CS464 Introduction to Machine Learning

Emotion Detection by Speech

Final Report

Group 6

Funda Tan	21801861
Lara Fenercioğlu	21802536
Yekta Seçkin Satır	21903227
Latife Sezin İsmailoğlu	21704280
Eren İhsan Erzin	21703244

Table of Contents

1. Introduction

The main purpose of this project is to detect which emotion is present in a given speech. A human speech consists of various features and there are still debates about what features of the human voice influence the emotion present in the speech. Even though we as humans have difficulty recognizing the emotion of a person from their speech, it makes it even harder for the computer to recognize.

We have converted the speeches to MFCC to take advantage of its ability to demonstrate the short-term power spectrum of a sound and take the coefficients as features. So, in this project, we use SVM, CNN, and RNN algorithms to classify different emotions. The dataset, feature selection algorithm, and the details about the models will be discussed in this report which will allow us to comment on the accuracies of the models.

With our SVM classifier, we have obtained a 97.2% accuracy in predicting the emotion of a speech. Some important metric values that we have obtained from our SVM model were

- Precision: 0.9683
- Recall: 0.9683
- F_Score: 0.9666

The confusion matrix shows that the SVM classifier is good at labeling samples that belong to angry and neutral emotions but has some problems with predicting fear emotion.

We have trained two different models for CNN, one is with SGD and the other one is with Adam optimizer. The best model is Adam. So, with our CNN classifier, we have obtained a 96.28% accuracy in predicting the emotion of a speech. Some important metrics values that we have obtained from our CNN model were

- Precision: 0.9597523
- Recall: 0.96723866
- F_Score: 0.9634809

The confusion matrix shows that the classifier is most successful at predicting speeches that belong to neutral and angry emotions and least successful at predicting speeches that belong to the fear emotion by confusing with happy emotion.

With our RNN classifier, we have obtained a 91.18% accuracy in predicting the emotion of a speech. Some important metric values that we have obtained from our best RNN model were

- Precision: 0.9040

- Recall: 0.9182
- F_Score: 0.9110

2. Problem Description

In this project, our aim is to detect the emotion of a specific speech and determine its emotion according to the chosen emotions which are angry, disgust, fear, neutral, happy, and sad. Since it is hard to detect a very specific emotion we have chosen some basic ones. Also, it is hard to determine which features of the human voice affect emotion so mfcc is a way of overcoming this challenge. It chooses which coefficient affects the emotion the most. Here is the question we are trying to answer:

- Which emotions are more likely to be confused by the model?

