Emotion Recognition Using Speech

March 14, 2021

Note: Please use Jupyter Notebook for best viewing experience. Some things in the file does not work with Google Colab.

0.1 # *Emotion Recognition Using Speech*

0.1.1 An experimental study by

Subodh Bijwe (2019AIM1011) and Gulshan Sharma (2019CSZ0004)

Abstract: Recently, with availability of high computation capabilities and advancements in the area of machine learning, attention to the emotion recognition through speech signal research has been boosted in human machine interfaces. Various theoretical and experimental studies are governed till now, to identify the emotional state of a person through examining speech signals. Preparation of an appropriate dataset, selection of suitable and promising features, designing proper classification methods are the main key issues of an speech emotion recognition system. This document demonstrates the collective work done by us to fullfill the CS-503 course project criteria. All experimentation is performed in a python 3.7 environment.

---

1 Table of Contents

1. Introduction

- Emotions Modelling
- Related Work
- Methodology
  1. Datasets
     1. RAVDESS
     - SAVEE
     - TESS
  2. Libraries Used
  3. Our Approtch
     1. Exploring Data
     - Defining Functions
     - Augmentation Methods
       1. **Static Noise**
       * **Shift**
       * **Pitch**
       * **Dynamic Change**

        * **Speed and Pitch**
    - Conductiong Experiments
    - Printing Results
        1. **Validation accuracy and validation loss plots**
        2. **Accuracy on test data**
        3. **Precision**
        4. **Recall**
        5. **F1-score**
        6. **Support**
        7. **Confusion Matrix**
- Experiments and Results
    1. Test the models trained on RAVDESS on the same dataset (Randomized split)
    2. Test the models trained on RAVDESS on the combined dataset
    3. Train the models on combined dataset and check for the performance on the same
    4. Now we experiment with the combined data with some augmentation methods
    5. Test the models trained on RAVDESS on the same dataset (Specified split)
    6. Experiments with 2D CNN Architecture
- Discussions
- Summary
- Future Scope
- References
- Code

1.1 Introduction

Recognition of human emotions has always been a topic of interest for data scientists. Multiple sources like facial expressions, movement of eyes, tone of speech can be used to detect emotions. We have chosen speech among these. However, humans also have habit of hiding their true emotions. This study does not consider that and believes humans always show their true emotions while speaking.

The idea behind this study is to see what, among the available methods are better for classification of emotions from audio files. Audio files are different than image files. It requires experienced person and cost to attach a categorizing label to an audio file after completely listening to it unlike images where it can be easily classified as dog or cat based on the what the image contains. And then using that data to train a model and see how it performs on unheard audio, that sparked our interest in this domain.

In this study we have done multiple experiments and checked what settings perform better on unheard audio files.

 Image Credit: depositphotos

Back to Table of Contents

1.2 Emotions Modelling

Modelling human emotions is a hard problem, Out of all the models proposed by various researchers, two methods are very popular for modelling emotions. First approach is to label the emotions in discrete categories, like Joy, Sadness, Surprise, Anger, Love and Fear etc. Though this modelling method provides separate discrete target classes, the problem with this model is that a given expression/stimuli may consist of blended emotions and the model is not adequate to express. Second approach is to have higher dimensional continuous scales to categorize emotions. ie Instead of choosing discrete target classes to categorize emotions, impression of each stimulus is indicated on continuous scales. Some examples for these continuous scales are attention-rejection, pleasant-unpleasant, simple-complicated scales etc. Out of these many scales valence and arousal is most common and famous. Here in this scale, pleasantness and unpleasantness of a stimulus is represented by positive valence and negative valence values respectively. Arousal is the second dimension and it represents the activation level of the stimuli. Since both valence and arousal are continuous scales, hence we can represent different emotional levels on a two dimensional plane with arousal and valence axis.

Back to Table of Contents

1.3 Related Work

As the saying goes, ‘standing of the shoulders of giants’, we would like to appreciate the related work done by the following mentioned. Their work not only helped us in building these experiments but also gave us deep insights on the working of the data and laid a foundation on which our study stands.

- Cao et al. [1] has proposed a ranking SVM method to solve the problem of binary classification for synthesize information about emotion recognition. This ranking strategy, treats data from every speech utterer as a seperate query then instruct SVM algorithms to mix all predictions from rankers to apply multi-class prediction. Ranking SVM approtch has two main advantages, first, for speaker-independent it obtains speaker specific data during training and testing steps. Second, Since each speaker can express mixture of multiple emotions, So this inturions is considered in this approtch to recognize the dominant emotion. It is observed that, this Ranking approach has achieved substantial gain in terms of accuracy compare to conventional SVM approtch.

- Nwe et al. [2] has proposed a complete new system for emotion classification of utterance signals. Proposed system employ a short time log frequency power coefficients (LFPC) as a feature vector to characterize the speech signals. and discrete HMM is used as classifier. This method is able to classify the emotions into six different categories. To train and test this newly proposed system, Author has used his private dataset. LFPC coefficients are is compared with the MFCC coefficients and LPCC coefficients. Result of this experimentaion shows that model is able to achieve classification accuracy of 78% and 96% in average and best case respectively. Also, It is observed that LFPC coefficients as a feature is a better option as compared to the standard features used for emotion classification.

- Chen et al. [3] has used three level speech emotion recognition method to classify various emotions from coarse to fine then Fisher rate is used to select appropriate features. This output feature from fisher rate is used as a input parameters for a multi-level SVM classifier. After that principal component analysis (PCA) is employed to reduce the dimensionality of feature space. Furthermore artificial neural network (ANN) is used for classification of four comparative experiments which include Fisher + SVM, PCA + SVM, Fisher + ANN and PCA + ANN. This experimentaion shows that dimension reduction Fisher is better than PCA for classification.

- Rong et al. [4] has proposed an ensemble random forest method with a high number of features for emotion recognition. Author has not referred any language, hence it remains an unclosed problem. This ensemble random forest on tree method is applied on a dataset cosistes of fewer instances but with high dimensional feature. This method achieved improvement on emotion recognition rate when evaluated on a Chinese emotional speech dataset. Furthermore, It is observed that ensemble random forest on tree method performs better than popular feature reduction methods like PCA, multi-dimensional scaling and recently developed ISOMap. Author has mentioned the best accuracy rate of 82.54% with 16 dimensional features in the female dataset, while 16% on 84 dimensional features is the worst case scenerio on natural data set.

- Wu et al. [5] has proposed a fusion-based method for speech emotion recognition. Author has employed acoustic-prosodic features and semantic labels on multiple classifier. Proposed fusion method consists of extracting acoustic-prosodic features first, Then three different types of base-level classifier are used. These classifiers are GMMs, SVMs, MLP and Meta decision trees. The maximum entropy model in the semantic labels method is used extract the the association information amoung the emotion association rules and emotion states in emotion recognition. Finally, the integrated information from the semantic labels based and acoustic-prosodic based models are utilized to define the emotion recognition outcome in the final state. This experimentation is done on a private dataset and it shows the performance based Accuracy parameter as follows. 80% on MDT archives, 80.92% on SL-based recognition archives, and 83.95% on the mixture of semantic labels based and acoustic-prosodic model.

- Narayanan [6] has proposed a domain-specific emotion recognition system using speech signals collected from the call centers. Author's main research focus is on detecting negative and non-negative emotions and using acoustic, lexical, and discourse features for emotion recognition. K-NN and linear discriminant classifiers are used and experimental results confirm that the best results are obtained by combining both acoustic and language data. By combining three information sources together, classification accuracy is increased by 40.7% for males and 36.4% for females, instead of using single information source. This research tells combinig multiple information sources is better for a robust emotion recognition system.
- Yang & Lugger [7] has proposed a new set of harmony features used for emotion recognition in speech signals. These features are inspired from the psychoacoustic perception from music theory. Firstly calculating predicted pitch of a speech signals, then computing spherical autocorrelation of pitch histogram. Cause of a harmonic or inharmonic impression is computed by calculating the extent of dissimilarity two pitch durations. Bayesian classifier, with a Gaussian class-conditional likelihood is used in the classification step. Experimental result by using harmony features in Berlin emotion database indicates an improvement in average recognition rate by 2%.
- This study is highly inspired by Mitesh Puthran and the research given in his Speech Emotion Analyzer repository. We tried to replicate the results but he however predicted the results of live voices. Due to lack of resources we were unable to do that which is why we decided to compare on data found in different datasets. It will be mentioned in the next section briefly.
- Eu Jin Lok has also provided us with simple and easily understandable application of the same in Audio Emotion Series.
- Speech Emotion Recognition with Convolutional Neural Network by Reza Chu was also a part of our experiments.
- Last but not the least, we would like to thank the authors of the following datasets used in the experiments,
    1. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)
    2. Surrey Audio-Visual Expressed Emotion (SAVEE) Database
    3. Toronto emotional speech set (TESS)

Back to Table of Contents

1.4 Methodology

1.4.1 1. DataSets

As mentioned above 3 different datasets were used for the experiments.

A. RAVDESS: The RAVDESS dataset is a collection of 7356 files. The database contains speech and songs by 24 actors (12 male and 12 female). Speech includes different emotions such as calm, happy, sad, angry, fearful, surprise, and disgust. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: Audio-only (16bit, 48kHz .wav), Audio-Video (720p H.264, AAC 48kHz, .mp4), and Video-only (no sound). Also, there are no song files for Actor_18.

- **Audio-only files** ##### Audio-only files of all actors (01-24) are available as two separate zip files (~200 MB each):
- Speech file (Audio_Speech_Actors_01-24.zip, 215 MB) contains 1440 files: 60 trials per actor x 24 actors = 1440.
- Song file (Audio_Song_Actors_01-24.zip, 198 MB) contains 1012 files: 44 trials per actor x 23 actors = 1012. We won't be using the song files, instead our work will revolve around the speech files.
- **File naming convention**

Each of the 7356 RAVDESS files has a unique filename. The filename consists of a 7-part numerical identifier (e.g., 02-01-06-01-02-01-12.mp4). These identifiers define the stimulus characteristics:

- *Filename identifiers*
  - Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
  - Vocal channel (01 = speech, 02 = song).
  - Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
  - Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
  - Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
  - Repetition (01 = 1st repetition, 02 = 2nd repetition).
  - Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).
- *Filename example:* 02-01-06-01-02-01-12.mp4
  - Video-only (02)
  - Speech (01)
  - Fearful (06)
  - Normal intensity (01)
  - Statement "dogs" (02)
  - 1st Repetition (01)
  - 12th Actor (12) (Female, as the actor ID number is even.)

B. SAVEE: The SAVEE database was recorded from four native English male speakers (identified as DC, JE, JK, KL), postgraduate students and researchers at the University of Surrey aged from 27 to 31 years. Emotion has been described psychologically in discrete categories: anger, disgust, fear, happiness, sadness and surprise. This is supported by the cross-cultural studies of Ekman and studies of automatic emotion recognition tended to focus on recognizing these. We added neutral to provide recordings of 7 emotion categories. The text material consisted of 15 TIMIT sentences per emotion: 3 common, 2 emotion-specific and 10 generic sentences that were different for each emotion and phonetically-balanced. The 3 common and 2 $\times$ 6 = 12 emotion-specific sentences were recorded as neutral to give 30 neutral sentences. * This resulted in a total of 120 utterances per speaker, for example: * Common: She had your dark suit in greasy wash water all year. * Anger: Who authorized the unlimited expense account? * Disgust: Please take this dirty table cloth to the cleaners for me. * Fear: Call an ambulance for medical assistance.

* Happiness: Those musicians harmonize marvelously. * Sadness: The prospect of cutting back spending is an unpleasant one for any governor. * Surprise: The carpet cleaners shampooed our oriental rug. * Neutral: The best way to learn is to solve extra problems. * The original SAVEE dataset has 4 speakers but we have bundled all of them into one single folder and thus the first 2 letter prefix of the filename represents the speaker initials. Eg. 'DC_d03.wav' is the 3rd disgust sentence uttered by the speaker DC. It's worth nothing that they are all male speakers only. To balance it out with we also used the TESS dataset which is just female only.

C. TESS: There are a set of 200 target words were spoken in the carrier phrase "Say the word _' by two actresses (aged 26 and 64 years) and recordings were made of the set portraying each of seven emotions (anger, disgust, fear, happiness, pleasant surprise, sadness, and neutral). There are 2800 data points (audio files) in total. The dataset is organised such that each of the two female actor and their emotions are contain within its own folder. And within that, all 200 target words audio file can be found. The format of the audio file is a WAV format.

`# This is formatted as code`

Back to Table of Contents

1.5 2. Libraries Used

1. Numpy
2. Pandas
3. Librosa
4. Keras
5. Sklearn
6. Matplotlib
7. Tensorflow

Back to Table of Contents

1.6 3. Our Approach

-

  ### 1.6.1 Exploring Data

  - Paths for each files are saved in their dataframes.
  - Some samples are tested for different emotioins, a waveplot is also plotted to visualise the pitch in each sample.
  - We also used MFCCs (Mel Frequency Cepstral Coefficients). The mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10-20) which concisely describe the overall shape of a spectral envelope. In MIR, it is often used to describe timbre.

-

1.6.2 Defining Functions

- We defined some functions to perform some basic tasks like train test splitting and data pre-processing.
- Functions to build models are also defined which can be called in experiments.
- After the model training phase we saved the models and to predict results from these models, we defined another function.

•

1.6.3 Augmentation Methods

1. Static Noise

- Add static noise in the background of the audio file.

2. Shift

- Shift the audio file tiny bit to the left or right direction.

3. Pitch

- We use this method to stretch the audio, because of which the duration gets longer as well as the audio wave gets stretched too.

4. Dynamic Change

- We did some dynamic change in the original audio file.

5. Speed and Pitch

- As the name suggests, we change the speed and pitch at the same time.

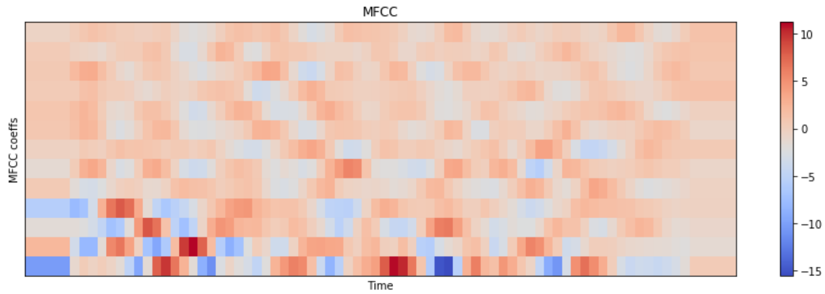
•

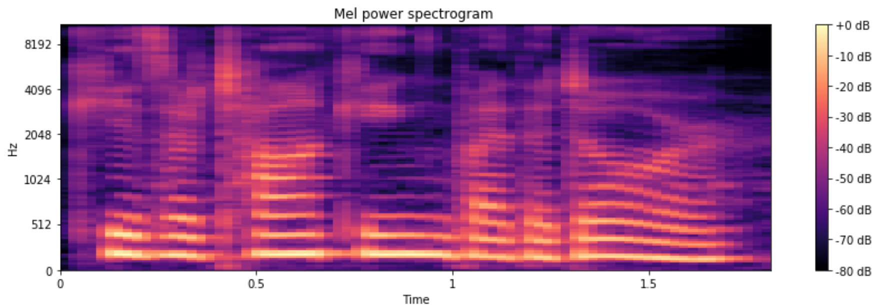
1.6.4 Conducting Experiments

- For conducting each experiment a CNN model is used and is trained on specific data called as training data and tested on data called as testing data.
- In experiments 1-5 an 1D CNN model is used and for 6th experiment 2D CNN model is used. These models can be easily found in the articles mentioned in the Related Work section.
- For extracting features from the audio files we have used two features from the librosa library. We consider these two as black box and focused on the features extracted after using them.
- **Mel Scale:** The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear. Noted from:- MFCC Tutorial

$$M(f) = 1125 \ln(1 + \frac{f}{700}) \tag{1}$$

* First is **MFCCs or Mel-Frequency Cepstral Coefficients**. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the *phoneme* being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.
* Second is **Mel Spectogram**. In short, Mel Spectrogram is a Spectrogram with the Mel Scale as its y axis.





* ***The above mentioned images are the MFCC and Mel Spectogram, respectively, of female actor no. 18 saying "Kids are sitting by the door".***

– In each of the experiments normalization is performed on the data, learning rate is set to 0.00001 and loss is taken as categorical_crossentropy.

– We also have categorised the results we got into two major sub-experimentation trials:
  * We have combined the predictions based on **gender** and printed results for **gender classification** done by the models.
  * We have combined the predictions based on **emotions** and printed results for **emotion classification** done by the models.

•

1.6.5 Printing Results

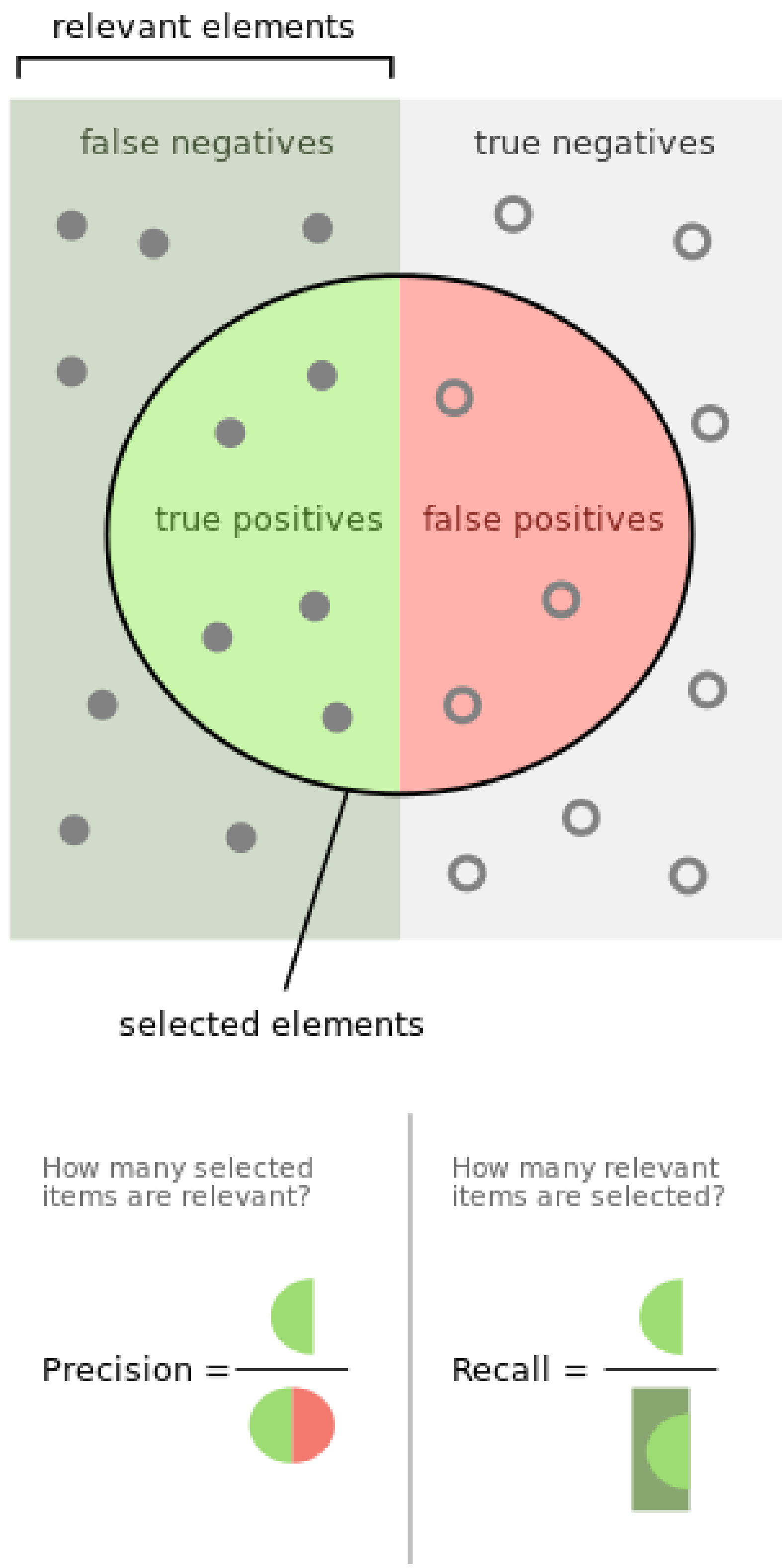
– The results we have evaluated for all experiments are:
  1. **Validation accuracy and validation loss plots** are as the name suggests, plots which shows how the accuracy or loss (on the y-axis) varies as number of epochs (on the x-axis) increases.
  2. **Accuracy on test data** is the ratio of number of correct predictions to the total number of input samples.
  3. **Precision** is the ability of a classiifer not to label an instance positive that is actually negative. For each class it is defined as as the ratio of true positives to the sum of true and false positives. Said another way, "for all instances classified positive, what percent was correct?"

