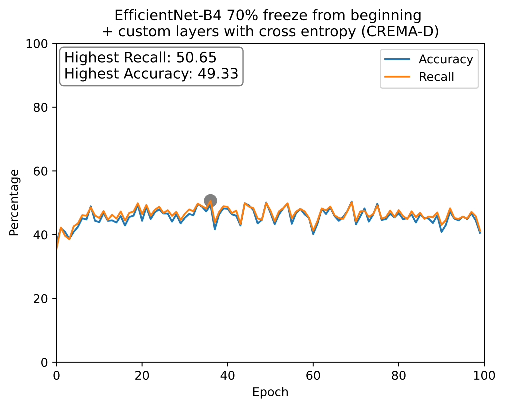


Figure A.52.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset.

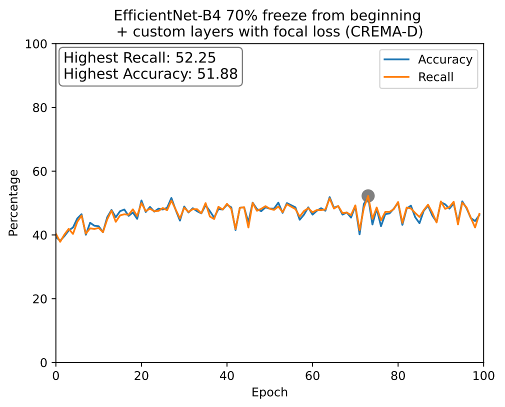


Figure A.53.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset.

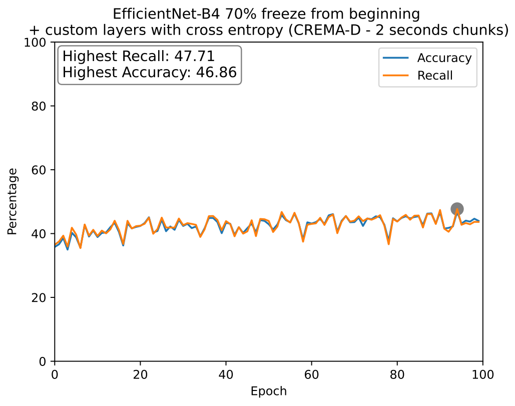


Figure A.54.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset (2 seconds chunks).

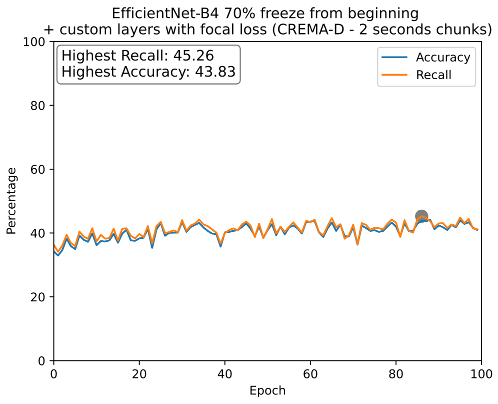


Figure A.55.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset (2 seconds chunks).

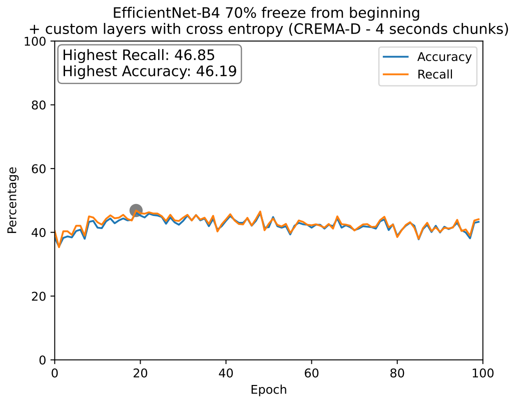


Figure A.56.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset (4 seconds chunks).

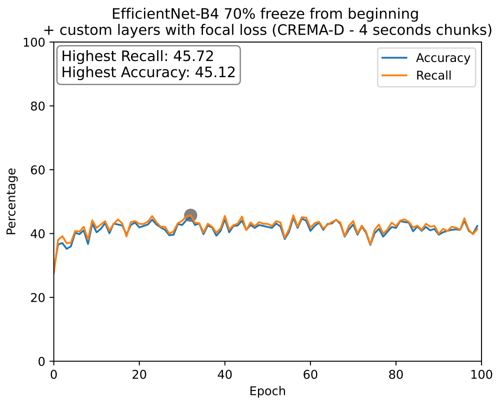


Figure A.57.: 100 epochs of training EfficientNet-B4 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset (4 seconds chunks).