3. Methods

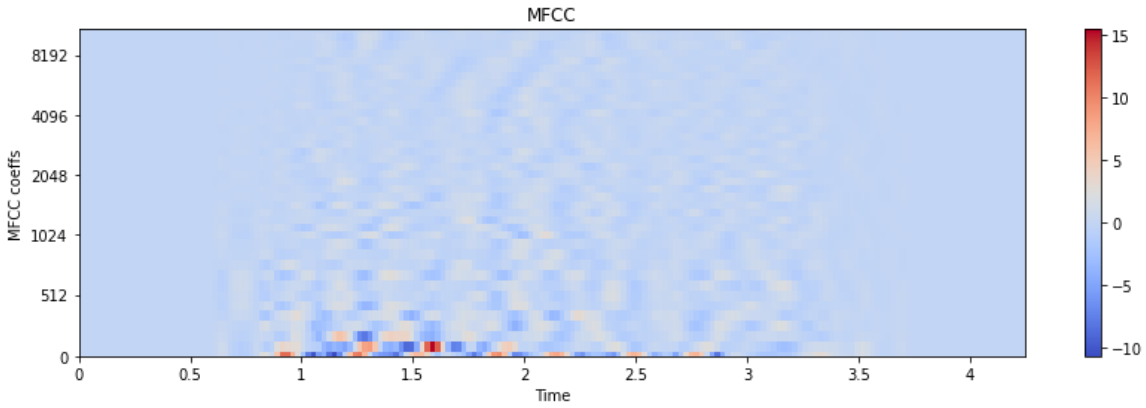
Dataset

We have chosen 2 different audio datasets which are TESS and RAVDESS. TESS dataset has 2800 audio files which feature actresses speaking from the Toronto area. The recordings were made of seven emotions, Anger, Disgust, Fear, Happiness, Pleasant Surprise, Sadness, and Neutral. Lastly, RAVDESS dataset contains 7356 files which are recorded from 12 females and 12 males. The speeches include Calm, Happy, Sad, Angry, Fearful, Surprised, and Disgust emotions. For our project, we only have chosen 6 emotions and their corresponding recordings from the prepared datasets and our labels are angry, disgusted, fearful, happy, neutral, and sad. We have used 828 audio samples from RAVDESS and 2400 samples from TESS. Also, in order to avoid imbalance class distribution, we have selected the same amount of emotion recordings. For training purposes, we have split the dataset into 3 parts. %70 is the train, %10 is the validation, and %20 is the test. We have used sklearn library's train_test_split() method. The reason we have a validation set is to validate the model's performance during training which is helpful to avoid overfitting because validation data is like a tool that determines whether the training is moving in the right direction or not.

MFCC

The method we use to transform the information in these songs into a form that we can process is MFCC, the Mel-frequency cepstral coefficients. The MFCC first takes the Fourier Transform of the signal then it maps the powers of the spectrum obtained onto the mel scale, using triangular overlapping windows. It

takes the logarithm of the power of each MEL frequency and by taking the cosine form of these logs, the result becomes a frequency spectrum and an amplitude plot. Here is a train sample's mfcc spectrogram below:



For Feature Extraction, MFCC is used. "extract_feature" function defined for this purpose. In order to use MFCC, we have loaded librosa library. It takes audio files as input by the librosa.load function. The name of the path for the given audio and resample type is the parameters. If you want to reduce the load time res_type can be added as an optional parameter with default value kaiser_best but we are using kaiser_fast to reduce load time even more. To calculate the coefficients, librosa.feature.mfcc is used. X is passed as y to indicate audio time series. sr is basically the sampling rate (samples taken per second) of y. Lastly, a number of MFCCs have been stated in the n_mfcc parameter and as a result, it gives 40 feature values for each audio file. The function can be seen in the Appendix.

CNN

- Initially, we have implemented the built-in model from the TensorFlow Keras. Here is the model creation: model = Sequential(). We have chosen this model because it is easier to build and understand. It is a linear stack of layers which we can add any layer by writing model.add().
- Then, we used a one-dimensional convolutional neural network with input size 40 since we have 40 MFCCs features. The input layer is called in the following way: model.add(Conv1D(256, 8, padding = "same", input_shape(X_train.shape[1],1))).
- Batch normalization is another layer we used in the CNN model before activation layers. It is used to standardize the inputs to a network. This normalization technique helps to accelerate the training (reducing the

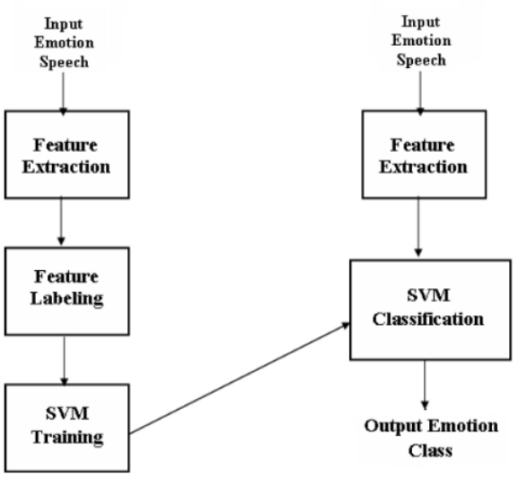
number of training epochs required to train a network) and reduce generalization error.