* According to our experiments if we consider a class $emotion_1$. The precision for this $emotion_1$ class is the number of correctly predicted $emotion_1$ audio files out of all predicted $emotion_1$ audio files.

$$Precision_{emotion_i} = \frac{(TruePositives)_{emotion_i}}{(TruePositives + FalsePositives)_{emotion_1}} \tag{2}$$

4. **Recall** is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. Said another way, "for all instances that were actually positive, what percent was classified correctly?"
   * According to our experiments if we consider a class $emotion_1$. The recall for this $emotion_1$ class is the number of correctly predicted $emotion_1$ audio files out of all actual $emotion_1$ audio files.

$$Recall_{emotion_i} = \frac{(TruePositives)_{emotion_i}}{(TruePositives + FalseNegatives)_{emotion_1}} \tag{3}$$



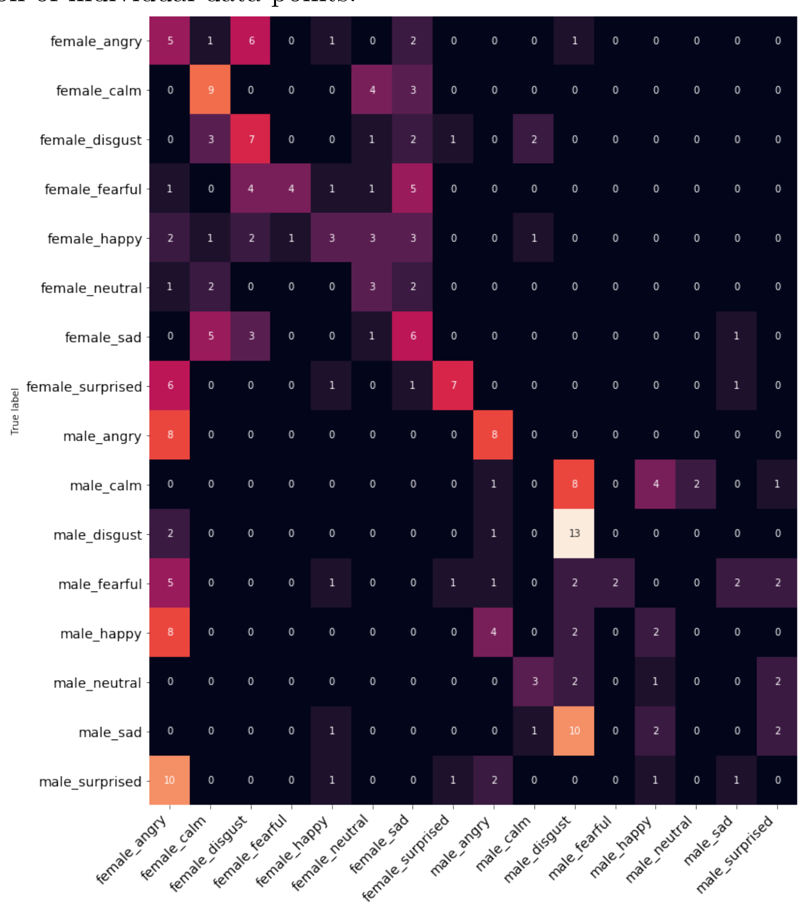
∗ Image source Wikipedia

5. **F1-score** is a weighted harmonic mean of precision and recall such that the best

score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy. The scores corresponding to every class will tell you the accuracy of the classifier in classifying the data points in that particular class compared to all other classes.

$$F1 - score = \frac{2 \times (Precision_{emotion_i} \times Recall_{emotion_i})}{(Precision_{emotion_i} + Recall_{emotion_i})} \tag{4}$$

6. **Support** is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.
7. **Confusion Matrix** takes a fitted scikit-learn classifier and a set of test X and y values and returns a report showing how each of the test values predicted classes compare to their actual classes. Data scientists use confusion matrices to understand which classes are most easily confused. These provide similar information as what is available in a ClassificationReport, but rather than top-level scores, they provide deeper insight into the classification of individual data points.



   * Example of Confusion Matrix:

– In almost all the experiments we have evaluated these parameters for the test data, test data classified based on genders and test data classified based on emotions. This gave us an insight on how the models are performing in classifying the unheard audio files into proper gender and proper emotions.

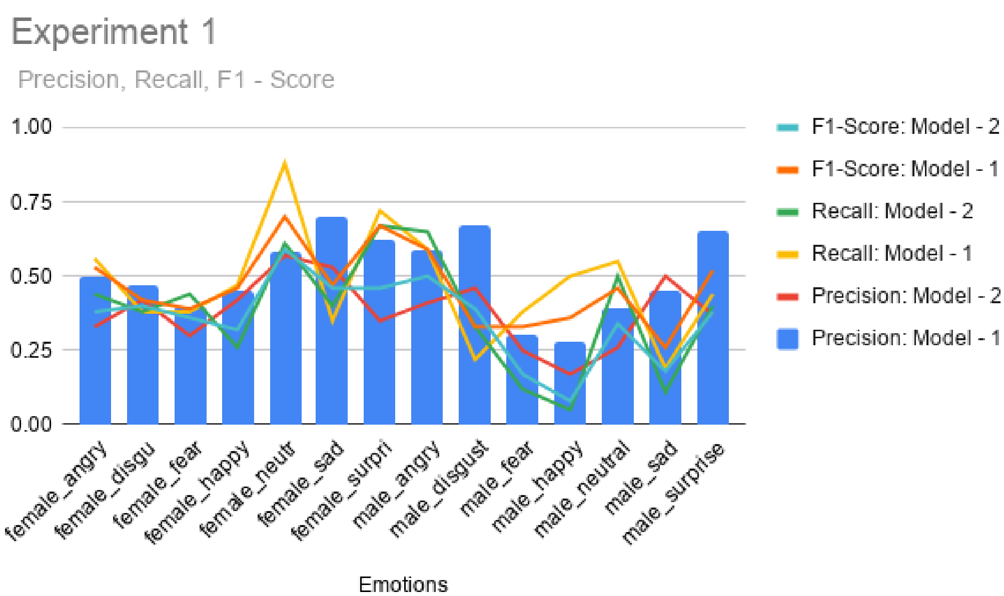
These values are generated by using 'classification_report' metric of sklearn, this and this blog clearly explains all of them.

Back to Table of Contents

1.7 Experiments and Results

The first five experiments are for 1D CNN architecture while the 6th(and last experiment) is for 2D CNN architecture. The data used for plotting the graphs is taken from the results in each section and is properly available in **this link**. Most of the graphs are self-evident, in the sense we can easily deduct how the models have performed on what dataset with how much of precision, recall and f1-score. The accuracies are mentioned seperately and gender and emotion based classification can be found in their particular section of the experiment in which the sub-experiment is conducted. ## 1. Test the models trained on RAVDESS on the same dataset (Randomized split) * In this experiment we trained both of our models on RAVDESS dataset. We plot both the Validation Accuracy and Validation Loss for both the models (This has been done after every model training step). * We saved the models for future use and then load them for testing on the RAVDESS test set.
* Model 1 Results: * The accuracy is **48.26%** * Model 2 Results: * The accuracy is **38.54%**

- The precision, f1-score, recall and confusion matrix can be found in respective sections.

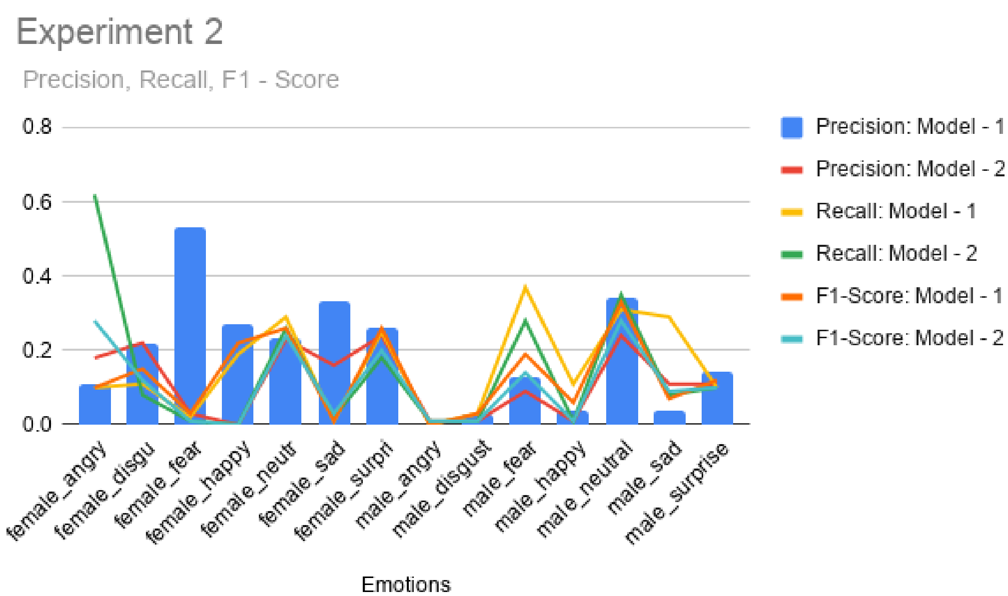


- As further we explored the results we group together the results in two genders, male and female, and we found that the accuracy we get is **90.625%**. This means our model was capable of distinguishing between male and female voices very properly.
- And in another trial of exploration we look for emotion classification, where we got accuracy of **64.12%**.

- This concludes our first experiment.

1.8 2. Test the models trained on RAVDESS on the combined dataset

- Remember the models we have trained on the RAVDESS dataset? Now we test them on the combination of all the datset.
- For this experiment we have combined the RAVDESS, SAVEE and TESS dataset.
- We had not expected high results from this experiment as the training set is very small and does not represent whole distribution of dataset. Still for the sake of experimentation we see how the models performed.
  - Model 1 Results:
    - The accuracy is **15.21%**
  - Model 2 Results:
    - The accuracy is **15.97%**
- The precision, f1-score, recall and confusion matrix can be found in respective sections.

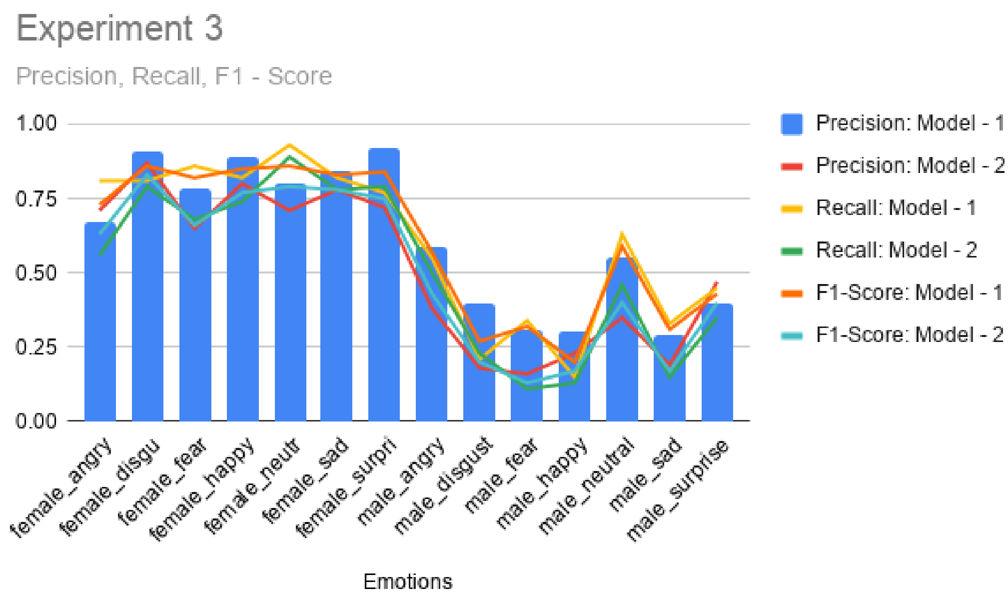


- This concludes our second experiment.

1.9 3. Train the models on combined dataset and check for the performance on the same

- This is a major experiment because we will be using all the datasets to train the two models defined in earlier experiments and also for the testing purpose.
- We have expected high accuracy from this experiment as now the training data has lots of variation and so does the testing data, and the models did perform well.
  - Model 1 Results:
    - The accuracy is **72.10%**
  - Model 2 Results:

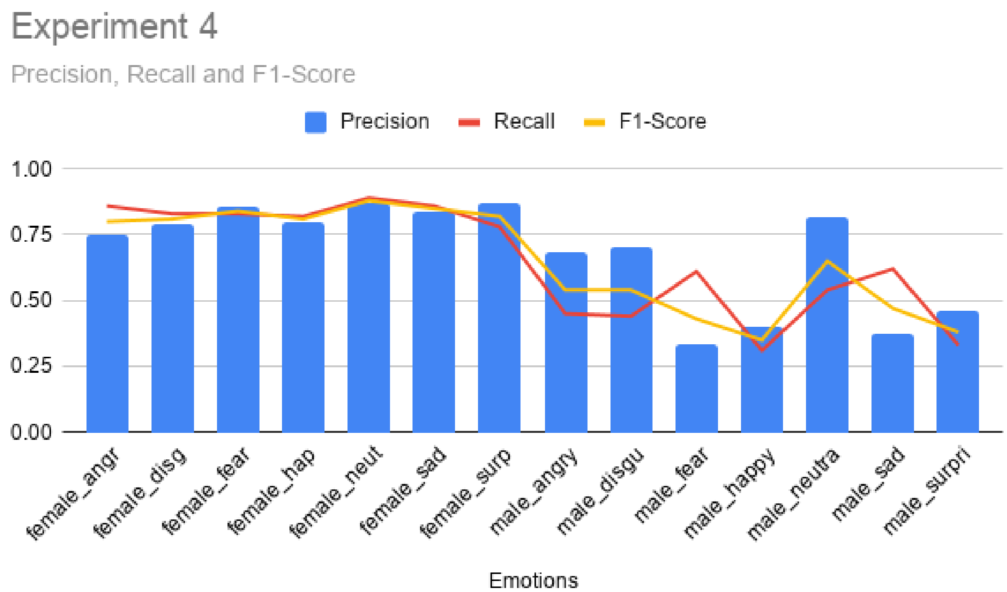
        * The accuracy is **62.78%**
- The precision, f1-score, recall and confusion matrix can be found in respective sections.



- As further we explored the results we group together the results in two genders, male and female, and we found that the accuracy we get is **97.52%**. This means our model was capable of distinguishing between male and female voices very properly.
- And in another trial of exploration we look for emotion classification, where we got accuracy of **64.12%**.
- This concludes our third experiment.

1.10 4. Now we experiment with the combined data with some augmentation methods

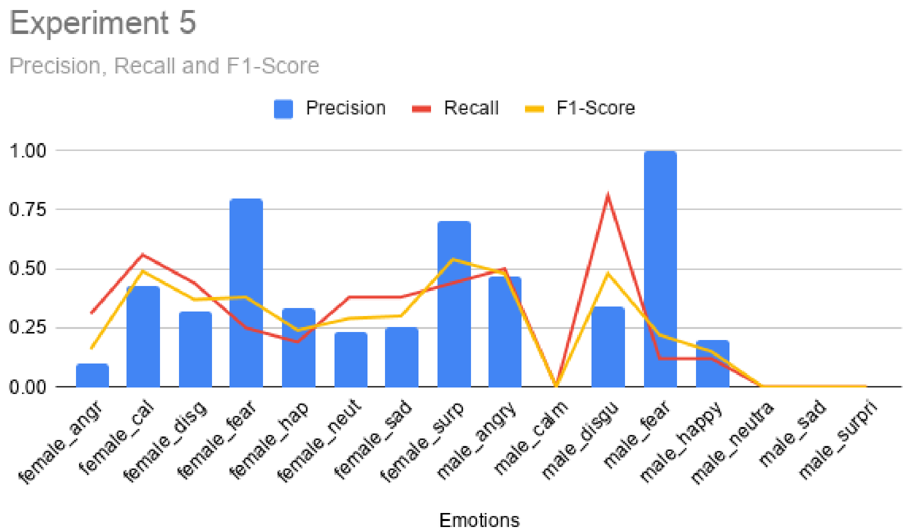
- Remember the augmentation methods we defined earlier? We used them in this experiment.
- What we did was, we first explored the methods, saw how each of the methods affects a random sound file, and for preprocessing step we applied noise, speed and pitch to all the audio files in combined dataset.
- So now we have a dataset with original sounds, sounds with noise and sounds with augmented speed and pitch.
- We did the basic preprocessing like test-train splitting, normalization, expanding dimensions so we have data ready for our models.
- As in the previous experiment model 1 performed better so we choose only that model to work on in this experiment.
    - Model Results:
        * The accuracy is **74.66%**
- The precision, f1-score, recall and confusion matrix can be found in respective sections.



- As further we explored the results we group together the results in two genders, male and female, and we found that the accuracy we get is **97.20%**. This means our model was capable of distinguishing between male and female voices very properly.
- And in another trial of exploration we look for emotion classification, where we got accuracy of **75.53%**.
- This concludes our fourth experiment.