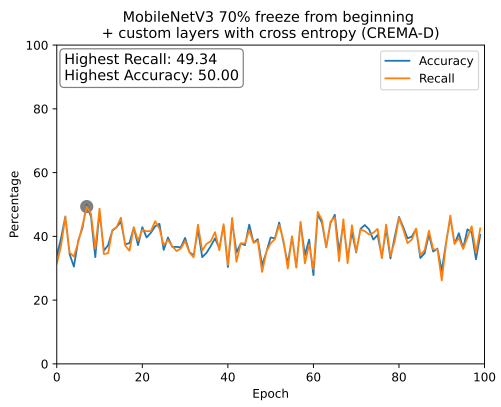


Figure A.58.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset.

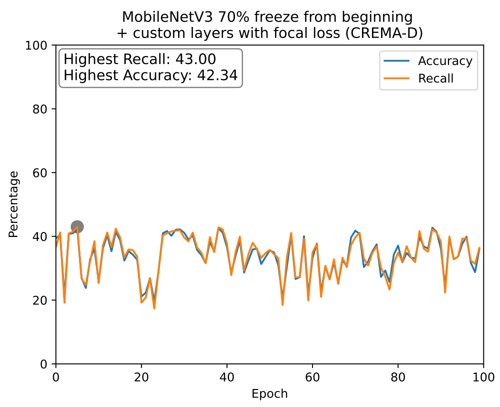


Figure A.59.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset.

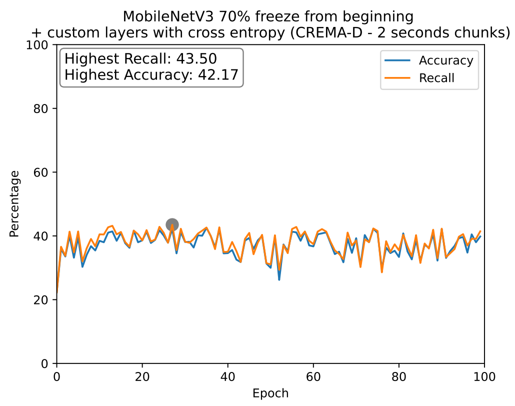


Figure A.60.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset (2 seconds chunks).

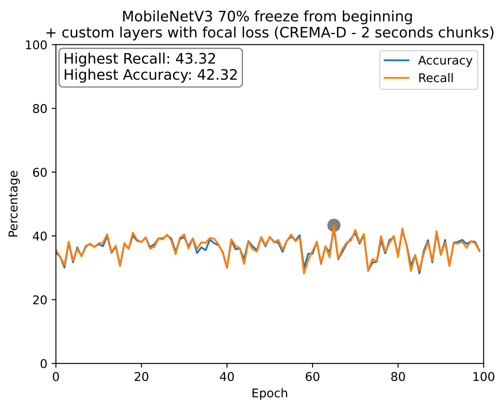


Figure A.61.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset (2 seconds chunks).

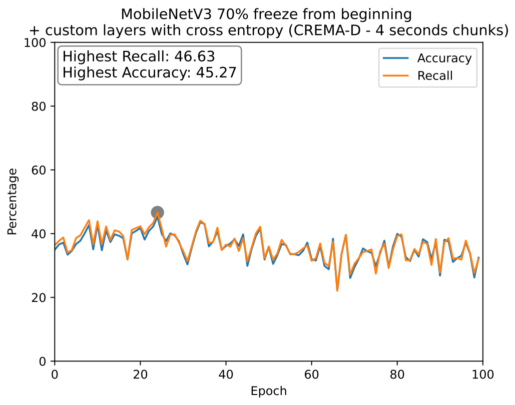


Figure A.62.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with cross entropy on CREMA-D dataset (4 seconds chunks).

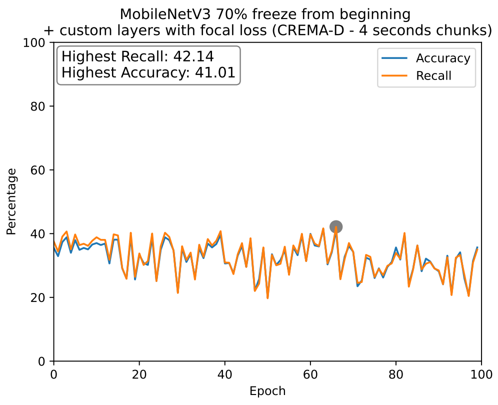


Figure A.63.: 100 epochs of training MobileNetV3 model with 70% freeze from beginning plus custom layers at the end with focal loss ($\gamma = 5$) on CREMA-D dataset (4 seconds chunks).

Bibliography

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[2] audEERING GmbH. opensmile python. `https://audeering.github.io/opensmile-python/`, 2023. Accessed: 2023-04-10.