- We added activation layers using a rectified linear unit (ReLu) for faster training. The activation function is ReLu because it tends to converge much more quickly and reliably compared to a sigmoid function.
- We used Max Pooling 1D with pool sizes 8 and 5. Max pooling reduced the dimensionality of the feature map by calculating the maximum value in patches.
- Then we added dropout layers with a rate of 0.25. We used dropout layers as they prevent overfitting in training data by dropping out some neurons randomly.
- The output layer is called according to the label count with model.add(Dense(6)).
- We have chosen the learning rate as 0.0001 because we saw that as the learning rate decreases to 0.0001, the better the model becomes. Learning rate is an important hyperparameter because it controls how quickly or slowly a neural network model learns a specific task.
- The loss function is determined to be the categorical cross-entropy from the Keras library.
- No of the convolutional layers in the network is 7. It is usually good to add more layers until test error no longer decreases. However, the trade-off is that as we increase the layer number, the model becomes computationally more expensive to train the network. On the other hand, having a small number of layers may lead to underfitting while having more layers is usually not harmful when appropriate regularization is applied.
- The softmax activation function is used at the last layer since it converts a vector of numbers into a vector of probabilities.
- The epoch size is determined to be 700 with a batch size of 16. As can be understood from the loss plots, the loss converges after 300 epochs which means that even if we increase the epoch size there wouldn't be much decrease in the loss. Also, since larger batch sizes result in faster progress in training, they don't always converge fast; however, smaller batch sizes train slower but can converge faster.
- We have chosen two different optimizers which are SGD and Adam.
- For training purposes, we have used a validation set and also callbacks. These callbacks are used for managing the model training. Keras library has an EarlyStopping function which prevents the training from overfitting. This callback will stop the training when there is no improvement in the loss for twenty consecutive epochs, it is called patience. This number is

decided based on how the model learns. We have tried patience = 5 which made the model underfit so 20 is the best we have found for now.

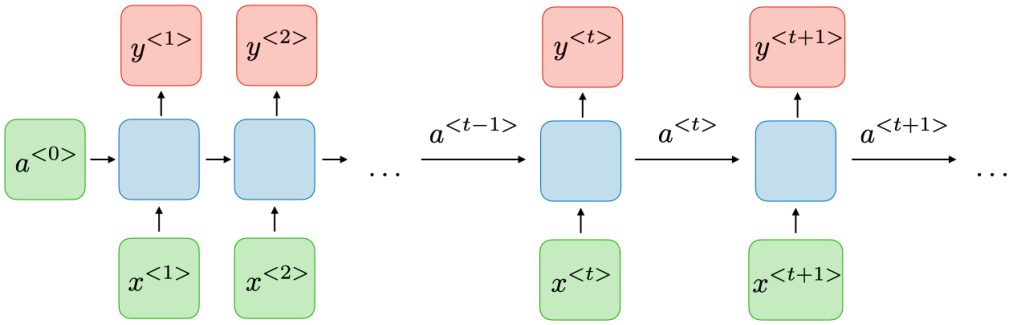
SVM

The Support Vector Machine is used for classification and regression purposes. It performs classification by constructing (N-1) dimensional hyperplanes which optimally separate the N-dimensional data space according to its labels, where N is the number of features. This classification occurs by linearly or nonlinearly separating the input feature surface. Its main purpose is to transform the original input set into a high-dimensional feature space by using a specific kernel function. With the help of this kernel function, optimum classification in the new feature space is achieved. In our case, we have used a linear kernel since our data space is high dimensional and linearly separable. Training being less costly was another motivation for this choice.



RNN

Recurrent Neural Networks (RNNs) have been used in speech recognition, as they outperformed traditional models. The reason for this performance lies in the architecture of the network itself. The difference between CNN and RNN is that RNN uses current input as well as what it has learned from previous inputs, while in CNN, the network has no memory of the previous input. Feedforward networks like CNN cannot handle sequential data, they consider only the current input, and cannot memorize previous inputs.



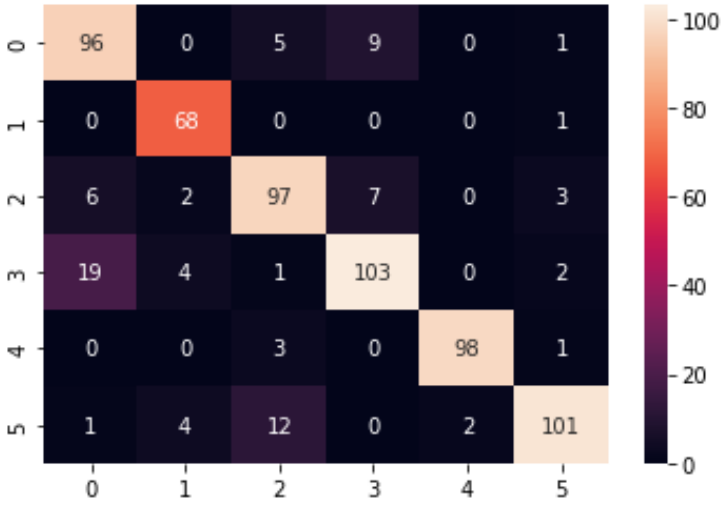
In the implementation, several methods were tried. The built-in method Sequential() in the Keras library was used to implement the Neural Network. Then, using the add() function, a simple RNN was added with 128 hidden units, a value chosen for SGD optimizer, and 35 hidden units, chosen for Adam optimizer; with a specified input size of 40. For RNN, the use of mel spectrogram was planned, however, trials have shown that the spectrogram size depended on the input voice length, and rescaling trials using methods such as Nearest Neighbor Interpolation and Box Sampling could not be optimized for use in different length input sequences. The resulting spectrograms could not be made to have the same shape, and since the difference between RNN and CNN approaches was specified as the main subject of interest, the MFCC coefficients were utilized.