1.11 5. Test the models trained on RAVDESS on the same dataset (Specified split)

- This was more of a curiosity experiment. We wanted to see how will a model perform if we train it on some male and female sound files and test on other files. This experiment might look similar to the first experiment but it is different in a specific way.
- In randomized case we had no choice on which data we are allocating for training and which for testing, but in this case we wanted to have that freedom, which is why we trained our model on first 20 actors in RAVDESS dataset and tested them on the remaining dataset.
- RAVDESS is a dataset in which alternate actors are male and female, so 24 actors means 12 male actors and 12 female actors. We took 20 of them for training i.e. 10 male and 10 female and similarly 2 male and 2 female for testing purposes.
- So we set aside the testing data i.e. the 4 actors(Actors 21-24).
- We did some preprocessing on the training data and also added data with noise and changed pitch in along with the original dataset (We haven't done this in the first experiment).
  - Model Results:



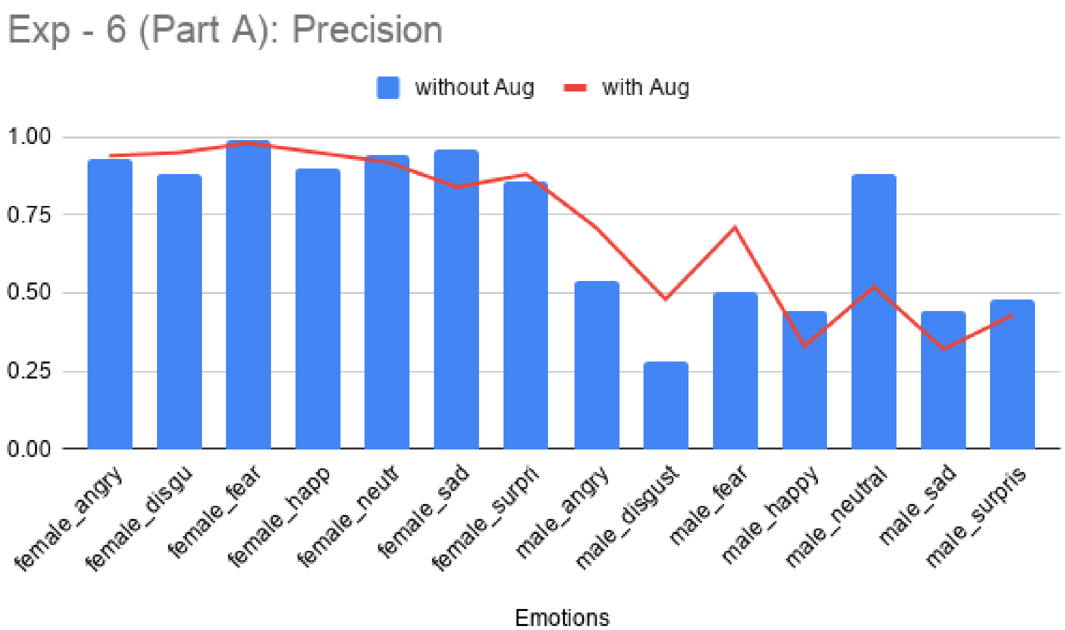
  * The accuracy is **28.75%**

- The precision, f1-score, recall and confusion matrix can be found in respective sections.
- As further we explored the results we group together the results in two genders, male and female, and we found that the accuracy we get is **81.66%**. This means our model was capable of distinguishing between male and female voices very properly.
- And in another trial of exploration we look for emotion classification, where we got accuracy of **32.91%**.
- This concludes our fifth experiment.

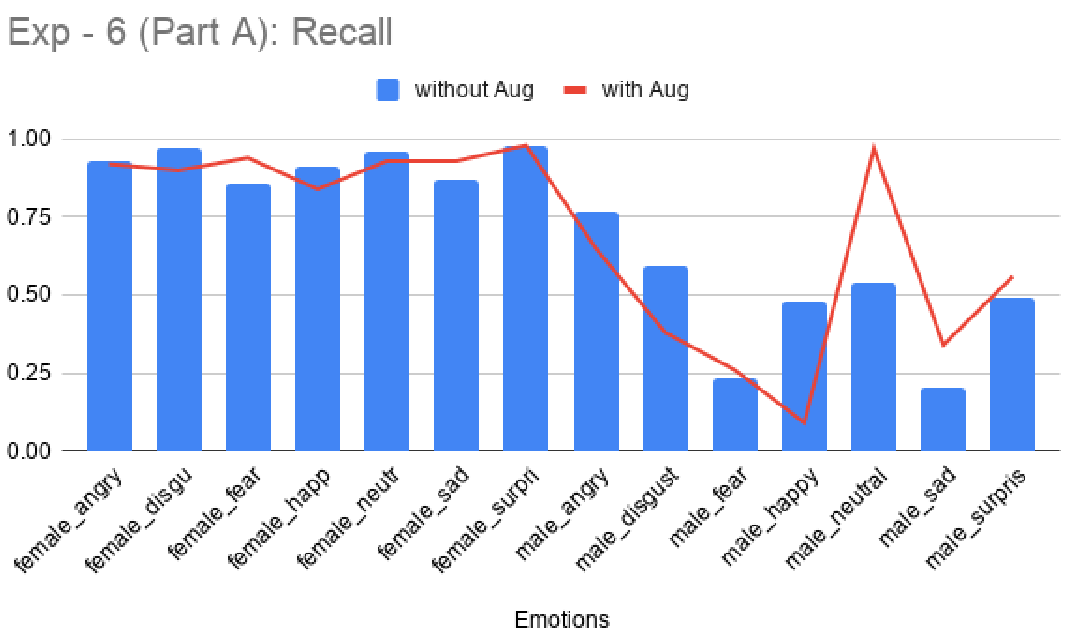
1.12 6. Experiments with 2D CNN Architecture

- As mentioned earlier in this experiment we have explored the same performance measures on combined data with 2D CNN model.
- This experiment is divided into 4 different sections. We will explore two different features provided by Librosa library, viz, mfcc and mel spectogram. In earlier experiments we only have used the mfcc feature, now we will use both with and without augmentation on 2D CNN.

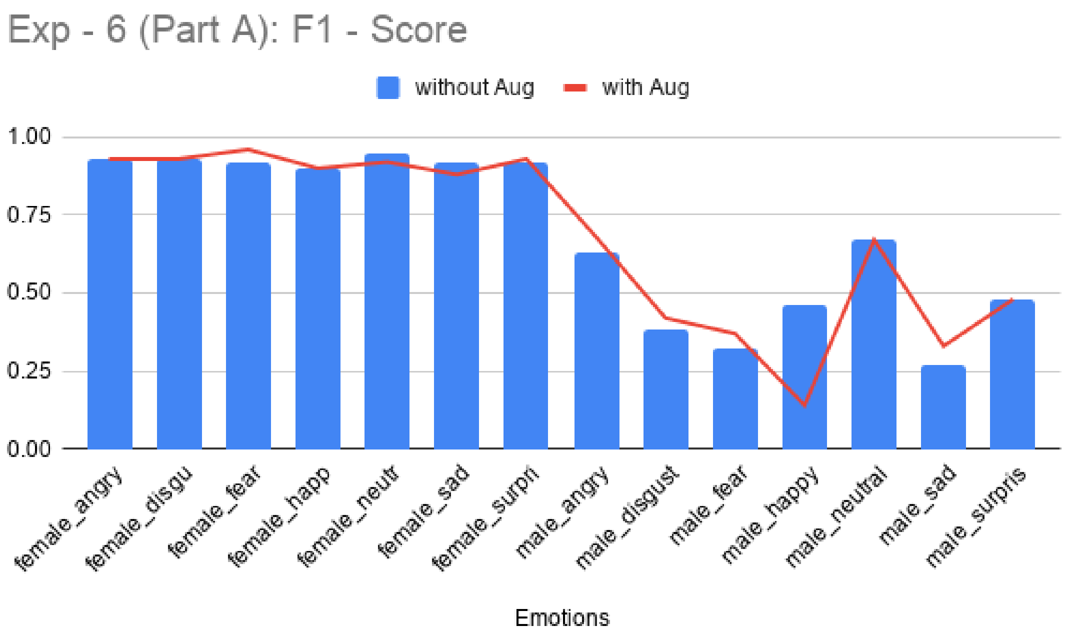
Part: A 1. MFCC * Model Results: * The accuracy is **80.76%** * Gender classification accuracy is **99.49%** 2. MFCC with Augmentation * Model Results: * The accuracy is **80.93%** * Gender classification accuracy is **99.23%**



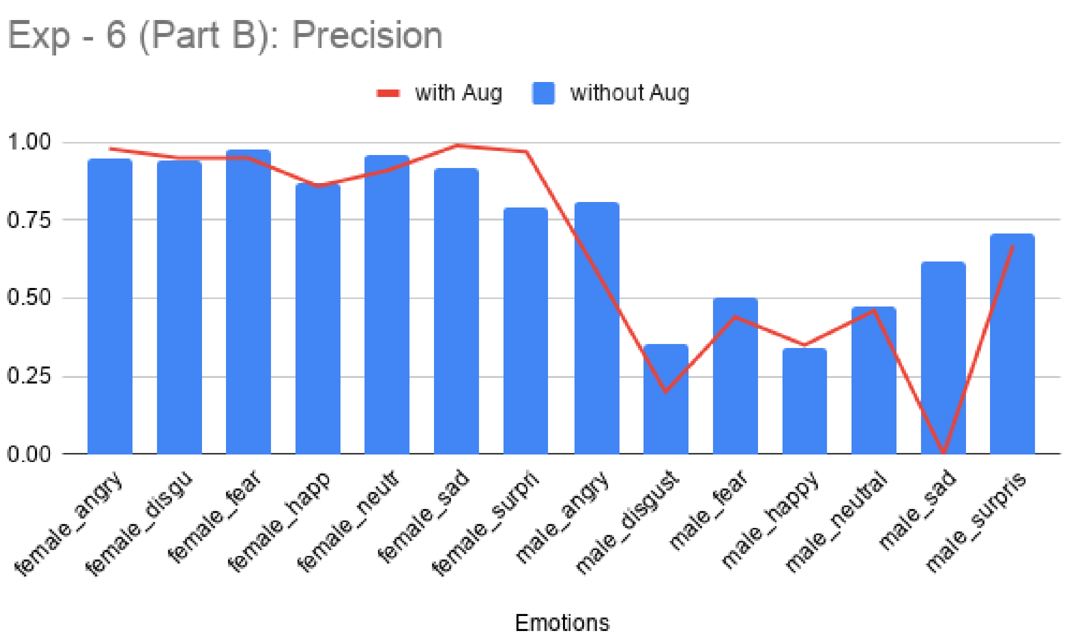
Exp - 6 (Part A): Precision
without Aug
with Aug
1.00
0.75
0.50
0.25
0.00
female_angry
female_disgu
female_fear
female_happ
female_neutr
female_sad
female_surpri
male_angry
male_disgust
male_fear
male_happy
male_neutral
male_sad
male_surpris
Emotions

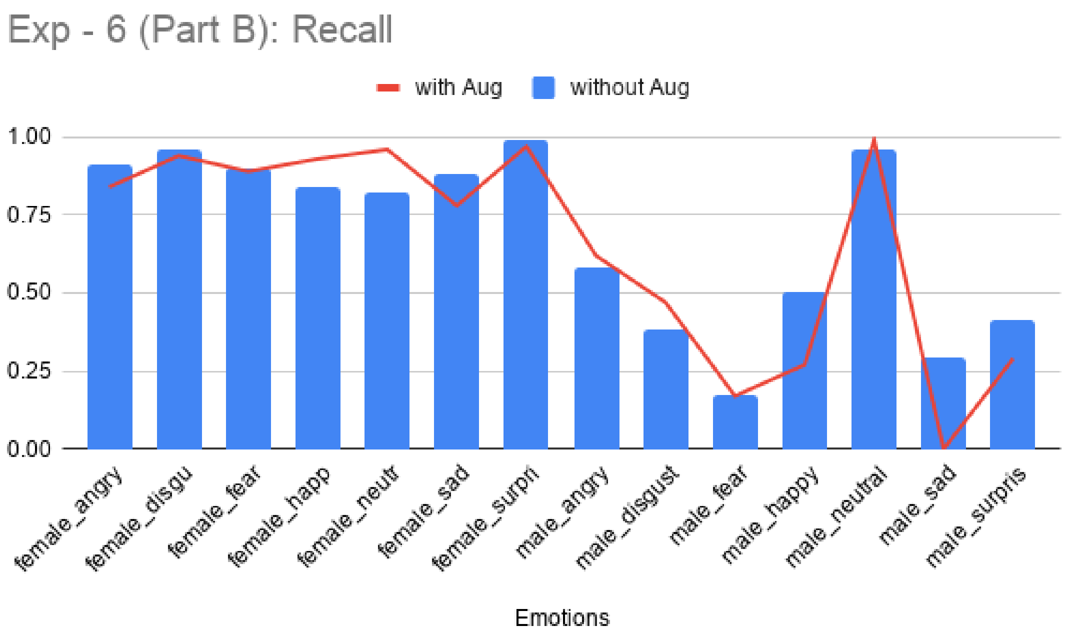


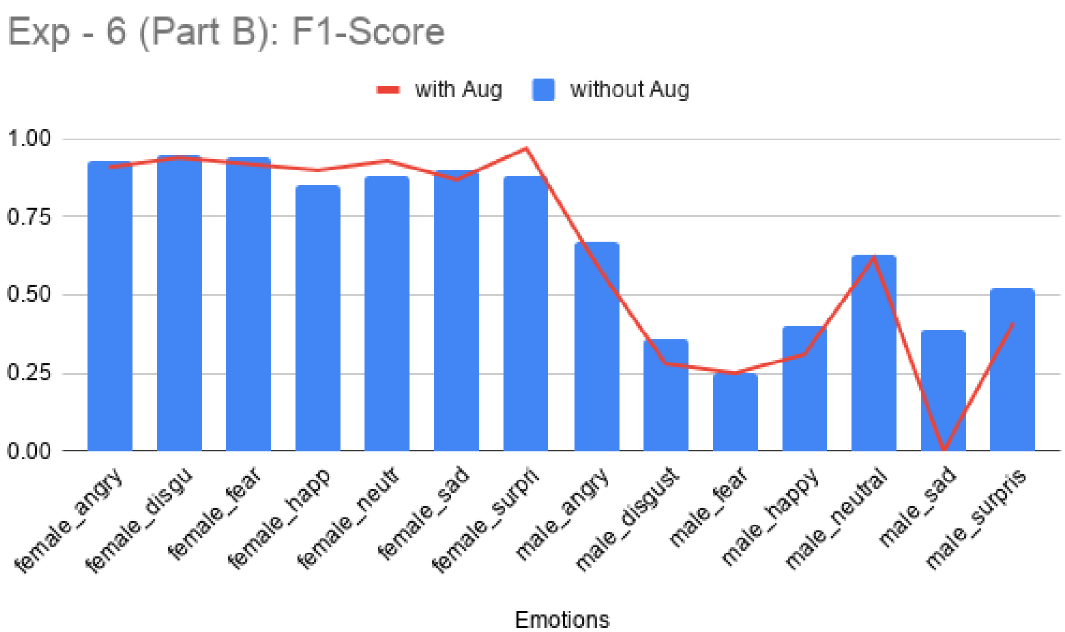
Exp - 6 (Part A): Recall
without Aug
with Aug
1.00
0.75
0.50
0.25
0.00
female_angry
female_disgu
female_fear
female_happ
female_neutr
female_sad
female_surpri
male_angry
male_disgust
male_fear
male_happy
male_neutral
male_sad
male_surpris
Emotions



Part: B 3. Mel Spectogram * Model Results: * The accuracy is **79.58%** * Gender classification accuracy is **98.89%** 4. Mel Spectogram with Augmentation * Model Results: * The accuracy is **78.31%** * Gender classification accuracy is **96.86%**







- In each of these experiments text and train split is 25:75 and the test and train samples are similar throughout the experiment.
- The precision, f1-score, recall and confusion matrix can be found in respective sections along with the validitation loss and accuracy plots.
- This concludes our sixth and final experiment.

Back to Table of Contents

Back to Experiments and Results

1.13 Discussions

Some points were directly evident from the results we found in previous section:

1. Almost all the models were able to distinguish between male and female voices properly with high accuracy *(inferred from all experiments)*.
2. 2D CNN architecture gave better results in case of accuracy on test data as compared to 1D CNN architectures *(inferred by comparing 6th experiment to earlier ones)*.
3. Randomized training data gave better results than not-randomized training data *(inferred from experiment 1st and 5th)*.
4. Augmentation methods when applied to data gave better results in comparison to them being not applied *(inferred from experiment 3rd and 4th)*.
5. Librosa feature MFCC gives better result as compared to Mel Spectogram features *(inferred from 6th experiment)*.
6. Normalizing the data was a wiser decision as it is proven to improve the accuracy and speed up the training process.
7. Even though most of the accuracies may not seem high, but even if we try to randomly guess, then the chances of it being correct is 1 out of 14 which is 7% so compared to that even 15% is huge improvement.
8. The 2D CNN takes in a 2D array of 30 MFCC bands by 216 audio length as input data. We can imagine it as a 30 x 216 pixel image. It has 4 convolution blocks of batch normalisation, max pooling and a dropout node. So your standard setup similar to VGG19, just not as deep. And we're using Adam for our optimiser.
9. We have shown the visualisation of the MFCC in few experiments, where it captures all the core information of the audio file into a single image. Well, if an audio information can be interpreted as an image, then can we can apply the same image recognition approaches like VGG19 or RESNET to them?
   - The answer is yes. And is suprisingly very fast and accurate. Its not as accurate as when applying RNN type models on the audio wave itself. But its very close to its accuracy potential, and heaps faster. There are some assumptions and limitations depending on use cases of course.
10. Data augmentation adds value in training the models and helps in increasing accuracy.
11. We wanted to add some more datasets in the experiments like the CREMA-D and EmoDB.
12. We can even add some more augmentation methods and also tune them to get optimal results.

Some mistakes that were made:

1. In experiment 5 we trained the model for 700 epochs, maybe that is the reason it has performed so poorly. This is because the accuracy it reached on the 700th epoch is 99.41% which means that the model was overiftted on the training data and was unable to classify the testing data that is the unheard audio files properly. 100 to 150 epochs are more than enough for the training process.
2. As we increased the number of experiments based on different ideas, we realized that the earlier functions are not generalized and so were not reused properly. We have defined functions properly in the last experiment though.
3. In the later sections, we haven't saved the models for future use. But that won't matter because we have set parameters in such a way that the models are easily reproducible.

Back to Table of Contents

1.14 Summary

- We have conducted 6 experiments on various combination of 3 datasets. What we learned is that if we take just one dataset we have higher chances of running into problem of overfitting and so whilst their hold-out accuracy is high, they don't work well on new unseen data. This might be because the classifier is trained on the same dataset and given the similar circumstances that the dataset was obtained or produced, (eg. audio quality, speaker repetition, duration and sentence uttered).
- We have plotted precision, recall and f1-score graphs (Experiments and Results) to compare between different models and different experimenting techniques (like the use of Augmentation and librosa features; MFCC and Mel Spectogram) implemented in the project. Most of the graphs are self-evident in the sense we can easily deduct how the model has performed in classifying different emotions.
- The gender seperation turns out to be a crucial implementation in order to accurately classify emotions. Upon closer inspection of the confusion matrix, it seems that female tends to express emotions in a more, obvious manner, for the lack of a better word. Whilst males tend to be very placid or subtle. This is probably why we see the error rate amongst males are really high. For example, male happy and angry gets mixed up quite often.
- Our experiments showed that data augmentation does help improve the accuracy albeit slightly. Note that we only introduced two augmentation methods. Perhaps, if we include more it may make it more accurate. But there comes to a point where we have to consider the trade off between speed and accuracy.

Back to Table of Contents

1.15 Future Scope

- There is a wide range of experiments that can be done with audio files. Multiple datasets are available for the same. Our study inculcates a small portion of them. We can try different combination of datasets, with and without augmentation and using different features of libraries like librosa.
- Most of the models did a pretty good job in differentiating genders and performed average when it comes to emotions. We can also implement various CNN models for this task. Also some benchmark models like RESNET or XCEPTION or VGG19 can be used for transfer learning.
- Last but not the least we can even use DCGANs for generating audios from the given datasets, that would be an interesting thing to do. This is because the MFCC and mel-spectogram we used are in form of images and DCGAN also uses images for learning.

Back to Table of Contents

1.16 References

1. H. Cao, R. Verma, and A. Nenkova, "Speaker-sensitive emotion recognition via ranking: Studies on acted andspontaneous speech," Comput. Speech Lang., vol. 28, no. 1, pp. 186–202, Jan. 2015.
2. T. L. Nwe, S. W. Foo, and L. C. De Silva, "Speech emotion recognition using hidden Markov models," SpeechCommun., vol. 41, no. 4, pp. 603–623, Nov. 2003.
3. L. Chen, X. Mao, Y. Xue, and L. L. Cheng, "Speech emotion recognition: Features and classification models,"Digit. Signal Process., vol. 22, no. 6, pp. 1154–1160, Dec. 2012.
4. J. Rong, G. Li, and Y.-P. P. Chen, "Acoustic feature selection for automatic emotion recognition from speech,"Inf. Process. Manag., vol. 45, no. 3, pp. 315–328, May 2009.
5. C.-H. Wu and W.-B. Liang, "Emotion Recognition of Affective Speech Based on Multiple Classifiers UsingAcoustic-Prosodic Information and Semantic Labels," IEEE Trans. Affect. Comput., vol. 2, no. 1, pp. 10–21,Jan. 2011.
6. S. S. Narayanan, "Toward detecting emotions in spoken dialogs," IEEE Trans. Speech Audio Process., vol. 13,no. 2, pp. 293–303, Mar. 2005.
7. B. Yang and M. Lugger, "Emotion recognition from speech signals using new harmony features," SignalProcessing, vol. 90, no. 5, pp. 1415–1423, May 2010.