[3] Yalin Baştanlar and Mustafa Özuysal. Introduction to machine learning. *miRNomics: MicroRNA biology and computational analysis*, pages 105–128, 2014.

[4] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1, chapter 5.3 Generalization, Capacity, Overfitting and Underfitting. MIT press Cambridge, MA, USA, 2017.

[5] Jason Brownlee. A gentle introduction to early stopping to avoid overtraining neural networks. `https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/`, 2019. Accessed: 2023-04-05.

[6] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control.* Cambridge University Press, 2022.

[7] Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N Chang, Sungbok Lee, and Shrikanth S Narayanan. Iemocap: Interactive emotional dyadic motion capture database. *Language resources and evaluation*, 42:335–359, 2008.

[8] Houwei Cao, David G. Cooper, Michael K. Keutmann, Ruben C. Gur, Ani Nenkova, and Ragini Verma. Crema-d: Crowd-sourced emotional multimodal actors dataset. *IEEE Transactions on Affective Computing*, 5(4):377–390, 2014.

[9] PyTorch Contributors. Adaptiveavgpool2d. `https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html`, 2023. Accessed: 2023-04-05.

[10] PyTorch Contributors. Batchnorm1d. `https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html`, 2023. Accessed: 2023-04-05.

[11] PyTorch Contributors. Dropout. `https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html`, 2023. Accessed: 2023-04-05.

[12] PyTorch Contributors. Linear. `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html`, 2023. Accessed: 2023-04-05.

[13] PyTorch Contributors. Torch.flatten. `https://pytorch.org/docs/stable/generated/torch.flatten.html`, 2023. Accessed: 2023-04-05.

[14] PyTorch Contributors. Torch.utils.data. `https://pytorch.org/docs/stable/data.html`, 2023. Accessed: 2023-04-05.

[15] Tientcheu Touko Landry dejoli. Asvp-esd(speech &amp; non-speech emotional sound), 2022.

[16] Jun Deng, Zixing Zhang, Erik Marchi, and Björn Schuller. Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 511–516, 2013.

[17] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

[18] Devopedia. Audio feature extraction. `https://devopedia.org/audio-feature-extraction#cite-as`, 2021. Version 8, May 23. Accessed: 2023-04-03.

[19] Ketan Doshi. Audio deep learning made simple - why mel spectrograms perform better. `https://ketanhdoshi.github.io/Audio-Mel/`, 2021. Accessed: 2023-04-02.

[20] Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. *Electrical Engineering and Computer Science*, 2002.

[21] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[22] Daniel Durstewitz, Georgia Koppe, and Andreas Meyer-Lindenberg. Deep neural networks in psychiatry. *Molecular psychiatry*, 24(11):1583–1598, 2019.

[23] Florian Eyben, Felix Weninger, Florian Gross, and Björn Schuller. Recent developments in opensmile, the munich open-source multimedia feature extractor. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, page 835–838, New York, NY, USA, 2013. Association for Computing Machinery.

[24] Florian Eyben, Martin Wöllmer, and Björn Schuller. Opensmile: The munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1459–1462, New York, NY, USA, 2010. Association for Computing Machinery.

[25] Yin Fan, Xiangju Lu, Dian Li, and Yuanliu Liu. Video-based emotion recognition using cnn-rnn and c3d hybrid networks. In *Proceedings of the 18th ACM international conference on multimodal interaction*, pages 445–450, 2016.

[26] Haytham M. Fayek. Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between. `https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html`, 2016. Accessed: 2023-04-02.

[27] Haytham M Fayek, Margaret Lech, and Lawrence Cavedon. Evaluating deep learning architectures for speech emotion recognition. *Neural Networks*, 92:60–68, 2017.

[28] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[29] Google. Classification: Accuracy. `https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=en`. Accessed: 2023-04-06.

[30] Google. Classification: Precision and recall. `https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall?hl=en`. Accessed: 2023-04-06.

[31] Fabio M. Graetz. Why adamw matters. `https://towardsdatascience.com/why-adamw-matters-736223f31b5d`, 2018. Accessed: 2023-04-05.

[32] Colin Green. Cross entropy. `https://heliosphan.org/cross-entropy.html`, 2016. Accessed: 2023-04-04.

[33] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[34] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[36] Jonathan Johnson. What's a deep neural network? deep nets explained. `https://www.bmc.com/blogs/deep-neural-network/`, 2020. Accessed: 2023-04-04.

[37] Ruhul Amin Khalil, Edward Jones, Mohammad Inayatullah Babar, Tariqullah Jan, Mohammad Haseeb Zafar, and Thamer Alhussain. Speech emotion recognition using deep learning techniques: A review. *IEEE Access*, 7:117327–117345, 2019.