4. Results

SVM Results

For the Support Vector Machine model, the Kernel function was chosen to be linear as mentioned. This is because it is computationally cheaper and performs sufficiently. Our data space has numerous features (40) and is linearly separable.

In the Google CoLab environment, training without scaling took approximately 16.5 seconds. When tested, it scored an accuracy of 87% and the following confusion matrix is the result.



Rows are the true labels, and columns are the predicted labels. Going from 0 to 5, indexes represent the emotions sad, angry, happy, disgust, neutral, fear respectively. But since an SVM classifier takes the distances between data points into account when constructing decision boundaries, features having different scales can disrupt the results.

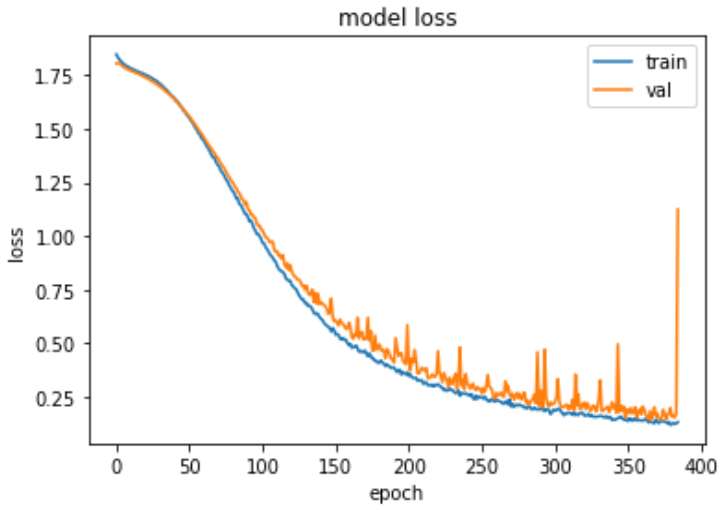
The data was scaled and another SVM classifier was trained on the scaled data. This time, the model performed a much higher accuracy of approximately 97%. Following is the confusion matrix resulting from this model, structured the same way. The labels are written in the following way: {0: sad, 1: angry, 2: happy, 3: disgust, 4: neutral, 5: fear}.

As training this model is computationally inexpensive, to make better use of our data, we applied cross-validation for performance evaluation. The data was separated into 5 folds and cross-validated. Following accuracies are obtained:

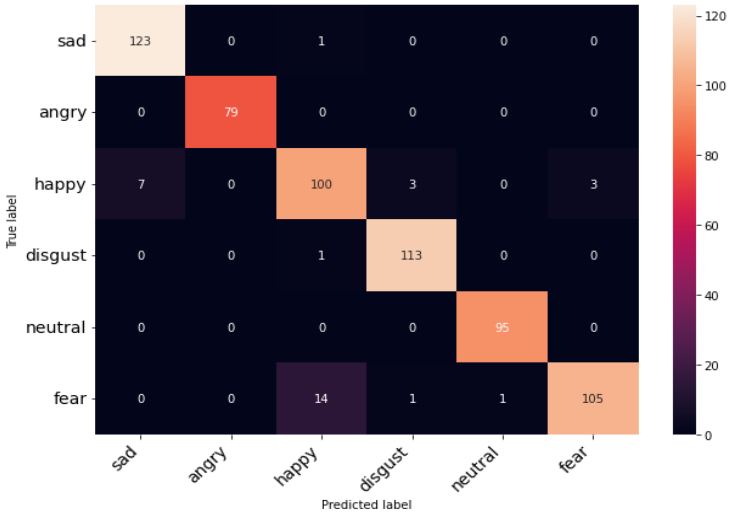
1. 0.96439628
2. 0.96594427
3. 0.97832817
4. 0.9627907
5. 0.9627907

CNN Results

When the optimizer is SGD, the loss plot and confusion matrix are stated below. After the epoch size becomes 385, the loss function converges and after the ~15th epoch, overfitting occurs. This can be interpreted from validation data's loss.