Back to Table of Contents

1.17 Code

1.18 Importing Libraries

[0]:
```
# Import libraries
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.pyplot import specgram
from tqdm import tqdm, tqdm_pandas
import scipy
from scipy.stats import skew
import pandas as pd
import glob
import os
import sys
import warnings
import json
import seaborn as sns
import pickle
from sklearn.metrics import confusion_matrix
```

```
import IPython.display as ipd  # To play sound in the notebook
# ignore warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Keras
import keras
from keras import regularizers
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model, model_from_json
from keras.layers import Dense, Embedding, LSTM
from keras.layers import Input, Flatten, Dropout, Activation, BatchNormalization
from keras.layers import Conv1D, MaxPooling1D, AveragePooling1D
from keras.utils import np_utils, to_categorical
from keras.callbacks import ModelCheckpoint
from keras.callbacks import (EarlyStopping, LearningRateScheduler,
                             ModelCheckpoint, TensorBoard, ReduceLROnPlateau)
from keras import losses, models, optimizers
from keras.activations import relu, softmax
from keras.layers import (Convolution2D, GlobalAveragePooling2D,␣
 ↪BatchNormalization, Flatten, Dropout,
                          GlobalMaxPool2D, MaxPool2D, concatenate, Activation,␣
 ↪Input, Dense)

# sklearn
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
/home/subodh/anaconda3/lib/python3.7/site-
packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is
deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
Using TensorFlow backend.
```

1.19 Exploring Data

[0]:
```
RAV = "./big_size/"
SAVEE = "./SAVEE_used/"
TESS = "./TESS/TESS_data/"
```

[0]:
```
dir_list_rav = os.listdir(RAV)
dir_list_savee = os.listdir(SAVEE)
```

```
dir_list_tess = os.listdir(TESS)
```

1.20 A. RAVDESS

[0]:
```
emotion = []
gender = []
path = []
for i in dir_list_rav:
    fname = os.listdir(RAV + i)
    for f in fname:
        part = f.split('.')[0].split('-')
        #print(i, f, part)
        emotion.append(int(part[2]))
        temp = int(part[6])
        if temp%2 == 0:
            temp = "female"
        else:
            temp = "male"
        gender.append(temp)
        path.append(RAV + i + '/' + f)

RAV_df = pd.DataFrame(emotion)
RAV_df = RAV_df.replace({1:'neutral', 2:'neutral', 3:'happy', 4:'sad', 5:
↪'angry', 6:'fear', 7:'disgust', 8:'surprise'})
RAV_df = pd.concat([pd.DataFrame(gender),RAV_df],axis=1)
RAV_df.columns = ['gender','emotion']
RAV_df['labels'] =RAV_df.gender + '_' + RAV_df.emotion
RAV_df['source'] = 'RAVDESS'
RAV_df = pd.concat([RAV_df,pd.DataFrame(path, columns = ['path'])],axis=1)
RAV_df = RAV_df.drop(['gender', 'emotion'], axis=1)
RAV_df.labels.value_counts()
```

[0]:
```
male_neutral       144
female_neutral     144
female_sad          96
male_angry          96
male_surprise       96
female_happy        96
male_disgust        96
female_fear         96
female_surprise     96
male_happy          96
male_fear           96
female_disgust      96
male_sad            96
female_angry        96
Name: labels, dtype: int64
```

1.21 B. SAVEE

[0]:
```
# parse the filename to get the emotions
emotion=[]
path = []
for i in dir_list_savee:
    if i[-8:-6]=='_a':
        emotion.append('male_angry')
    elif i[-8:-6]=='_d':
        emotion.append('male_disgust')
    elif i[-8:-6]=='_f':
        emotion.append('male_fear')
    elif i[-8:-6]=='_h':
        emotion.append('male_happy')
    elif i[-8:-6]=='_n':
        emotion.append('male_neutral')
    elif i[-8:-6]=='sa':
        emotion.append('male_sad')
    elif i[-8:-6]=='su':
        emotion.append('male_surprise')
    else:
        emotion.append('male_error')
    path.append(SAVEE + i)

# Now check out the label count distribution
SAVEE_df = pd.DataFrame(emotion, columns = ['labels'])
SAVEE_df['source'] = 'SAVEE'
SAVEE_df = pd.concat([SAVEE_df, pd.DataFrame(path, columns = ['path'])], axis =␣
 ↪1)
SAVEE_df.labels.value_counts()
```

[0]:
```
male_neutral     120
male_disgust      60
male_happy        60
male_sad          60
male_angry        60
male_fear         60
male_surprise     60
Name: labels, dtype: int64
```

1.22 C. TESS

[0]:
```
path = []
emotion = []

for i in dir_list_tess:
    fname = os.listdir(TESS + i)
```

```
    for f in fname:
        if i == 'OAF_angry' or i == 'YAF_angry':
            emotion.append('female_angry')
        elif i == 'OAF_disgust' or i == 'YAF_disgust':
            emotion.append('female_disgust')
        elif i == 'OAF_Fear' or i == 'YAF_fear':
            emotion.append('female_fear')
        elif i == 'OAF_happy' or i == 'YAF_happy':
            emotion.append('female_happy')
        elif i == 'OAF_neutral' or i == 'YAF_neutral':
            emotion.append('female_neutral')
        elif i == 'OAF_Pleasant_surprise' or i == 'YAF_pleasant_surprised':
            emotion.append('female_surprise')
        elif i == 'OAF_Sad' or i == 'YAF_sad':
            emotion.append('female_sad')
        else:
            emotion.append('Unknown')
        path.append(TESS + i + "/" + f)

TESS_df = pd.DataFrame(emotion, columns = ['labels'])
TESS_df['source'] = 'TESS'
TESS_df = pd.concat([TESS_df,pd.DataFrame(path, columns = ['path'])],axis=1)
TESS_df.labels.value_counts()
```

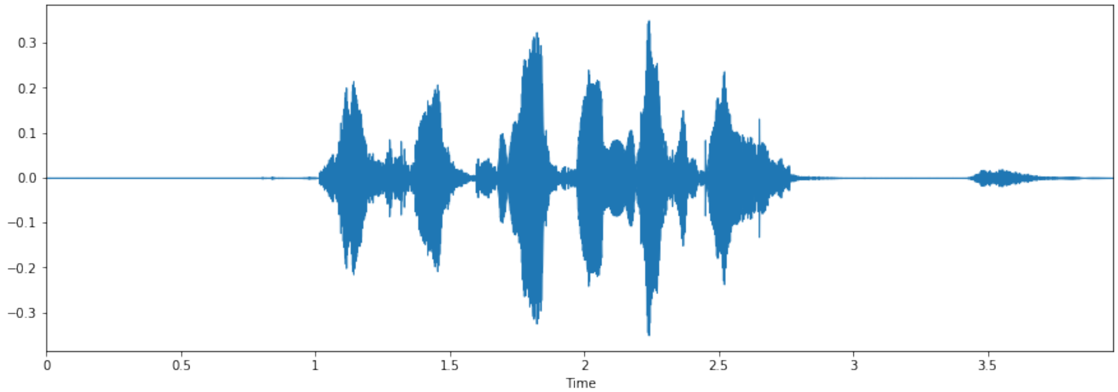
[0]:
```
female_sad         400
female_neutral     400
female_angry       400
female_happy       400
female_fear        400
female_surprise    400
female_disgust     400
Name: labels, dtype: int64
```

1.23 Let us take a look at our audio files

1.23.1 We will first compare angry female and angry male for the same sentence

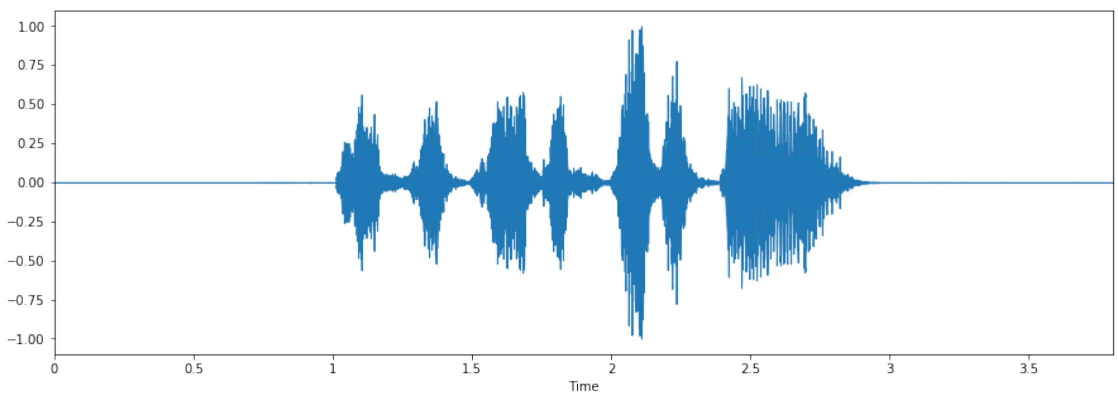
[0]:
```
# Source: RAVDESS
# Gender: Female
# Emotion: Anger
path_female = "./big_size/Actor_08/03-01-05-02-01-01-08.wav"
data, sampling_rate = librosa.load(path_female)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
# Play
ipd.Audio(data, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>



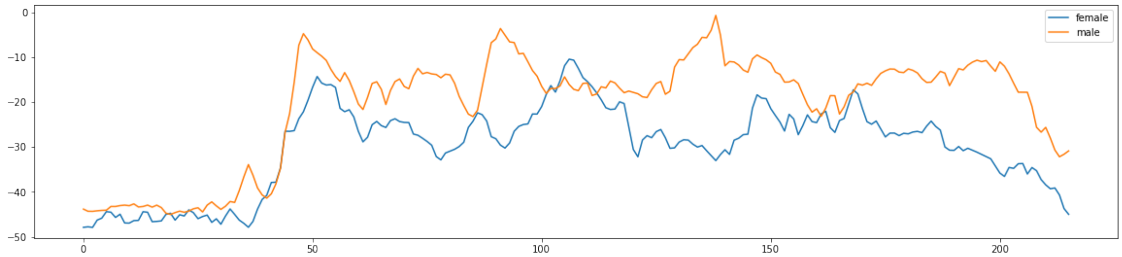

```
[0]: # Source: RAVDESS
     # Gender: Male
     # Emotion: Anger
     path_male = "./big_size/Actor_09/03-01-05-02-01-01-09.wav"
     data, sampling_rate = librosa.load(path_male)
     plt.figure(figsize=(15, 5))
     librosa.display.waveplot(data, sr=sampling_rate)
     # Play
     ipd.Audio(data, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>




```
[0]: X, sample_rate = librosa.load(path_female, res_type='kaiser_fast',duration=2.
     ↪5,sr=22050*2,offset=0.5)
     female = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
     female = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)
```

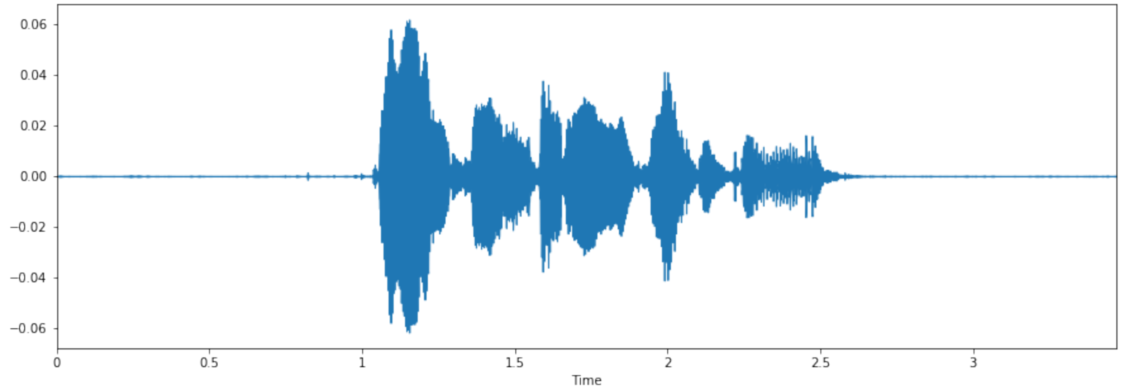
```
X, sample_rate = librosa.load(path_male, res_type='kaiser_fast',duration=2.
↪5,sr=22050*2,offset=0.5)
male = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
male = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
plt.plot(female, label='female')
plt.plot(male, label='male')
plt.legend()
plt.show()
```



1.23.2 Now we will compare happy male and happy female for the same sentence

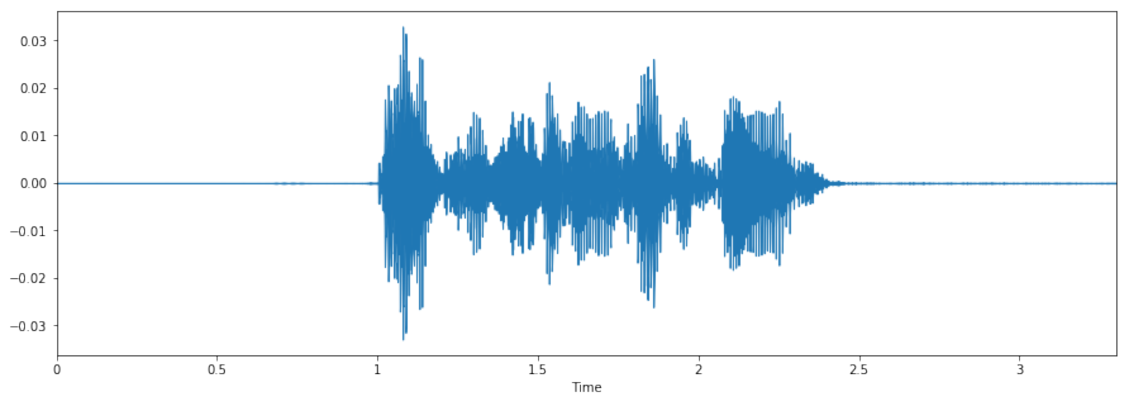
[0]:
```
# Source: RAVDESS
# Gender: Female
# Emotion: Happy
path_female = "./big_size/Actor_08/03-01-03-01-02-01-08.wav"
data, sampling_rate = librosa.load(path_female)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)
# Play
ipd.Audio(data, rate=sampling_rate)
```

[0]:
```
<IPython.lib.display.Audio object>
```



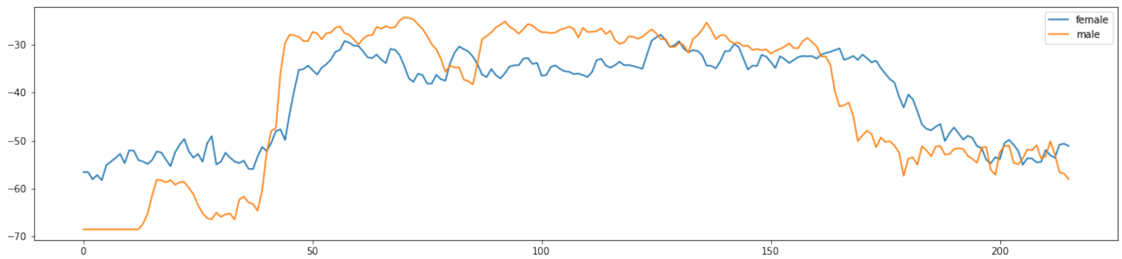

```
[0]: # Source: RAVDESS
     # Gender: Male
     # Emotion: Happy
     path_male = "./big_size/Actor_09/03-01-03-01-02-01-09.wav"
     data, sampling_rate = librosa.load(path_male)
     plt.figure(figsize=(15, 5))
     librosa.display.waveplot(data, sr=sampling_rate)
     # Play
     ipd.Audio(data, rate=sampling_rate)
```

```
[0]: <IPython.lib.display.Audio object>
```




```
[0]: X, sample_rate = librosa.load(path_female, res_type='kaiser_fast',duration=2.
     ↪5,sr=22050*2,offset=0.5)
     female = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
     female = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)

     X, sample_rate = librosa.load(path_male, res_type='kaiser_fast',duration=2.
     ↪5,sr=22050*2,offset=0.5)
```

```
male = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13)
male = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13), axis=0)

# audio wave
plt.figure(figsize=(20, 15))
plt.subplot(3,1,1)
plt.plot(female, label='female')
plt.plot(male, label='male')
plt.legend()
plt.show()
```



[0]:

```
```

1.23.3 We will save everything for future use

[0]:

```
df_rav = pd.concat([RAV_df], axis = 0)
df_savee = pd.concat([SAVEE_df], axis = 0)
df_tess = pd.concat([TESS_df], axis = 0)
df_combined = pd.concat([SAVEE_df, RAV_df, TESS_df], axis = 0)
print("RAVDESS    : \n",df_rav.labels.value_counts())
print("\nSAVEE    : \n",df_savee.labels.value_counts())
print("\nTESS     : \n",df_tess.labels.value_counts())
print("\nCOMBINED: \n",df_combined.labels.value_counts())

df_rav.to_csv("Data_rav.csv",index=False)
df_savee.to_csv("Data_savee.csv",index=False)
df_tess.to_csv("Data_tess.csv",index=False)
df_combined.to_csv("Data_combined.csv",index=False)
```

```
RAVDESS    :
 male_neutral       144
female_neutral     144
female_sad          96
male_angry          96
male_surprise       96
female_happy        96
male_disgust        96
```

```
female_fear         96
female_surprise     96
male_happy          96
male_fear           96
female_disgust      96
male_sad            96
female_angry        96
Name: labels, dtype: int64

SAVEE   :
 male_neutral     120
male_disgust      60
male_happy        60
male_sad          60
male_angry        60
male_fear         60
male_surprise     60
Name: labels, dtype: int64

TESS    :
 female_sad         400
female_neutral     400
female_angry       400
female_happy       400
female_fear        400
female_surprise    400
female_disgust     400
Name: labels, dtype: int64

COMBINED:
 female_neutral     544
female_sad         496
female_angry       496
female_happy       496
female_fear        496
female_surprise    496
female_disgust     496
male_neutral       264
male_surprise      156
male_disgust       156
male_angry         156
male_happy         156
male_fear          156
male_sad           156
Name: labels, dtype: int64
```

[0]: '\ndf_rav.to_csv("/content/drive/My Drive/Project/project_data/Data_rav.csv",index=False)\ndf_savee.to_csv("/content/drive/My Drive/Project/project_data/Data_savee.csv",index=False)\ndf_tess.to_csv("/content/drive/My Drive/Project/project_data/Data_tess.csv",index=False)\ndf_combined.to_csv("/content/drive/My Drive/Project/project_data/Data_combined.csv",index=False)\n'

[0]:
```
rav_data_path = 'Data_rav.csv'
savee_data_path = 'Data_savee.csv'
tess_data_path = 'Data_tess.csv'
combined_data_path = 'Data_combined.csv'
```

Data Exploration Ends Here

1.24 Required functions for data processing

[0]:
```
def read_dataset(filepath):
    return pd.read_csv(filepath)#data_set
```