[38] Thomas A Lampert and Simon EM O'Keefe. A survey of spectrogram track detection algorithms. *Applied acoustics*, 71(2):87–100, 2010.

[39] Dejoli Landry, Qianhua He, Haikang Yan, and Yanxiong Li. Asvp-esd: A dataset and its benchmark for emotion recognition using both speech and non-speech utterances. *Global Scientific Journals*, 2020.

[40] Siddique Latif, Rajib Rana, Sara Khalifa, Raja Jurdak, Junaid Qadir, and Bjoern W Schuller. Survey of deep representation learning for speech emotion recognition. *IEEE Transactions on Affective Computing*, pages 1–1, 2021.

[41] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[42] librosa. librosa.stft. `https://librosa.org/doc/main/generated/librosa.stft.html#librosa.stft`. Accessed: 2023-04-03.

[43] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[44] Wei-Cheng Lin and Carlos Busso. Chunk-level speech emotion recognition: A general framework of sequence-to-one dynamic temporal modeling. *IEEE Transactions on Affective Computing*, 2021.

[45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[46] Reza Lotfian and Carlos Busso. Building naturalistic emotionally balanced speech corpus by retrieving emotional speech from existing podcast recordings. *IEEE Transactions on Affective Computing*, 10(4):471–483, 2019.

[47] Vikashraj Luhaniwal. Forward propagation in neural networks - simplified math and code version. `https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250`, 2019. Accessed: 2023-04-04.

[48] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. `http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/`, 2009. Accessed: 2023-04-02.

[49] MathWorks. hz2mel - convert from hertz to mel scale. `https://www.mathworks.com/help/audio/ref/hz2mel.html`, 2019. Accessed: 2023-04-03.

[50] Brian McFee, Colin Raffel, Dawen Liang, Daniel P Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25, 2015.

[51] Sakorn Mekruksavanich, Anuchit Jitpattanakul, and Narit Hnoohom. Negative emotion recognition using deep learning for thai language. In *2020 joint international conference on digital arts, media and technology with ECTI northern section conference on electrical, electronics, computer and telecommunications engineering (ECTI DAMT & NCON)*, pages 71–74. IEEE, 2020.

[52] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[53] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 4:5, 2005.

[54] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[55] Srinivas Parthasarathy and Carlos Busso. Predicting emotionally salient regions using qualitative agreement of deep neural network regressors. *IEEE Transactions on Affective Computing*, 12(2):402–416, 2021.

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2015.

[59] S.R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991.

[60] Björn Schuller, Stefan Steidl, Anton Batliner, Julia Hirschberg, Judee K Burgoon, Alice Baird, Aaron Elkins, Yue Zhang, Eduardo Coutinho, and Keelan Evanini. The interspeech 2016 computational paralinguistics challenge: Deception, sincerity & native language. In *17TH Annual Conference of the International Speech Communication Association (Interspeech 2016), Vols 1-5*, volume 8, pages 2001–2005. ISCA, 2016.

[61] scikit learn. 1.17. neural network models (supervised). `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`. Accessed: 2023-03-29.

[62] scikit learn. 1.4. support vector machines. `https://scikit-learn.org/stable/modules/svm.html`. Accessed: 2023-03-28.

[63] scikit learn. sklearn.neural_network.mlpclassifiern. `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`. Accessed: 2023-03-29.

[64] scikit learn. sklearn.tree.decisiontreeclassifier. `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. Accessed: 2023-03-31.

[65] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[66] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[67] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[68] Arslan Tariq. What is the difference between micro and macro averaging? `https://www.educative.io/answers/what-is-the-difference-between-micro-and-macro-averaging`. Accessed: 2023-04-06.

[69] Wikipedia. Cross entropy. `https://en.wikipedia.org/wiki/Cross_entropy`. Accessed: 2023-04-04.

[70] Wikipedia. Mel scale. `https://en.wikipedia.org/wiki/Mel_scale`. Accessed: 2023-04-03.

[71] Thomas Wood. Softmax function. `https://deepai.org/machine-learning-glossary-and-terms/softmax-layer`. Accessed: 2023-04-04.

[72] Laurenz Wuttke. Deep learning: Definition, beispiele & frameworks. `https://datasolut.com/was-ist-deep-learning/`. Accessed: 2023-04-04.

[73] Lonce Wyse. Audio spectrogram representations for processing with convolutional neural networks. *arXiv preprint arXiv:1706.09559*, 2017.

[74] Neil Zeghidour, Olivier Teboul, Félix de Chaumont Quitry, and Marco Tagliasacchi. Leaf: A learnable frontend for audio classification. *arXiv preprint arXiv:2101.08596*, 2021.