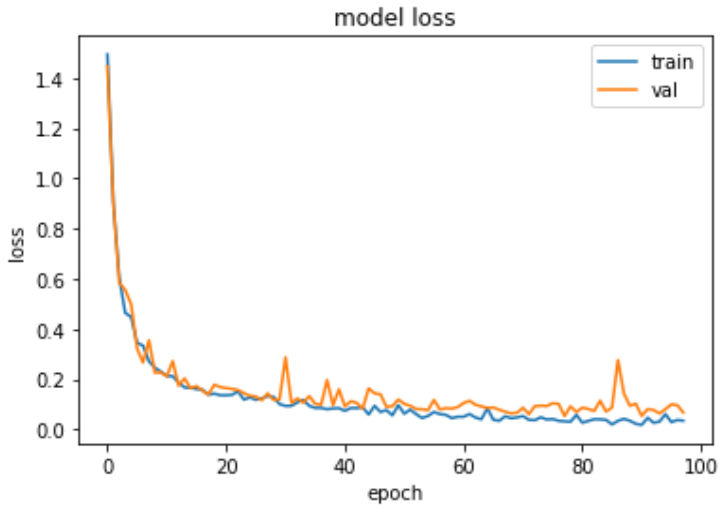


The below confusion matrix shows that the model performs very well on predicting angry and neutral emotions. However, there are some false predictions when the emotion is fear. The model has problems differentiating happy and fear emotions which is quite surprising since they are two distinct emotions.

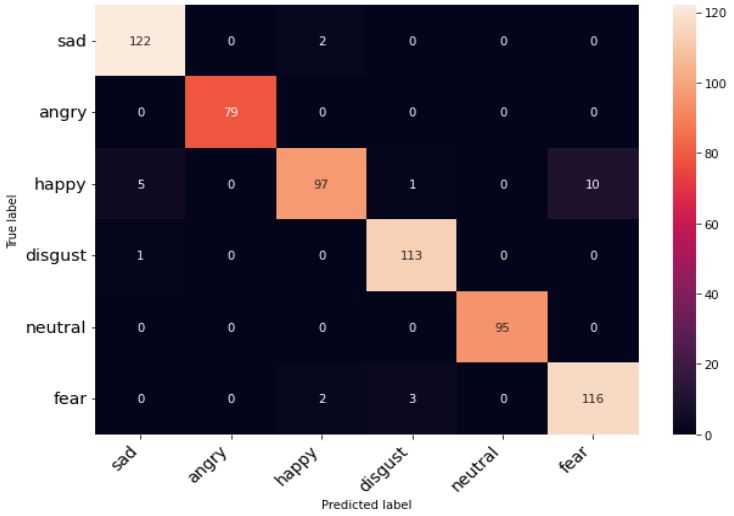


Performance metrics are stated below.
Accuracy: 0.9520
Precision: 0.9303405
Recall: 0.9724919
F Score: 0.95094925

When the optimizer is ADAM, the results are stated below. Compared to the SGD model, this model converges much faster, after 98 epochs.



The below confusion matrix shows that there are still some problems with differentiating happy and fear emotions. However, the model is good at labeling angry and neutral emotions as in the above model.