1.24.1 Transforming the audio files in dataframes using librosa and mfcc

[0]:
```
def create_feature_dataframe(data_set):
    df = pd.DataFrame(columns=['feature'])

    counter = 0
    for index, path in enumerate(data_set.path):
        X, sample_rate = librosa.load(path, res_type = 'kaiser_fast'
                                      ,duration=2.5
                                      ,sr=44100
                                      ,offset=0.5)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y = X, sr = sample_rate, n_mfcc =␣
↪13), axis = 0)
        df.loc[counter] = [mfccs]
        counter += 1
    df = pd.concat([data_set, pd.DataFrame(df['feature'].values.
↪tolist())],axis=1)
    df = df.fillna(0)
    return df
rav_df = create_feature_dataframe(read_dataset(rav_data_path))
savee_df = create_feature_dataframe(read_dataset(savee_data_path))
tess_df = create_feature_dataframe(read_dataset(tess_data_path))
combined_df = create_feature_dataframe(read_dataset(combined_data_path))
rav_df.head()
```

[0]:
```
        labels source                       path          0          1  \
0    male_fear  SAVEE   ./SAVEE_used/KL_f04.wav -30.205614 -28.294016
```

```
1  male_angry  SAVEE   ./SAVEE_used/KL_a11.wav -40.303398 -37.919300
2   male_fear  SAVEE   ./SAVEE_used/JE_f11.wav -21.392101 -21.662266
3    male_sad  SAVEE  ./SAVEE_used/DC_sa11.wav -24.900761 -24.354008
4   male_fear  SAVEE   ./SAVEE_used/KL_f08.wav -35.302341 -35.131142

           2          3          4          5          6  …        206  \
0 -27.909389 -28.509830 -28.120195 -28.570707 -29.546034  …   0.000000
1 -36.645100 -28.010498 -24.288029 -22.791922 -23.459490  …   0.000000
2 -22.259338 -24.444984 -23.050682 -23.140684 -22.903954  … -22.763048
3 -23.842062 -23.961361 -21.653095 -21.758453 -23.224234  … -25.985544
4 -36.651817 -40.392941 -40.899864 -39.890297 -37.871014  … -38.050453

         207        208        209        210        211        212  \
0   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
2 -21.701115 -11.972590  -9.322908 -10.145143 -12.772147 -14.352438
3 -19.891569 -15.183666 -13.054301 -12.797897 -11.673220  -9.116754
4 -36.393749 -36.336716 -37.973526 -32.837669 -29.188053 -29.468193

         213        214        215
0   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000
2 -16.328117 -12.141912  -6.584519
3  -8.331745  -8.620239  -7.165553
4 -31.124041 -26.687956 -22.051907

[5 rows x 219 columns]
```

[0]:
```
tess_df.head()
```

[0]:
```
           labels source                                               path  \
0  female_neutral   TESS    ./TESS/TESS_data/OAF_neutral/OAF_red_neutral.wav
1  female_neutral   TESS  ./TESS/TESS_data/OAF_neutral/OAF_thought_neutr…
2  female_neutral   TESS  ./TESS/TESS_data/OAF_neutral/OAF_note_neutral.wav
3  female_neutral   TESS  ./TESS/TESS_data/OAF_neutral/OAF_date_neutral.wav
4  female_neutral   TESS  ./TESS/TESS_data/OAF_neutral/OAF_life_neutral.wav

           0          1          2          3          4          5  \
0 -27.531227 -27.284294 -28.146057 -27.797749 -27.840431 -28.782686
1 -15.580937 -18.178825 -27.634718 -26.854889 -26.591505 -25.873421
2 -15.811186 -19.120970 -26.876249 -27.312294 -28.224232 -27.710793
3 -19.699177 -21.673155 -26.733738 -27.016909 -27.891336 -27.544559
4 -19.158430 -21.439772 -27.321255 -26.996880 -27.611162 -26.968708

           6  …  205  206  207  208  209  210  211  212  213  214
0 -29.777445  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1 -26.057098  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
2 -26.631382  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3 -27.069057  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4 -26.897402  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 218 columns]
```

[0]:
```python
combined_df.head()
```

[0]:
```
       labels source                        path          0          1  \
0   male_fear  SAVEE   ./SAVEE_used/KL_f04.wav -30.205614 -28.294016
1  male_angry  SAVEE   ./SAVEE_used/KL_a11.wav -40.303398 -37.919300
2   male_fear  SAVEE   ./SAVEE_used/JE_f11.wav -21.392101 -21.662266
3    male_sad  SAVEE  ./SAVEE_used/DC_sa11.wav -24.900761 -24.354008
4   male_fear  SAVEE   ./SAVEE_used/KL_f08.wav -35.302341 -35.131142

           2          3          4          5          6  …        206  \
0 -27.909389 -28.509830 -28.120195 -28.570707 -29.546034  …   0.000000
1 -36.645100 -28.010498 -24.288029 -22.791922 -23.459490  …   0.000000
2 -22.259338 -24.444984 -23.050682 -23.140684 -22.903954  … -22.763048
3 -23.842062 -23.961361 -21.653095 -21.758453 -23.224234  … -25.985544
4 -36.651817 -40.392941 -40.899864 -39.890297 -37.871014  … -38.050453

         207        208        209        210        211        212  \
0   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
2 -21.701115 -11.972590  -9.322908 -10.145143 -12.772147 -14.352438
3 -19.891569 -15.183666 -13.054301 -12.797897 -11.673220  -9.116754
4 -36.393749 -36.336716 -37.973526 -32.837669 -29.188053 -29.468193

         213        214        215
0   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000
2 -16.328117 -12.141912  -6.584519
3  -8.331745  -8.620239  -7.165553
4 -31.124041 -26.687956 -22.051907

[5 rows x 219 columns]
```

1.25 Test Train Split

[0]:
```python
# Train, Test split and Normalization

def test_train_split(df, fraction = 0.25):
    X_train, X_test, y_train, y_test = train_test_split(df.drop(['path',␣
 ↪'labels','source'], axis = 1)
                                                        , df.labels
                                                        , test_size = fraction
```

```
                                                    , shuffle = True
                                                    , random_state = 42)
    mean = np.mean(X_train, axis = 0)
    std = np.std(X_train, axis = 0)
    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    return X_train, y_train, X_test, y_test
```

1.25.1 Following function process data and returns the test and train data and labels

[0]:
```
# Lets do few preparation steps to get it into the correct format for Keras
def process_data(df, fraction):
    X_train, y_train, X_test, y_test = test_train_split(df, fraction)
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)

    # expand dimensions
    X_train = np.expand_dims(X_train, axis=2)
    X_test = np.expand_dims(X_test, axis=2)

    # one hot encode the target
    lb = LabelEncoder()
    y_train = np_utils.to_categorical(lb.fit_transform(y_train))
    y_test = np_utils.to_categorical(lb.fit_transform(y_test))

    return X_train, y_train, X_test, y_test, lb

X_train, y_train, X_test, y_test, labels = process_data(rav_df, 0.2)
```

[0]:
```
labels.classes_
```

[0]:
```
array(['female_angry', 'female_disgust', 'female_fear', 'female_happy',
       'female_neutral', 'female_sad', 'female_surprise', 'male_angry',
       'male_disgust', 'male_fear', 'male_happy', 'male_neutral',
       'male_sad', 'male_surprise'], dtype=object)
```

[0]:
```
## Save the labels
filename = 'labels'
outfile = open(filename,'wb')
pickle.dump(labels, outfile)
outfile.close()
```

[0]:
```
X_train.shape
```

[0]: (1152, 216, 1)

[0]:
```
X_test.shape
```

[0]: (288, 216, 1)

1.25.2 We now define two of our models that we will work with throughout the experiments

[0]:
```
def build_model_1():
    model = Sequential()
    model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1))) ␣
↪# X_train.shape[1] = No. of Columns
    model.add(Activation('relu'))
    model.add(Conv1D(256, 8, padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    model.add(MaxPooling1D(pool_size=(8)))
    model.add(Conv1D(128, 8, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(128, 8, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(128, 8, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(128, 8, padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    model.add(MaxPooling1D(pool_size=(8)))
    model.add(Conv1D(64, 8, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(64, 8, padding='same'))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(14)) # Target class number
    model.add(Activation('softmax'))
    model.summary()
    opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
    model.compile(loss='categorical_crossentropy',␣
↪optimizer=opt,metrics=['accuracy'])
    tf.keras.utils.plot_model(model, to_file='model_1.png', show_shapes=True,␣
↪show_layer_names=True)
    return model
```

[0]:
```
from keras import regularizers
def build_model_2():
    model = Sequential()
    model.add(Conv1D(128, 5, padding='same',input_shape=(X_train.shape[1],1))) ␣
↪# X_train.shape[1] = No. of Columns
    model.add(Activation('relu'))
    model.add(Conv1D(128, 5, padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.1))
    model.add(MaxPooling1D(pool_size=(8)))
    model.add(Conv1D(128, 5, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(128, 5, padding='same'))
    model.add(Activation('relu'))
    model.add(Conv1D(128, 5, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Conv1D(128, 5, padding='same'))
    #model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(14))
    model.add(Activation('softmax'))
    # opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0,␣
↪nesterov=False)
    # opt = keras.optimizers.Adam(lr=0.0001)
    opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
    model.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics=['accuracy'])
    model.summary()
    tf.keras.utils.plot_model(model, to_file='model_2.png', show_shapes=True,␣
↪show_layer_names=True)
    return model
```

1.25.3 Training Model 1 on RAVDESS train data

[0]:
```
model_1 = build_model_1()
model_1_history=model_1.fit(X_train, y_train, batch_size=16, epochs=100,␣
↪validation_data=(X_test, y_test))
```

```
Model: "sequential_3"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_9 (Conv1D)            (None, 216, 256)          2304

_________________________________________________________________
```

```
activation_10 (Activation)   (None, 216, 256)          0
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 216, 256)          524544
_________________________________________________________________
batch_normalization_3 (Batch (None, 216, 256)          1024
_________________________________________________________________
activation_11 (Activation)   (None, 216, 256)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 216, 256)          0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 27, 256)           0
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 27, 128)           262272
_________________________________________________________________
activation_12 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
activation_13 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
activation_14 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
batch_normalization_4 (Batch (None, 27, 128)           512
_________________________________________________________________
activation_15 (Activation)   (None, 27, 128)           0
_________________________________________________________________
dropout_4 (Dropout)          (None, 27, 128)           0
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 3, 128)            0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 3, 64)             65600
_________________________________________________________________
activation_16 (Activation)   (None, 3, 64)             0
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 3, 64)             32832
_________________________________________________________________
activation_17 (Activation)   (None, 3, 64)             0
_________________________________________________________________
flatten_2 (Flatten)          (None, 192)               0
_________________________________________________________________
dense_2 (Dense)              (None, 14)                2702
_________________________________________________________________
activation_18 (Activation)   (None, 14)                0
=================================================================
```

```
Total params: 1,285,390
Trainable params: 1,284,622
Non-trainable params: 768
_________________________________________________________________
Train on 1152 samples, validate on 288 samples
Epoch 1/100
1152/1152 [==============================] - 10s 8ms/step - loss: 2.5799 -
accuracy: 0.1293 - val_loss: 2.6383 - val_accuracy: 0.0556
Epoch 2/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.4254 -
accuracy: 0.2231 - val_loss: 2.6371 - val_accuracy: 0.0556
Epoch 3/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.3169 -
accuracy: 0.2535 - val_loss: 2.6322 - val_accuracy: 0.0556
Epoch 4/100
1152/1152 [==============================] - 9s 8ms/step - loss: 2.2540 -
accuracy: 0.2700 - val_loss: 2.6220 - val_accuracy: 0.0556
Epoch 5/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.1867 -
accuracy: 0.2847 - val_loss: 2.5976 - val_accuracy: 0.0625
Epoch 6/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.1208 -
accuracy: 0.3116 - val_loss: 2.5581 - val_accuracy: 0.1111
Epoch 7/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.0776 -
accuracy: 0.3255 - val_loss: 2.4885 - val_accuracy: 0.2049
Epoch 8/100
1152/1152 [==============================] - 8s 7ms/step - loss: 2.0224 -
accuracy: 0.3411 - val_loss: 2.4132 - val_accuracy: 0.2569
Epoch 9/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.9924 -
accuracy: 0.3594 - val_loss: 2.3231 - val_accuracy: 0.2847
Epoch 10/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.9410 -
accuracy: 0.3793 - val_loss: 2.2575 - val_accuracy: 0.2882
Epoch 11/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.8990 -
accuracy: 0.3863 - val_loss: 2.2016 - val_accuracy: 0.3125
Epoch 12/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.8649 -
accuracy: 0.3958 - val_loss: 2.1737 - val_accuracy: 0.3472
Epoch 13/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.8277 -
accuracy: 0.4097 - val_loss: 2.1342 - val_accuracy: 0.3472
Epoch 14/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.8011 -
accuracy: 0.4245 - val_loss: 2.1127 - val_accuracy: 0.3681
Epoch 15/100
```

```
1152/1152 [==============================] - 8s 7ms/step - loss: 1.7729 -
accuracy: 0.4280 - val_loss: 2.0752 - val_accuracy: 0.3819
Epoch 16/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.7382 -
accuracy: 0.4410 - val_loss: 2.0877 - val_accuracy: 0.3681
Epoch 17/100
1152/1152 [==============================] - 10s 9ms/step - loss: 1.6978 -
accuracy: 0.4661 - val_loss: 2.0551 - val_accuracy: 0.3646
Epoch 18/100
1152/1152 [==============================] - 10s 9ms/step - loss: 1.6775 -
accuracy: 0.4583 - val_loss: 2.0235 - val_accuracy: 0.3889
Epoch 19/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.6537 -
accuracy: 0.4870 - val_loss: 2.0029 - val_accuracy: 0.3819
Epoch 20/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.6176 -
accuracy: 0.4974 - val_loss: 1.9952 - val_accuracy: 0.3924
Epoch 21/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.6091 -
accuracy: 0.4852 - val_loss: 1.9712 - val_accuracy: 0.3993
Epoch 22/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.5720 -
accuracy: 0.5139 - val_loss: 1.9616 - val_accuracy: 0.3889
Epoch 23/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.5535 -
accuracy: 0.5035 - val_loss: 1.9453 - val_accuracy: 0.4028
Epoch 24/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.5302 -
accuracy: 0.5226 - val_loss: 1.9340 - val_accuracy: 0.4028
Epoch 25/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.5062 -
accuracy: 0.5295 - val_loss: 1.9191 - val_accuracy: 0.4167
Epoch 26/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.4860 -
accuracy: 0.5339 - val_loss: 1.9046 - val_accuracy: 0.4097
Epoch 27/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.4583 -
accuracy: 0.5312 - val_loss: 1.9047 - val_accuracy: 0.4167
Epoch 28/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.4436 -
accuracy: 0.5451 - val_loss: 1.8897 - val_accuracy: 0.4097
Epoch 29/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.4312 -
accuracy: 0.5477 - val_loss: 1.8765 - val_accuracy: 0.4444
Epoch 30/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.4050 -
accuracy: 0.5642 - val_loss: 1.8728 - val_accuracy: 0.4271
Epoch 31/100
```

```
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3792 -
accuracy: 0.5799 - val_loss: 1.8718 - val_accuracy: 0.4514
Epoch 32/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3537 -
accuracy: 0.5842 - val_loss: 1.8757 - val_accuracy: 0.4271
Epoch 33/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3408 -
accuracy: 0.5938 - val_loss: 1.8389 - val_accuracy: 0.4653
Epoch 34/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3321 -
accuracy: 0.5833 - val_loss: 1.8356 - val_accuracy: 0.4514
Epoch 35/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3066 -
accuracy: 0.6076 - val_loss: 1.8399 - val_accuracy: 0.4514
Epoch 36/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.3000 -
accuracy: 0.6094 - val_loss: 1.8218 - val_accuracy: 0.4549
Epoch 37/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.2676 -
accuracy: 0.6337 - val_loss: 1.8042 - val_accuracy: 0.4722
Epoch 38/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.2463 -
accuracy: 0.6337 - val_loss: 1.8068 - val_accuracy: 0.4583
Epoch 39/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.2247 -
accuracy: 0.6476 - val_loss: 1.7927 - val_accuracy: 0.4583
Epoch 40/100
1152/1152 [==============================] - 10s 9ms/step - loss: 1.2166 -
accuracy: 0.6528 - val_loss: 1.7856 - val_accuracy: 0.4688
Epoch 41/100
1152/1152 [==============================] - 13s 11ms/step - loss: 1.2025 -
accuracy: 0.6493 - val_loss: 1.7984 - val_accuracy: 0.4757
Epoch 42/100
1152/1152 [==============================] - 11s 9ms/step - loss: 1.1838 -
accuracy: 0.6623 - val_loss: 1.7714 - val_accuracy: 0.4375
Epoch 43/100
1152/1152 [==============================] - 11s 9ms/step - loss: 1.1582 -
accuracy: 0.6736 - val_loss: 1.7713 - val_accuracy: 0.4549
Epoch 44/100
1152/1152 [==============================] - 9s 8ms/step - loss: 1.1456 -
accuracy: 0.6866 - val_loss: 1.7736 - val_accuracy: 0.4479
Epoch 45/100
1152/1152 [==============================] - 10s 9ms/step - loss: 1.1343 -
accuracy: 0.6753 - val_loss: 1.7563 - val_accuracy: 0.4653
Epoch 46/100
1152/1152 [==============================] - 11s 10ms/step - loss: 1.0992 -
accuracy: 0.6988 - val_loss: 1.7647 - val_accuracy: 0.4479
Epoch 47/100
```

```
1152/1152 [==============================] - 11s 10ms/step - loss: 1.1040 -
accuracy: 0.6936 - val_loss: 1.7429 - val_accuracy: 0.4549
Epoch 48/100
1152/1152 [==============================] - 9s 8ms/step - loss: 1.0735 -
accuracy: 0.7179 - val_loss: 1.7526 - val_accuracy: 0.4653
Epoch 49/100
1152/1152 [==============================] - 11s 10ms/step - loss: 1.0528 -
accuracy: 0.7266 - val_loss: 1.7357 - val_accuracy: 0.4583
Epoch 50/100
1152/1152 [==============================] - 11s 10ms/step - loss: 1.0357 -
accuracy: 0.7231 - val_loss: 1.7327 - val_accuracy: 0.4514
Epoch 51/100
1152/1152 [==============================] - 10s 9ms/step - loss: 1.0400 -
accuracy: 0.7352 - val_loss: 1.7330 - val_accuracy: 0.4583
Epoch 52/100
1152/1152 [==============================] - 8s 7ms/step - loss: 1.0156 -
accuracy: 0.7222 - val_loss: 1.7328 - val_accuracy: 0.4757
Epoch 53/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.9837 -
accuracy: 0.7422 - val_loss: 1.7339 - val_accuracy: 0.4653
Epoch 54/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.9860 -
accuracy: 0.7500 - val_loss: 1.7309 - val_accuracy: 0.4514
Epoch 55/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.9662 -
accuracy: 0.7439 - val_loss: 1.7125 - val_accuracy: 0.4479
Epoch 56/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.9556 -
accuracy: 0.7526 - val_loss: 1.6938 - val_accuracy: 0.4792
Epoch 57/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.9411 -
accuracy: 0.7656 - val_loss: 1.6873 - val_accuracy: 0.4722
Epoch 58/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.9094 -
accuracy: 0.7812 - val_loss: 1.6937 - val_accuracy: 0.4722
Epoch 59/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.9184 -
accuracy: 0.7847 - val_loss: 1.6849 - val_accuracy: 0.4896
Epoch 60/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.8852 -
accuracy: 0.7812 - val_loss: 1.6734 - val_accuracy: 0.4931
Epoch 61/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.8815 -
accuracy: 0.7882 - val_loss: 1.6787 - val_accuracy: 0.4722
Epoch 62/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.8515 -
accuracy: 0.8021 - val_loss: 1.6886 - val_accuracy: 0.4688
Epoch 63/100
```

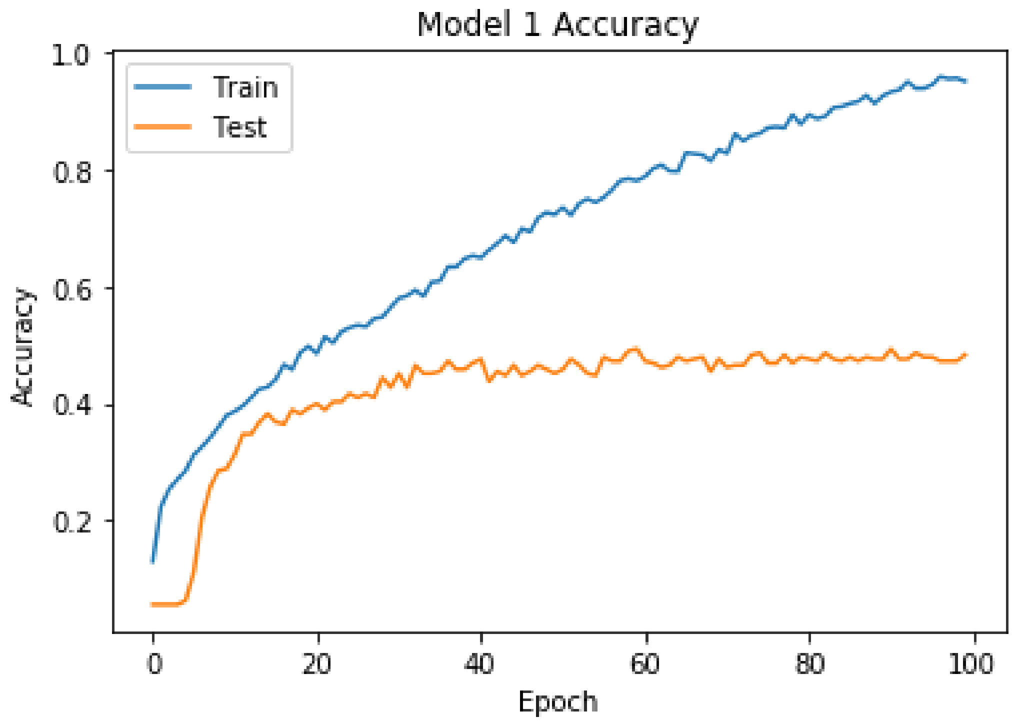
```
1152/1152 [==============================] - 9s 8ms/step - loss: 0.8354 -
accuracy: 0.8082 - val_loss: 1.6694 - val_accuracy: 0.4618
Epoch 64/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.8437 -
accuracy: 0.7977 - val_loss: 1.6702 - val_accuracy: 0.4653
Epoch 65/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.8223 -
accuracy: 0.7969 - val_loss: 1.6689 - val_accuracy: 0.4792
Epoch 66/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.8007 -
accuracy: 0.8290 - val_loss: 1.6790 - val_accuracy: 0.4722
Epoch 67/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.7819 -
accuracy: 0.8273 - val_loss: 1.6478 - val_accuracy: 0.4757
Epoch 68/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.7689 -
accuracy: 0.8255 - val_loss: 1.6527 - val_accuracy: 0.4792
Epoch 69/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.7669 -
accuracy: 0.8151 - val_loss: 1.6541 - val_accuracy: 0.4549
Epoch 70/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.7434 -
accuracy: 0.8342 - val_loss: 1.6469 - val_accuracy: 0.4757
Epoch 71/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.7365 -
accuracy: 0.8281 - val_loss: 1.6698 - val_accuracy: 0.4618
Epoch 72/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.7270 -
accuracy: 0.8611 - val_loss: 1.6345 - val_accuracy: 0.4653
Epoch 73/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.6967 -
accuracy: 0.8490 - val_loss: 1.6240 - val_accuracy: 0.4653
Epoch 74/100
1152/1152 [==============================] - 8s 7ms/step - loss: 0.6821 -
accuracy: 0.8585 - val_loss: 1.6160 - val_accuracy: 0.4826
Epoch 75/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.6871 -
accuracy: 0.8620 - val_loss: 1.6123 - val_accuracy: 0.4861
Epoch 76/100
1152/1152 [==============================] - 9s 7ms/step - loss: 0.6577 -
accuracy: 0.8715 - val_loss: 1.6299 - val_accuracy: 0.4688
Epoch 77/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.6455 -
accuracy: 0.8733 - val_loss: 1.6235 - val_accuracy: 0.4688
Epoch 78/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.6379 -
accuracy: 0.8715 - val_loss: 1.6095 - val_accuracy: 0.4826
Epoch 79/100
```

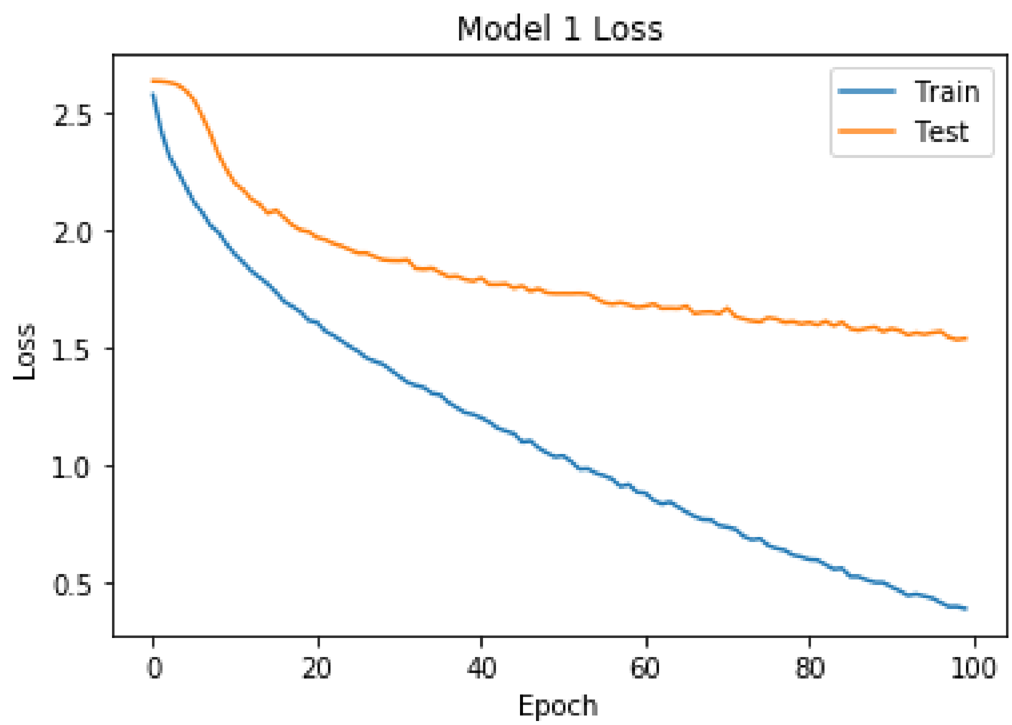
```
1152/1152 [==============================] - 10s 8ms/step - loss: 0.6172 -
accuracy: 0.8941 - val_loss: 1.6129 - val_accuracy: 0.4688
Epoch 80/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.6097 -
accuracy: 0.8776 - val_loss: 1.6032 - val_accuracy: 0.4792
Epoch 81/100
1152/1152 [==============================] - 13s 11ms/step - loss: 0.5983 -
accuracy: 0.8941 - val_loss: 1.6085 - val_accuracy: 0.4757
Epoch 82/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5959 -
accuracy: 0.8872 - val_loss: 1.5989 - val_accuracy: 0.4722
Epoch 83/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5755 -
accuracy: 0.8915 - val_loss: 1.6143 - val_accuracy: 0.4861
Epoch 84/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5564 -
accuracy: 0.9062 - val_loss: 1.5950 - val_accuracy: 0.4757
Epoch 85/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5604 -
accuracy: 0.9080 - val_loss: 1.6102 - val_accuracy: 0.4722
Epoch 86/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5252 -
accuracy: 0.9132 - val_loss: 1.5828 - val_accuracy: 0.4792
Epoch 87/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.5247 -
accuracy: 0.9167 - val_loss: 1.5756 - val_accuracy: 0.4722
Epoch 88/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5108 -
accuracy: 0.9271 - val_loss: 1.5839 - val_accuracy: 0.4792
Epoch 89/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.5005 -
accuracy: 0.9132 - val_loss: 1.5895 - val_accuracy: 0.4757
Epoch 90/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.4999 -
accuracy: 0.9262 - val_loss: 1.5697 - val_accuracy: 0.4757
Epoch 91/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.4815 -
accuracy: 0.9332 - val_loss: 1.5828 - val_accuracy: 0.4931
Epoch 92/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.4650 -
accuracy: 0.9366 - val_loss: 1.5746 - val_accuracy: 0.4757
Epoch 93/100
1152/1152 [==============================] - 13s 11ms/step - loss: 0.4440 -
accuracy: 0.9505 - val_loss: 1.5557 - val_accuracy: 0.4757
Epoch 94/100
1152/1152 [==============================] - 9s 8ms/step - loss: 0.4504 -
accuracy: 0.9392 - val_loss: 1.5649 - val_accuracy: 0.4861
Epoch 95/100
```

```
1152/1152 [==============================] - 13s 11ms/step - loss: 0.4418 -
accuracy: 0.9392 - val_loss: 1.5584 - val_accuracy: 0.4792
Epoch 96/100
1152/1152 [==============================] - 11s 10ms/step - loss: 0.4330 -
accuracy: 0.9453 - val_loss: 1.5655 - val_accuracy: 0.4792
Epoch 97/100
1152/1152 [==============================] - 10s 8ms/step - loss: 0.4141 -
accuracy: 0.9592 - val_loss: 1.5708 - val_accuracy: 0.4722
Epoch 98/100
1152/1152 [==============================] - 10s 9ms/step - loss: 0.3970 -
accuracy: 0.9557 - val_loss: 1.5458 - val_accuracy: 0.4722
Epoch 99/100
1152/1152 [==============================] - 12s 11ms/step - loss: 0.3973 -
accuracy: 0.9566 - val_loss: 1.5357 - val_accuracy: 0.4722
Epoch 100/100
1152/1152 [==============================] - 10s 9ms/step - loss: 0.3884 -
accuracy: 0.9514 - val_loss: 1.5407 - val_accuracy: 0.4826
```

[0]:

```
plt.plot(model_1_history.history['accuracy'])
plt.plot(model_1_history.history['val_accuracy'])
plt.title('Model 1 Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
plt.plot(model_1_history.history['loss'])
plt.plot(model_1_history.history['val_loss'])
plt.title('Model 1 Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```


Model 1 Accuracy
Train
Test
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100


Model 1 Loss
Train
Test
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100

[0]:
```
# Save model and weights
model_name = 'Model_1.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model_1.save(model_path)
print('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model_1.to_json()
with open("model_1_json.json", "w") as json_file:
    json_file.write(model_json)
```

```
Save model and weights at /home/subodh/Second Semester/ML/Random
Project/Audio/saved_models/Model_1.h5
```

1.25.4 Training Model 2 on RAVDESS train data

[0]:
```
model_2 = build_model_2()
model_2_history=model_2.fit(X_train, y_train, batch_size=16, epochs=100,␣
↪validation_data=(X_test, y_test))
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_23 (Conv1D)           (None, 216, 128)          768
_________________________________________________________________
activation_33 (Activation)   (None, 216, 128)          0
_________________________________________________________________
conv1d_24 (Conv1D)           (None, 216, 128)          82048
_________________________________________________________________
activation_34 (Activation)   (None, 216, 128)          0
_________________________________________________________________
dropout_12 (Dropout)         (None, 216, 128)          0
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 27, 128)           0
_________________________________________________________________
conv1d_25 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_35 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_26 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_36 (Activation)   (None, 27, 128)           0
_________________________________________________________________
```

```
conv1d_27 (Conv1D)           (None, 27, 128)          82048
_________________________________________________________________
activation_37 (Activation)   (None, 27, 128)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 27, 128)          0
_________________________________________________________________
conv1d_28 (Conv1D)           (None, 27, 128)          82048
_________________________________________________________________
activation_38 (Activation)   (None, 27, 128)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 3456)             0
_________________________________________________________________
dense_11 (Dense)             (None, 14)               48398
_________________________________________________________________
activation_39 (Activation)   (None, 14)               0
=================================================================
Total params: 459,406
Trainable params: 459,406
Non-trainable params: 0
_________________________________________________________________
Train on 1152 samples, validate on 288 samples
Epoch 1/100
1152/1152 [==============================] - 4s 3ms/step - loss: 2.6322 -
accuracy: 0.1111 - val_loss: 2.6306 - val_accuracy: 0.0694
Epoch 2/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.6168 -
accuracy: 0.1207 - val_loss: 2.6168 - val_accuracy: 0.0764
Epoch 3/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.5954 -
accuracy: 0.1345 - val_loss: 2.5947 - val_accuracy: 0.1562
Epoch 4/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.5597 -
accuracy: 0.1780 - val_loss: 2.5604 - val_accuracy: 0.1840
Epoch 5/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.5117 -
accuracy: 0.1884 - val_loss: 2.5113 - val_accuracy: 0.1875
Epoch 6/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.4468 -
accuracy: 0.2118 - val_loss: 2.4503 - val_accuracy: 0.1910
Epoch 7/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.3714 -
accuracy: 0.2161 - val_loss: 2.3868 - val_accuracy: 0.1910
Epoch 8/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.3060 -
accuracy: 0.2257 - val_loss: 2.3278 - val_accuracy: 0.1944
Epoch 9/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.2374 -
accuracy: 0.2361 - val_loss: 2.2726 - val_accuracy: 0.2083
```

```
Epoch 10/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.1765 -
accuracy: 0.2526 - val_loss: 2.2199 - val_accuracy: 0.2222
Epoch 11/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.1241 -
accuracy: 0.2613 - val_loss: 2.1732 - val_accuracy: 0.2292
Epoch 12/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.0901 -
accuracy: 0.2899 - val_loss: 2.1349 - val_accuracy: 0.2361
Epoch 13/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.0545 -
accuracy: 0.2908 - val_loss: 2.1032 - val_accuracy: 0.2674
Epoch 14/100
1152/1152 [==============================] - 2s 2ms/step - loss: 2.0288 -
accuracy: 0.2995 - val_loss: 2.0791 - val_accuracy: 0.2743
Epoch 15/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9988 -
accuracy: 0.3108 - val_loss: 2.0535 - val_accuracy: 0.2882
Epoch 16/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9792 -
accuracy: 0.3047 - val_loss: 2.0369 - val_accuracy: 0.2951
Epoch 17/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9753 -
accuracy: 0.3142 - val_loss: 2.0259 - val_accuracy: 0.3021
Epoch 18/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9475 -
accuracy: 0.3212 - val_loss: 2.0137 - val_accuracy: 0.2986
Epoch 19/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9500 -
accuracy: 0.3151 - val_loss: 1.9975 - val_accuracy: 0.3160
Epoch 20/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9274 -
accuracy: 0.3151 - val_loss: 1.9928 - val_accuracy: 0.3090
Epoch 21/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9395 -
accuracy: 0.3168 - val_loss: 1.9882 - val_accuracy: 0.3229
Epoch 22/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9219 -
accuracy: 0.3238 - val_loss: 1.9728 - val_accuracy: 0.3194
Epoch 23/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9064 -
accuracy: 0.3281 - val_loss: 1.9669 - val_accuracy: 0.3264
Epoch 24/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.9069 -
accuracy: 0.3194 - val_loss: 1.9536 - val_accuracy: 0.3160
Epoch 25/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8846 -
accuracy: 0.3394 - val_loss: 1.9453 - val_accuracy: 0.3333
```

```
Epoch 26/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8859 -
accuracy: 0.3342 - val_loss: 1.9423 - val_accuracy: 0.3368
Epoch 27/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8704 -
accuracy: 0.3316 - val_loss: 1.9359 - val_accuracy: 0.3333
Epoch 28/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8729 -
accuracy: 0.3273 - val_loss: 1.9324 - val_accuracy: 0.3264
Epoch 29/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8589 -
accuracy: 0.3464 - val_loss: 1.9248 - val_accuracy: 0.3299
Epoch 30/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8521 -
accuracy: 0.3446 - val_loss: 1.9197 - val_accuracy: 0.3403
Epoch 31/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8570 -
accuracy: 0.3264 - val_loss: 1.9188 - val_accuracy: 0.3472
Epoch 32/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8470 -
accuracy: 0.3481 - val_loss: 1.9103 - val_accuracy: 0.3299
Epoch 33/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8334 -
accuracy: 0.3359 - val_loss: 1.9161 - val_accuracy: 0.3368
Epoch 34/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8398 -
accuracy: 0.3542 - val_loss: 1.9067 - val_accuracy: 0.3542
Epoch 35/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8215 -
accuracy: 0.3446 - val_loss: 1.9030 - val_accuracy: 0.3333
Epoch 36/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8213 -
accuracy: 0.3455 - val_loss: 1.9019 - val_accuracy: 0.3403
Epoch 37/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8136 -
accuracy: 0.3472 - val_loss: 1.8941 - val_accuracy: 0.3368
Epoch 38/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8134 -
accuracy: 0.3290 - val_loss: 1.8928 - val_accuracy: 0.3403
Epoch 39/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8133 -
accuracy: 0.3481 - val_loss: 1.8870 - val_accuracy: 0.3403
Epoch 40/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.8010 -
accuracy: 0.3576 - val_loss: 1.8862 - val_accuracy: 0.3507
Epoch 41/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7988 -
accuracy: 0.3568 - val_loss: 1.8831 - val_accuracy: 0.3472
```

```
Epoch 42/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7847 -
accuracy: 0.3646 - val_loss: 1.8780 - val_accuracy: 0.3472
Epoch 43/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7743 -
accuracy: 0.3559 - val_loss: 1.8821 - val_accuracy: 0.3576
Epoch 44/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7853 -
accuracy: 0.3446 - val_loss: 1.8805 - val_accuracy: 0.3542
Epoch 45/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7830 -
accuracy: 0.3689 - val_loss: 1.8741 - val_accuracy: 0.3472
Epoch 46/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7704 -
accuracy: 0.3724 - val_loss: 1.8676 - val_accuracy: 0.3542
Epoch 47/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7747 -
accuracy: 0.3620 - val_loss: 1.8593 - val_accuracy: 0.3542
Epoch 48/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7643 -
accuracy: 0.3715 - val_loss: 1.8614 - val_accuracy: 0.3507
Epoch 49/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7736 -
accuracy: 0.3637 - val_loss: 1.8561 - val_accuracy: 0.3576
Epoch 50/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7618 -
accuracy: 0.3681 - val_loss: 1.8531 - val_accuracy: 0.3507
Epoch 51/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7461 -
accuracy: 0.3655 - val_loss: 1.8630 - val_accuracy: 0.3611
Epoch 52/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7409 -
accuracy: 0.3672 - val_loss: 1.8567 - val_accuracy: 0.3507
Epoch 53/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7571 -
accuracy: 0.3516 - val_loss: 1.8560 - val_accuracy: 0.3542
Epoch 54/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7356 -
accuracy: 0.3724 - val_loss: 1.8454 - val_accuracy: 0.3438
Epoch 55/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7171 -
accuracy: 0.3906 - val_loss: 1.8466 - val_accuracy: 0.3611
Epoch 56/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7336 -
accuracy: 0.3672 - val_loss: 1.8419 - val_accuracy: 0.3507
Epoch 57/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7128 -
accuracy: 0.3759 - val_loss: 1.8312 - val_accuracy: 0.3611
```