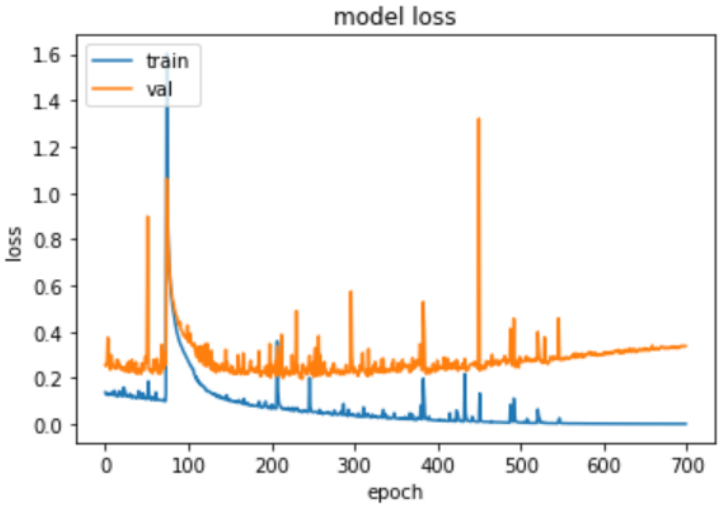
Performance metrics are stated below.
Accuracy: 0.9628
Precision: 0.9597523
Recall: 0.96723866
F Score: 0.9634809

In order to make a fair comparison between two CNN models, we have chosen the same hyperparameters. For instance, the learning rate is chosen as 0.0001. Since both optimizers have their own way to train the network, SGD made an early stop at the 385th epoch with a final validation loss of 1.1243 since the loss got bigger. However, in the Adam optimizer model, the training made an early stop at the 98th epoch with a final validation loss of 0.871.
In both of the loss plots, there are some parts that we saw overfitting. We can interpret this by saying that the model has learned the training dataset too well including the noise or random fluctuations in the training dataset. Especially, in the model we used SGD, validation loss decreases to a point but begins to increase partly. Although the Adam model also shows some increments in the validation loss, it is more of a good fit since the plot of validation loss decreases to a point of kind of stability and has small gaps with the training loss partly. So, comparing the two models, we can conclude that the Adam optimizer works and learns the pattern better.

RNN Results

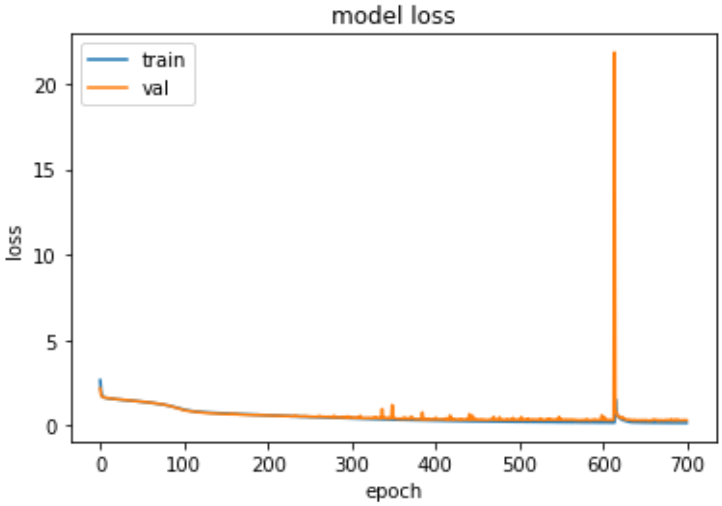
When the SGD optimizer was used, 128 hidden units were required in order to both stop underfitting and overfitting. The value given in the code was found via trial and error, and more hidden units resulted in overfitting,

which is characterized by the increase in the validation data accompanied by a decrease in loss when the loss is calculated using the training data. The characteristic overfitted model loss curves are shown below:



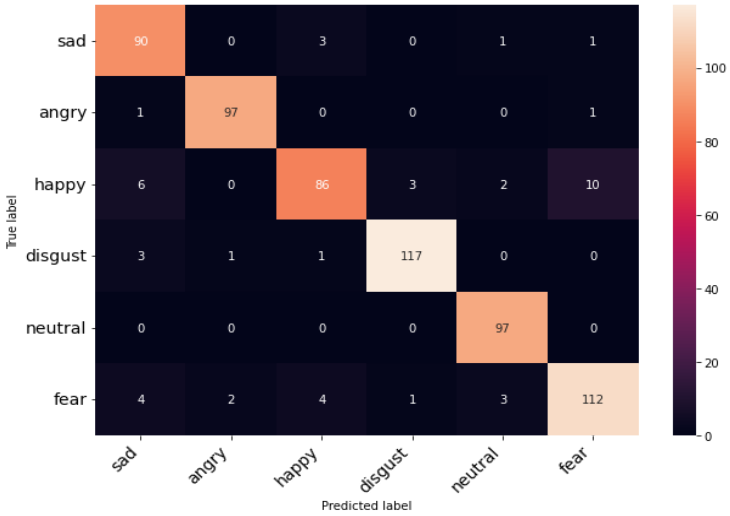
As one can see, the overfitting resulted in a loss of 0 calculated using the training set, while the validation set loss increased. This model had 256 hidden units. After noticing the overfitting curve, the hidden layer number was decreased until the overfitting problem was solved. Smaller hidden numbers were tried as well, but it resulted in the underfitting of the model, and the loss did not decrease as well, even after 700 epochs.