```
Epoch 58/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7222 -
accuracy: 0.3741 - val_loss: 1.8313 - val_accuracy: 0.3611
Epoch 59/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7070 -
accuracy: 0.3663 - val_loss: 1.8330 - val_accuracy: 0.3576
Epoch 60/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7126 -
accuracy: 0.3828 - val_loss: 1.8261 - val_accuracy: 0.3576
Epoch 61/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7164 -
accuracy: 0.3759 - val_loss: 1.8278 - val_accuracy: 0.3576
Epoch 62/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.7033 -
accuracy: 0.3880 - val_loss: 1.8306 - val_accuracy: 0.3333
Epoch 63/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6898 -
accuracy: 0.3932 - val_loss: 1.8240 - val_accuracy: 0.3438
Epoch 64/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6932 -
accuracy: 0.3845 - val_loss: 1.8295 - val_accuracy: 0.3438
Epoch 65/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6922 -
accuracy: 0.3924 - val_loss: 1.8280 - val_accuracy: 0.3472
Epoch 66/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6827 -
accuracy: 0.3819 - val_loss: 1.8276 - val_accuracy: 0.3542
Epoch 67/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6846 -
accuracy: 0.3793 - val_loss: 1.8214 - val_accuracy: 0.3576
Epoch 68/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6647 -
accuracy: 0.3950 - val_loss: 1.8236 - val_accuracy: 0.3576
Epoch 69/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6809 -
accuracy: 0.4028 - val_loss: 1.8083 - val_accuracy: 0.3507
Epoch 70/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6883 -
accuracy: 0.3898 - val_loss: 1.8126 - val_accuracy: 0.3472
Epoch 71/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6561 -
accuracy: 0.4036 - val_loss: 1.8212 - val_accuracy: 0.3576
Epoch 72/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6646 -
accuracy: 0.4054 - val_loss: 1.8130 - val_accuracy: 0.3542
Epoch 73/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6601 -
accuracy: 0.3976 - val_loss: 1.8044 - val_accuracy: 0.3715
```

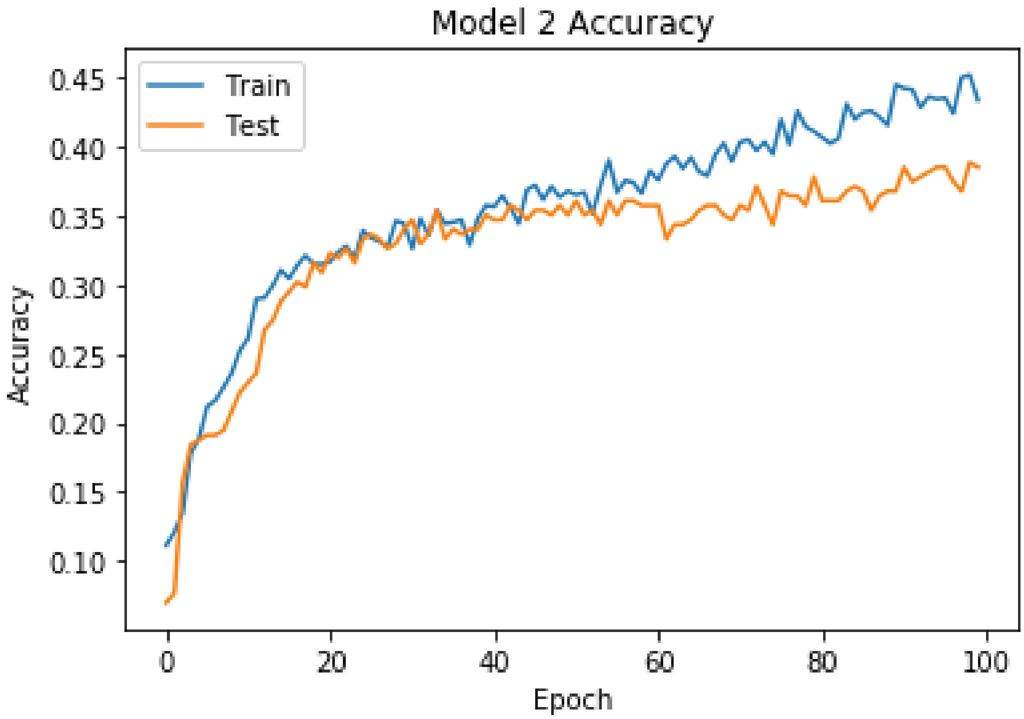
```
Epoch 74/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6454 -
accuracy: 0.4036 - val_loss: 1.8113 - val_accuracy: 0.3576
Epoch 75/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6596 -
accuracy: 0.3950 - val_loss: 1.8134 - val_accuracy: 0.3438
Epoch 76/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6561 -
accuracy: 0.4201 - val_loss: 1.8069 - val_accuracy: 0.3681
Epoch 77/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6491 -
accuracy: 0.4019 - val_loss: 1.7974 - val_accuracy: 0.3646
Epoch 78/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6367 -
accuracy: 0.4262 - val_loss: 1.7990 - val_accuracy: 0.3646
Epoch 79/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6279 -
accuracy: 0.4149 - val_loss: 1.8076 - val_accuracy: 0.3576
Epoch 80/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6302 -
accuracy: 0.4115 - val_loss: 1.8066 - val_accuracy: 0.3785
Epoch 81/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6369 -
accuracy: 0.4071 - val_loss: 1.7964 - val_accuracy: 0.3611
Epoch 82/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6250 -
accuracy: 0.4028 - val_loss: 1.8016 - val_accuracy: 0.3611
Epoch 83/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6307 -
accuracy: 0.4062 - val_loss: 1.8007 - val_accuracy: 0.3611
Epoch 84/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6155 -
accuracy: 0.4314 - val_loss: 1.7934 - val_accuracy: 0.3681
Epoch 85/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6101 -
accuracy: 0.4201 - val_loss: 1.7893 - val_accuracy: 0.3715
Epoch 86/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6031 -
accuracy: 0.4253 - val_loss: 1.7882 - val_accuracy: 0.3681
Epoch 87/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.6030 -
accuracy: 0.4262 - val_loss: 1.7921 - val_accuracy: 0.3542
Epoch 88/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5897 -
accuracy: 0.4219 - val_loss: 1.7818 - val_accuracy: 0.3646
Epoch 89/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5919 -
accuracy: 0.4158 - val_loss: 1.7994 - val_accuracy: 0.3681
```

```
Epoch 90/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5758 -
accuracy: 0.4453 - val_loss: 1.7865 - val_accuracy: 0.3681
Epoch 91/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5876 -
accuracy: 0.4427 - val_loss: 1.7888 - val_accuracy: 0.3854
Epoch 92/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5819 -
accuracy: 0.4418 - val_loss: 1.7801 - val_accuracy: 0.3750
Epoch 93/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5756 -
accuracy: 0.4288 - val_loss: 1.7827 - val_accuracy: 0.3785
Epoch 94/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5535 -
accuracy: 0.4366 - val_loss: 1.7838 - val_accuracy: 0.3819
Epoch 95/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5777 -
accuracy: 0.4349 - val_loss: 1.7758 - val_accuracy: 0.3854
Epoch 96/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5571 -
accuracy: 0.4358 - val_loss: 1.7730 - val_accuracy: 0.3854
Epoch 97/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5573 -
accuracy: 0.4245 - val_loss: 1.7672 - val_accuracy: 0.3750
Epoch 98/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5554 -
accuracy: 0.4505 - val_loss: 1.7687 - val_accuracy: 0.3681
Epoch 99/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5494 -
accuracy: 0.4523 - val_loss: 1.7657 - val_accuracy: 0.3889
Epoch 100/100
1152/1152 [==============================] - 2s 2ms/step - loss: 1.5584 -
accuracy: 0.4340 - val_loss: 1.7683 - val_accuracy: 0.3854
```

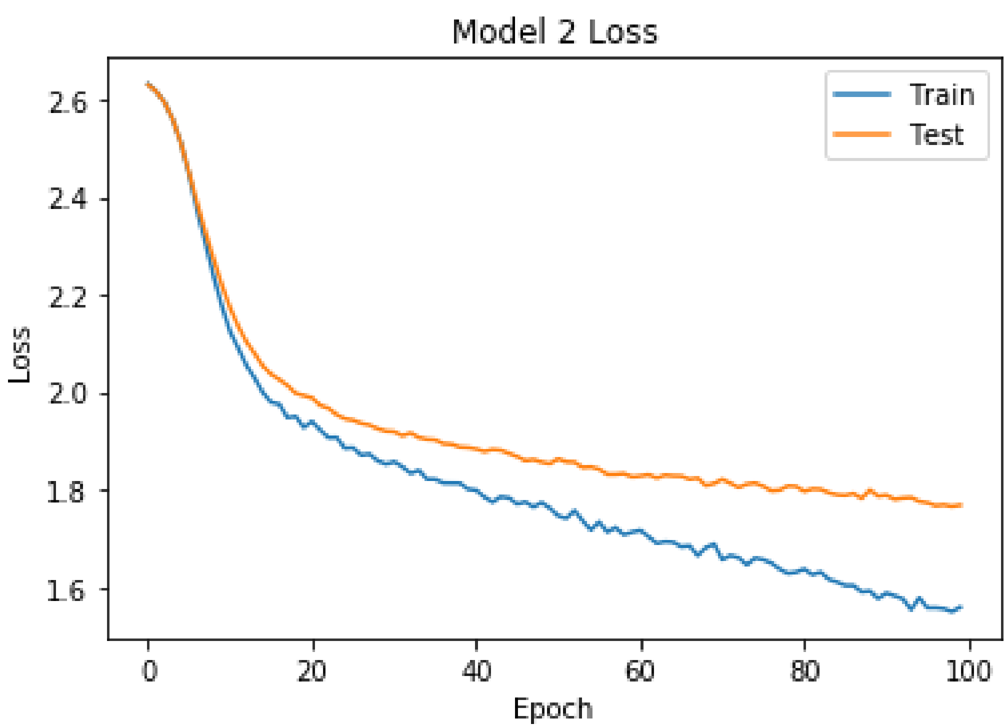
[0]:

```
plt.plot(model_2_history.history['accuracy'])
plt.plot(model_2_history.history['val_accuracy'])
plt.title('Model 2 Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
plt.plot(model_2_history.history['loss'])
plt.plot(model_2_history.history['val_loss'])
plt.title('Model 2 Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
```

```
plt.show()
```



Model 2 Accuracy
Train
Test
Accuracy
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100
Epoch



Model 2 Loss
Train
Test
Loss
2.6
2.4
2.2
2.0
1.8
1.6
0
20
40
60
80
100
Epoch

[0]:
```python
# Save model and weights
model_name = 'Model_2.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model_2.save(model_path)
print('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model_2.to_json()
with open("model_2_json.json", "w") as json_file:
    json_file.write(model_json)
```

```
Save model and weights at /home/subodh/Second Semester/ML/Random
Project/Audio/saved_models/Model_2.h5
```

1.26 Confusion Matrix

[0]:
```python
# the confusion matrix heat map plot
def print_confusion_matrix(confusion_matrix, class_names, figsize = (15,15),␣
↪fontsize=14):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
        bottom, top = heatmap.get_ylim()
        heatmap.set_ylim(bottom + 0.5, top - 0.5)
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")

    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,␣
↪ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,␣
↪ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Gender recode function
def gender(row):
    if row == 'female_disgust' or 'female_fear' or 'female_happy' or␣
↪'female_sad' or 'female_surprise' or 'female_neutral':
        return 'female'
```

```
    elif row == 'male_angry' or 'male_fear' or 'male_happy' or 'male_sad' or␣
↪'male_surprise' or 'male_neutral' or 'male_disgust':
        return 'male'
```

2 Experiment 1: Test the models trained on RAVDESS on the same dataset (Randomized split)

Back to Experiments and Results

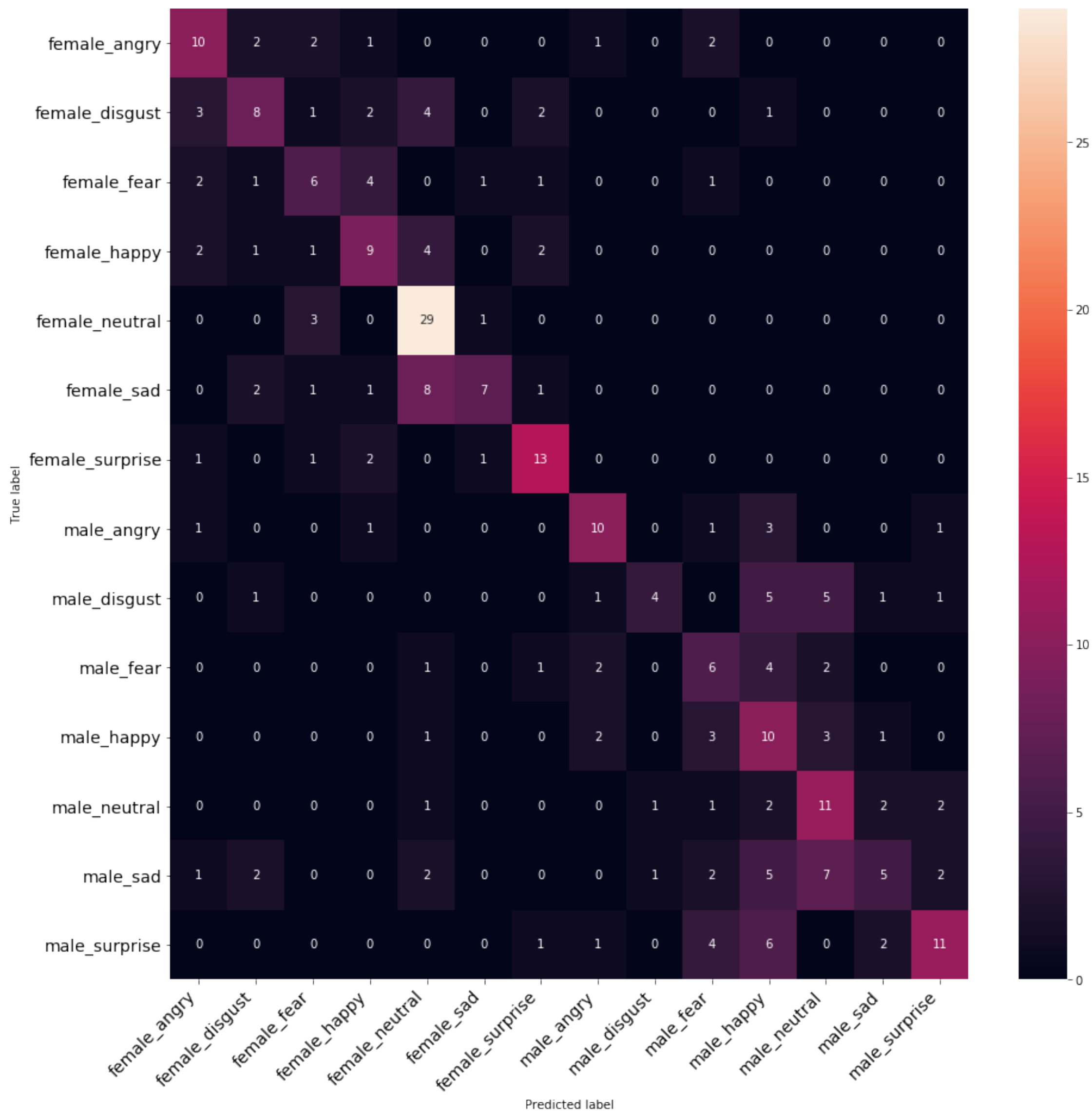
2.1 Load from file and print results

[0]:
```
def load_and_print_results(filename, json_filename):
    # loading json and model architecture
    json_file = open(json_filename, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights("saved_models/" + filename)
    print("Loaded model from disk")

    # Keras optimiser
    opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
    loaded_model.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics=['accuracy'])
    score = loaded_model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))


    #print(labels.classes_)
    preds = loaded_model.predict(X_test, batch_size = 16, verbose = 1)
    preds = preds.argmax(axis = 1)
    # predictions
    preds = preds.astype(int).flatten()
    preds = (labels.inverse_transform((preds)))
    preds = pd.DataFrame({'predictedvalues': preds})

    # Actual labels
    actual=y_test.argmax(axis=1)
    actual = actual.astype(int).flatten()
    actual = (labels.inverse_transform((actual)))
    actual = pd.DataFrame({'actualvalues': actual})

    # Lets combined both of them into a single dataframe
    rav_finaldf = actual.join(preds)
```

```
    print(rav_finaldf[170:180])
    rav_finaldf.to_csv('Predictions.csv', index = False)
    rav_finaldf.groupby('predictedvalues').count()


    rav_finaldf = pd.read_csv('Predictions.csv')
    classes = rav_finaldf.actualvalues.unique()
    classes.sort()

    c = confusion_matrix(rav_finaldf.actualvalues, rav_finaldf.predictedvalues)
    print(accuracy_score(rav_finaldf.actualvalues, rav_finaldf.predictedvalues))
    print_confusion_matrix(c, class_names = classes)

    classes = rav_finaldf.actualvalues.unique()
    classes.sort()
    print(classification_report(rav_finaldf.actualvalues, rav_finaldf.
 ↪predictedvalues, target_names=classes))

load_and_print_results('Model_1.h5', 'model_1_json.json')
```

```
Loaded model from disk
accuracy: 48.26%
288/288 [==============================] - 0s 2ms/step
       actualvalues predictedvalues
170   male_surprise      male_happy
171    male_neutral    male_neutral
172    female_angry    female_angry
173      female_sad      female_sad
174   male_surprise        male_sad
175  female_neutral  female_neutral
176      male_happy      male_happy
177    male_neutral    male_neutral
178    female_angry    female_angry
179    female_angry  female_disgust
0.4826388888888889
                 precision    recall  f1-score   support

   female_angry       0.50      0.56      0.53        18
 female_disgust       0.47      0.38      0.42        21
    female_fear       0.40      0.38      0.39        16
   female_happy       0.45      0.47      0.46        19
 female_neutral       0.58      0.88      0.70        33
     female_sad       0.70      0.35      0.47        20
female_surprise       0.62      0.72      0.67        18
     male_angry       0.59      0.59      0.59        17
   male_disgust       0.67      0.22      0.33        18
      male_fear       0.30      0.38      0.33        16
```

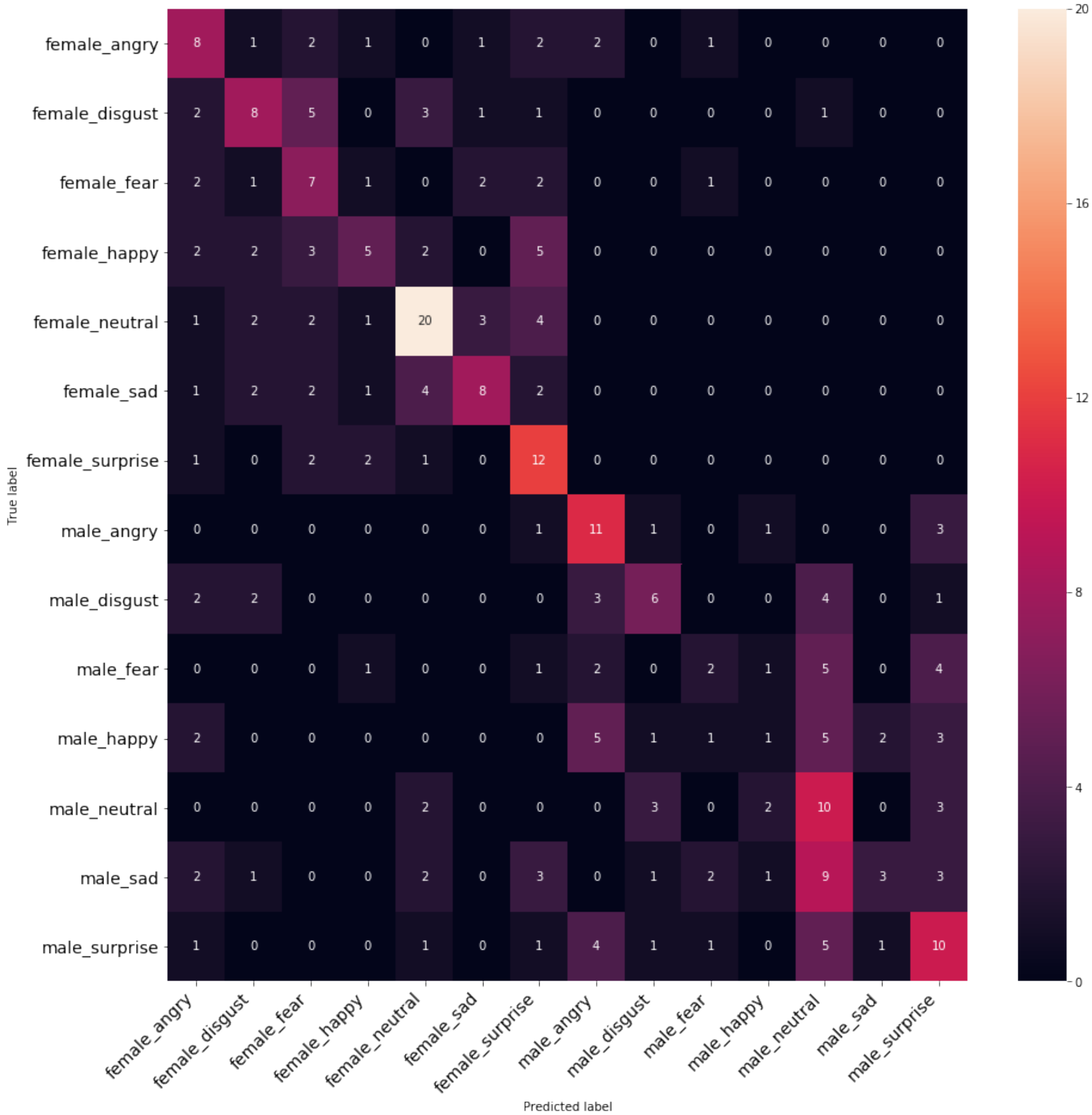
```
    male_happy       0.28      0.50      0.36        20
  male_neutral       0.39      0.55      0.46        20
      male_sad       0.45      0.19      0.26        27
 male_surprise       0.65      0.44      0.52        25

      accuracy                           0.48       288
     macro avg       0.50      0.47      0.46       288
  weighted avg       0.51      0.48      0.47       288
```



[0]:
```
load_and_print_results('Model_2.h5', 'model_2_json.json')
```