The final model loss data with the SGD optimizer and 128 hidden units are shown below:



As one can see, losses calculated using both the validation data and the training data decreased, indicating that the overfitting problem was solved.

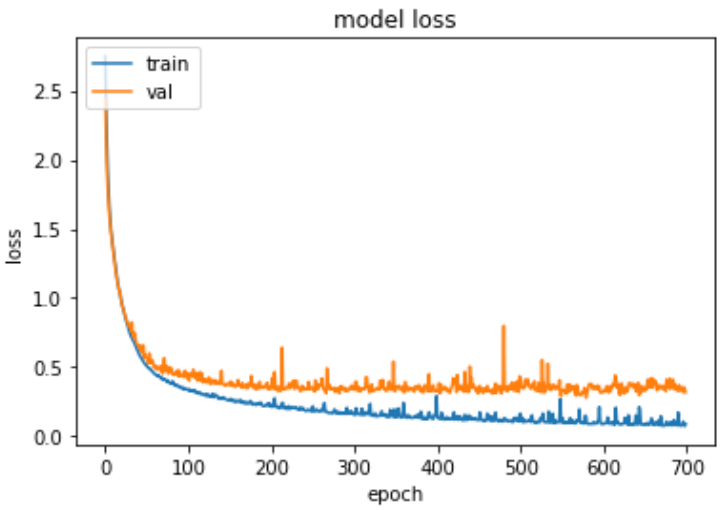
Then, the model was tested using a test data, and the confusion matrix was drawn:



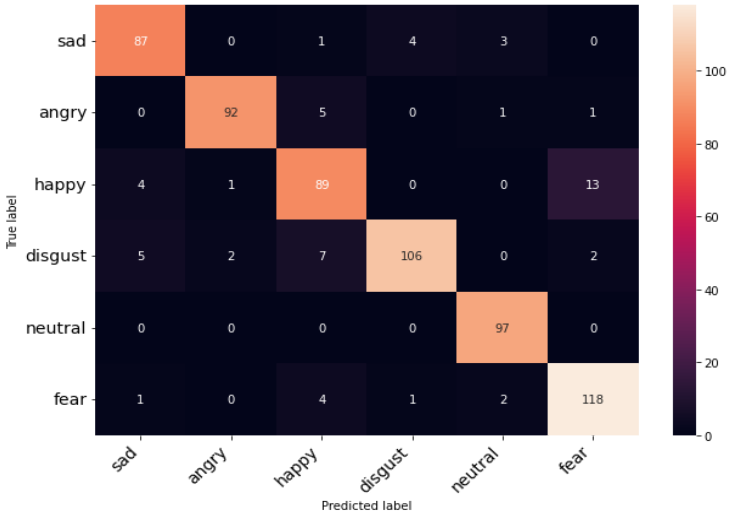
As one can see, similar to the CNN approach, the model had issues with distinguishing happy responses from fearful responses. However, 90.71% accuracy was obtained. The other performance metrics calculated using the test data are given as:

Precision: 0.9009
Recall: 0.9122
Fscore: 0.9065

Since the hidden number of parameters were high, training the model was time-consuming. This has resulted in a discussion on optimizer change. The new optimizer was chosen as the ADAM optimizer, after observing the convergence of the model in CNN. The model converged more quickly using the ADAM optimizer, however, the optimizer resulted in severe overfitting, even when 128 hidden units were used. Then, via trial and error, the hidden unit number was decreased and the optimal number of hidden units that did not result in both overfitting and underfitting was found to be 35. The loss curves that are found using both training and validation data is shown below:



The confusion matrix constructed using a seperate test set is given below:



The accuracy obtained from this model is 91.18%. The other metrics are given as:

Precision: 0.9040
Recall: 0.9182
Fscore: 0.9110

5. Discussion

In this project 3 different models are used; SVM, CNN, and RNN. In SVM, sklearn's SCV library is used and it is used with a linear kernel function. The model trained in 8 milliseconds which is quite fast however it didn't give the best

results. Its accuracy was 87.77% so, in order to get better metrics about the performance of the model; 5 cross-validations were applied. As a result; we got 0.89, 0.86, 0.88, 0.89 and 0.90 for each validation. To improve further, scaling was applied which resulted in a 97.2% accuracy. This was the best result compared to our other models. In CNN, we have used the keras library's Sequential model with 7 dense layers. In order to advance the model and build the hyperparameters, we have chosen two different optimizers which are SGD and Adam. Other hyperparameters like epoch no, batch size, or loss functions stayed the same. In the model with SGD optimizer, the training made an early stop with 385 epochs where the max epoch count was 700. It resulted in 95.2% accuracy. When the optimizer was Adam, the training made again an early stop with 98 epochs. This occurred because validation loss no longer decreased. Adam optimizer model gave 96.28% accuracy. Finally, in the RNN model, we have again used Keras library's Sequential model with a dense layer count 128 for the SGD optimizer and 35 for the ADAM optimizer. In this part, the optimizer choice directly influenced the structure of the model, as the ADAM optimizer was more efficient in learning the emotion content in the speech signals, and 128 hidden units resulted in an overfitting problem. To be able to compare CNN and RNN models, we have used the same hyperparameters and saw that CNN learned the pattern better compared to RNN. In three of the models, the accuracy metric is used. As a result, we ended up with the best accuracy results as 97.2% from the SVM classifier. This is because we applied further developments like cross-validation and scaling. Also, in three of the models, the models had problems with differentiating happy and fear emotions. Results show that with Adam optimizer, CNN was good at predicting angry and neutral, but it confused happy and fear. The model predicted happy 10 times when the true label was fear in 107 samples. With SGD optimizer, CNN was also good at predicting angry and neutral, and it confused fear. The model with SGD optimizer predicted happy 14 times, disgust and neutral 1 times, and predicted fear 105 times in 121 fear samples. SVM classifier was good at predicting angry and neutral but it had problems predicting fear. Also, the same confusion appeared in RNN. The best RNN model using the Adam optimizer predicted 13 times fear when the true label was happy in 97 samples. It is probably because both emotions have a sound with a high pitch and a high amplitude. However, the models are pretty good at understanding the difference between sad, angry, and neutral emotions. This is because these emotions have very distinct mel frequency coefficients.

Another result to discuss is the susceptibility of RNN to overfitting. Since RNN uses present inputs as well as past inputs, as the number of layers increases, the RNN overfits the training set, to the point where it reaches 0 loss in the training set. The results concerning the RNN overfitting problem and choosing the

number of hidden layers in the network was discussed under the results title. The choice of optimizer greatly affected the number of hidden parameters, as the Adam optimizer was more optimal at the task at hand, the hidden layer number needed to prevent overfitting while also preventing underfitting was greatly reduced.

6. Conclusion

Scaling improved accuracy of SVM model with cross-validation. Hyperparameters were chosen as the same in two models for CNN to make a comparison between Adam and SGD optimizers. Adam optimizer converged faster when compared to SGD as optimizer. Larger batch sizes resulted in faster training, but they do not always converge fast when compared to smaller batch sizes. While CNN has no memory of the previous input, RNN uses current input as well as its learnings from previous input.

All of our models had confusion between happy and fear emotions. This confusion could be arisen from the similar amplitude and pitch when these emotions are present.

We don’t consider the external factors that can affect the model such as the gender or age of the sound so, in the future, these factors can be considered. Also, there are lots of speech datasets and they can be used to train the model which makes an improvement. Furthermore, having more features may help the training to make the differentiation between emotions healthier.

7. Appendix

Funda Tan: CNN, and conclusion parts.

Lara Fenercioğlu: CNN, introduction and discussion parts.

Yekta Seçkin Satır: SVM

Latife Sezin İsmailoğlu: RNN, discussion, and conclusion parts.

Eren İhsan Erzin: MFCC

extract_feature function:

```
def extract_feature(file_name, mfcc):
    X, sample_rate = librosa.load(os.path.join(file_name), res_type='kaiser_fast')
    if mfcc:
        mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,
        axis=0)
        result=np.hstack((mfccs))
    return result
```