```
Loaded model from disk
```

```
accuracy: 38.54%
288/288 [==============================] - 0s 487us/step
       actualvalues  predictedvalues
170   male_surprise       male_angry
171    male_neutral     male_disgust
172    female_angry  female_surprise
173      female_sad   female_disgust
174   male_surprise     female_angry
175  female_neutral   female_neutral
176      male_happy       male_angry
177    male_neutral       male_happy
178    female_angry     female_angry
179    female_angry     female_angry
0.3854166666666667
                 precision    recall  f1-score   support

   female_angry       0.33      0.44      0.38        18
 female_disgust       0.42      0.38      0.40        21
    female_fear       0.30      0.44      0.36        16
   female_happy       0.42      0.26      0.32        19
 female_neutral       0.57      0.61      0.59        33
     female_sad       0.53      0.40      0.46        20
female_surprise       0.35      0.67      0.46        18
     male_angry       0.41      0.65      0.50        17
   male_disgust       0.46      0.33      0.39        18
      male_fear       0.25      0.12      0.17        16
     male_happy       0.17      0.05      0.08        20
   male_neutral       0.26      0.50      0.34        20
       male_sad       0.50      0.11      0.18        27
  male_surprise       0.37      0.40      0.38        25

       accuracy                           0.39       288
      macro avg       0.38      0.38      0.36       288
   weighted avg       0.40      0.39      0.36       288
```



As it can be easily seen, both the models do not perform very well when we train and test them on RAVDESS dataset. We will explore the gender based and emotion based results in the following cells. After that we will move on the testing to combined dataset and check the performance there.

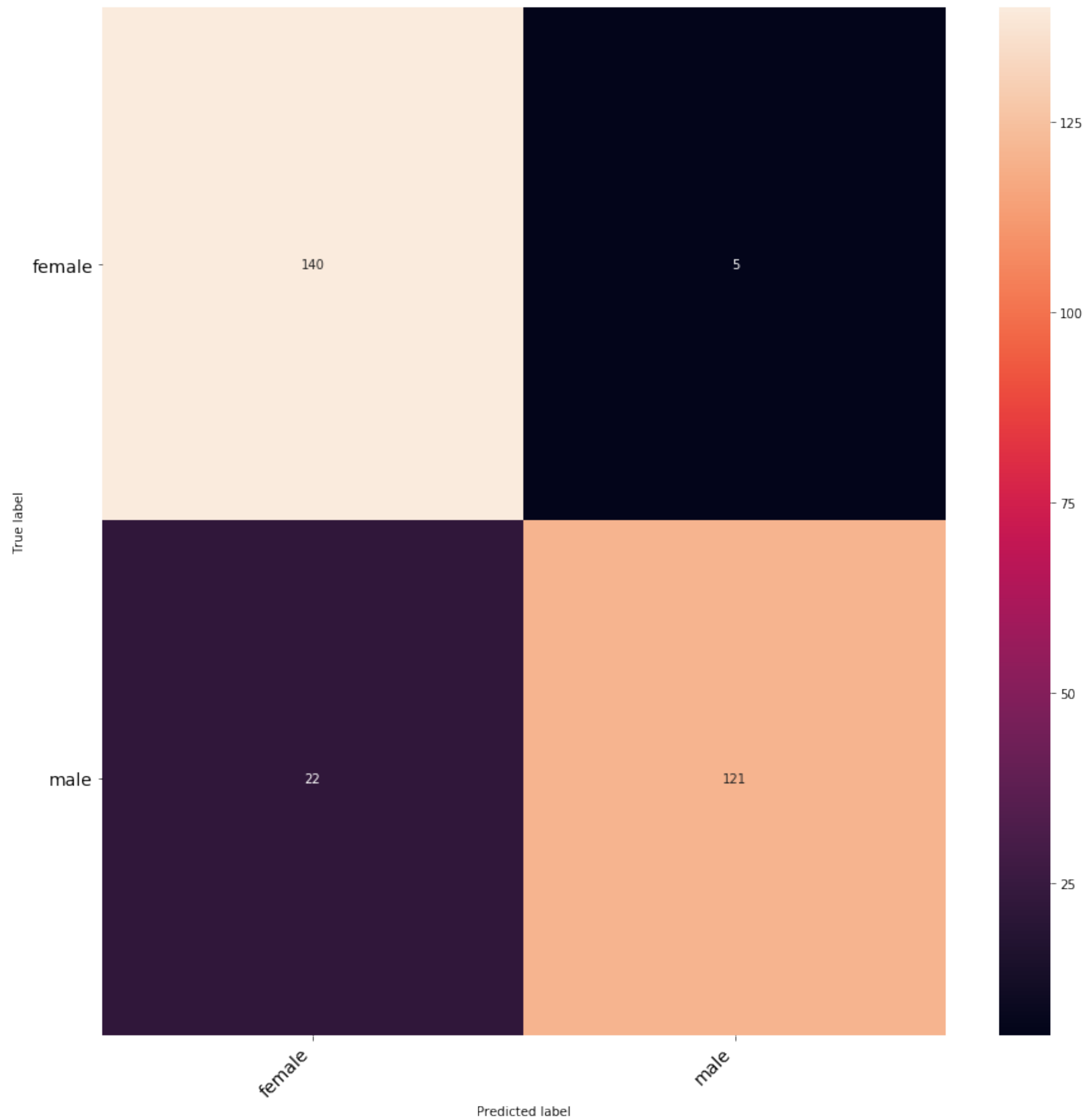
2.2 So, let's group the gender and check for the results

[0]:
```
modidf = rav_finaldf
modidf['actualvalues'] = rav_finaldf.actualvalues.replace({'female_angry':
 ↪'female'
                                            , 'female_disgust':'female'
                                            , 'female_fear':'female'
```

```
                                        , 'female_happy':'female'
                                        , 'female_sad':'female'
                                        , 'female_surprise':'female'
                                        , 'female_neutral':'female'
                                        , 'male_angry':'male'
                                        , 'male_fear':'male'
                                        , 'male_happy':'male'
                                        , 'male_sad':'male'
                                        , 'male_surprise':'male'
                                        , 'male_neutral':'male'
                                        , 'male_disgust':'male'
                                       })

modidf['predictedvalues'] = rav_finaldf.predictedvalues.replace({'female_angry':
 ↪'female'
                                        , 'female_disgust':'female'
                                        , 'female_fear':'female'
                                        , 'female_happy':'female'
                                        , 'female_sad':'female'
                                        , 'female_surprise':'female'
                                        , 'female_neutral':'female'
                                        , 'male_angry':'male'
                                        , 'male_fear':'male'
                                        , 'male_happy':'male'
                                        , 'male_sad':'male'
                                        , 'male_surprise':'male'
                                        , 'male_neutral':'male'
                                        , 'male_disgust':'male'
                                       })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
0.90625
              precision    recall  f1-score   support

      female       0.86      0.97      0.91       145
        male       0.96      0.85      0.90       143
```

```
    accuracy                           0.91       288
   macro avg       0.91      0.91      0.91       288
weighted avg       0.91      0.91      0.91       288
```



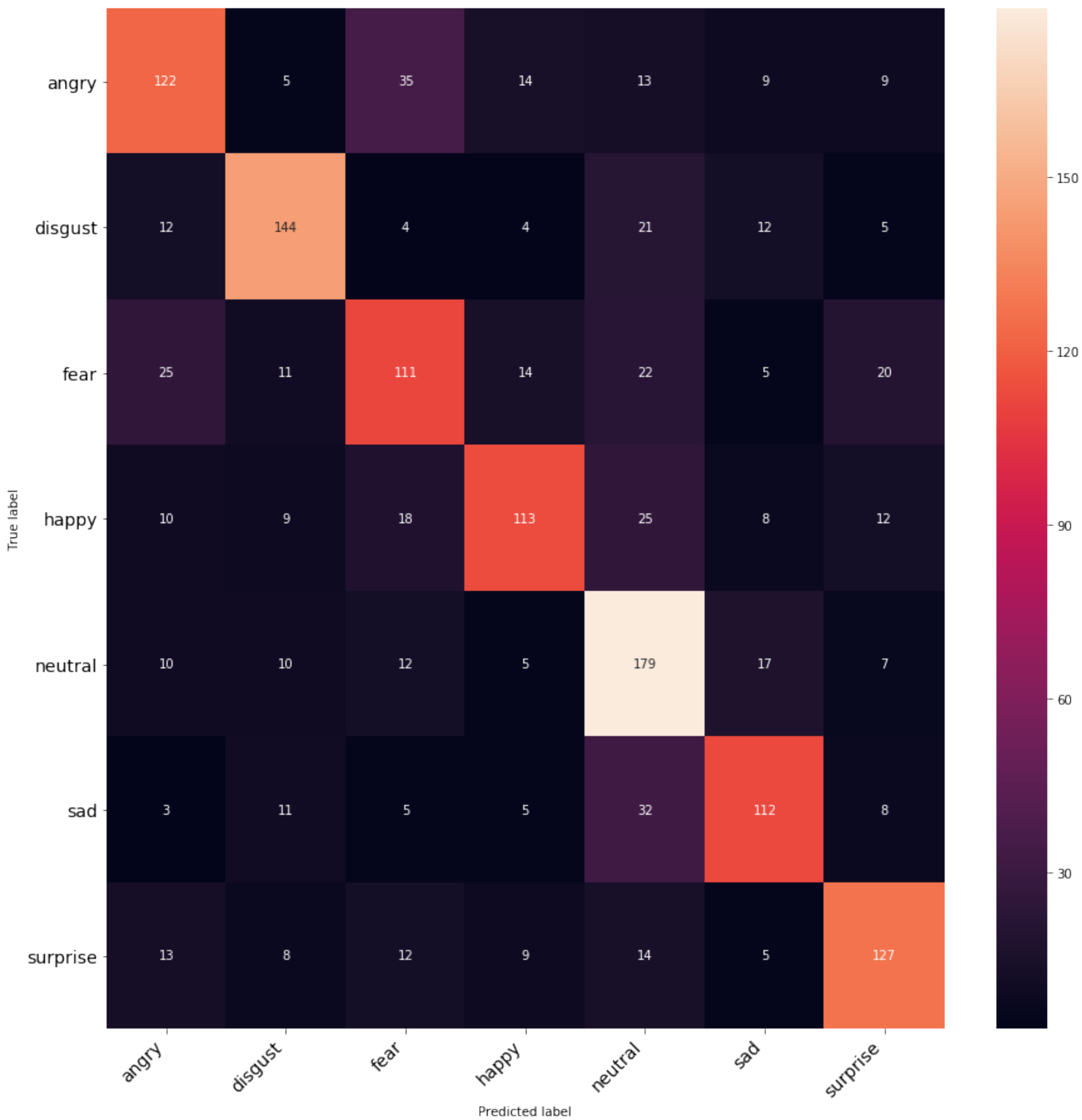
2.3 And now, let's group together the emotions and look for the performance

[0]:
```
modidf = pd.read_csv("Predictions.csv")
modidf['actualvalues'] = modidf.actualvalues.replace({'female_angry':'angry'
                                        , 'female_disgust':'disgust'
                                        , 'female_fear':'fear'
```

```
                                        , 'female_happy':'happy'
                                        , 'female_sad':'sad'
                                        , 'female_surprise':'surprise'
                                        , 'female_neutral':'neutral'
                                        , 'male_angry':'angry'
                                        , 'male_fear':'fear'
                                        , 'male_happy':'happy'
                                        , 'male_sad':'sad'
                                        , 'male_surprise':'surprise'
                                        , 'male_neutral':'neutral'
                                        , 'male_disgust':'disgust'
                                       })

modidf['predictedvalues'] = modidf.predictedvalues.replace({'female_angry':
 ↪'angry'
                                        , 'female_disgust':'disgust'
                                        , 'female_fear':'fear'
                                        , 'female_happy':'happy'
                                        , 'female_sad':'sad'
                                        , 'female_surprise':'surprise'
                                        , 'female_neutral':'neutral'
                                        , 'male_angry':'angry'
                                        , 'male_fear':'fear'
                                        , 'male_happy':'happy'
                                        , 'male_sad':'sad'
                                        , 'male_surprise':'surprise'
                                        , 'male_neutral':'neutral'
                                        , 'male_disgust':'disgust'
                                       })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)

# Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
0.6412429378531074
              precision    recall  f1-score   support
```

angry	0.63	0.59	0.61	207
disgust	0.73	0.71	0.72	202
fear	0.56	0.53	0.55	208
happy	0.69	0.58	0.63	195
neutral	0.58	0.75	0.66	240
sad	0.67	0.64	0.65	176
surprise	0.68	0.68	0.68	188
accuracy			0.64	1416
macro avg	0.65	0.64	0.64	1416
weighted avg	0.64	0.64	0.64	1416



[0]:

```

```

3 Experiment 2: Test the models trained on RAVDESS on the combined dataset

Back to Experiments and Results

3.1 Now we repeat the whole process for combined dataset

[0]:

```
com = combined_df

X_test = com.drop(['path', 'labels','source'], axis=1)
y_test = com.labels
mean = np.mean(X_test, axis=0)
std = np.std(X_test, axis=0)
X_test = (X_test - mean)/std
X_test = np.array(X_test)
y_test = np.array(y_test)
y_test = np_utils.to_categorical(labels.fit_transform(y_test))
X_test = np.expand_dims(X_test, axis=2)

X_test.shape
#print(com)
```

[0]:

```
(4720, 216, 1)
```

[0]:

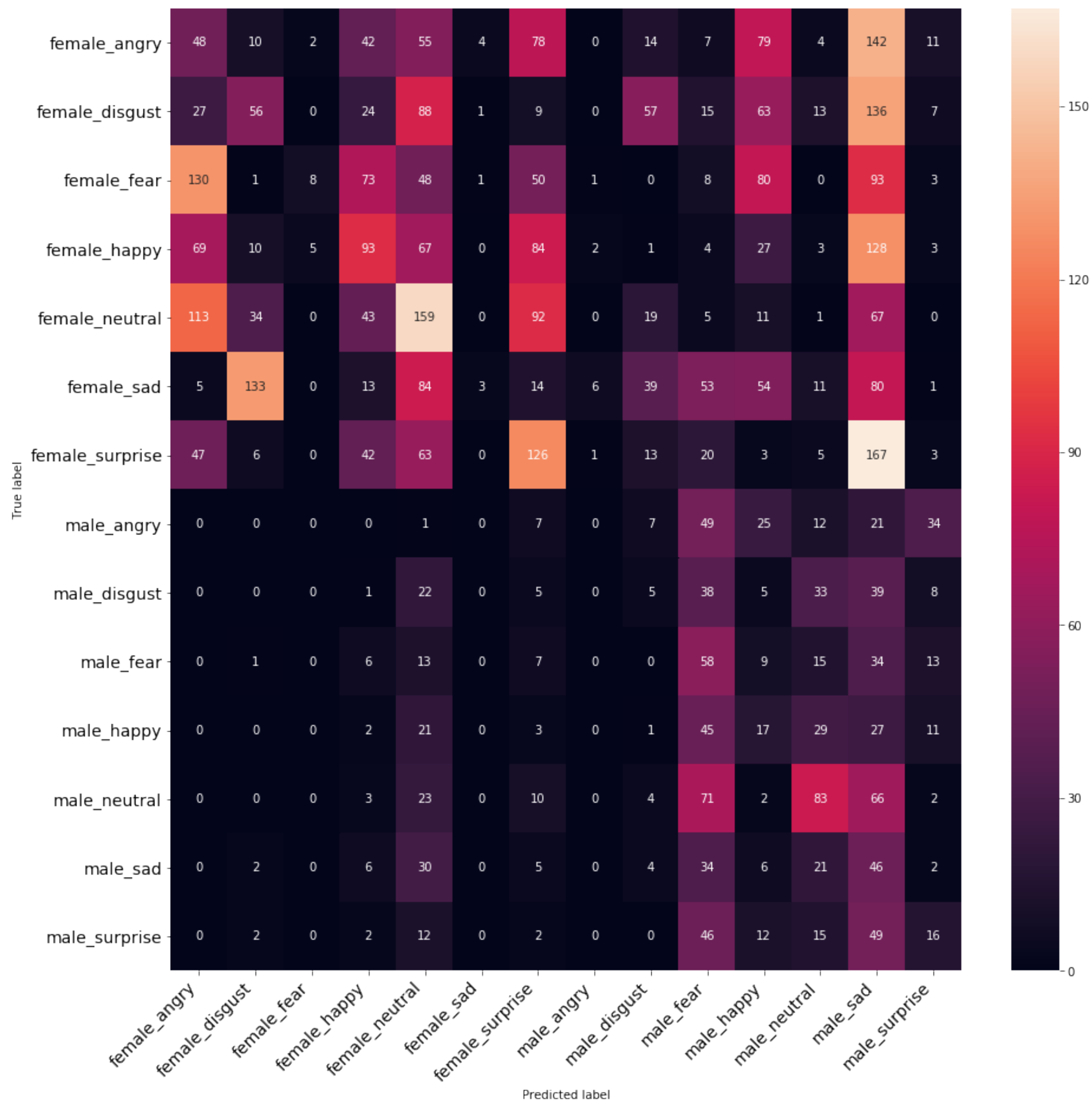
```
# loading json and model architecture
def load_and_predict_com(filename, json_filename):
    json_file = open(json_filename, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights("saved_models/" + filename)
    print("Loaded model from disk")

    # Keras optimiser
    opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
    loaded_model.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics=['accuracy'])
    score = loaded_model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))


    #print(labels.classes_)
```

```
    preds = loaded_model.predict(X_test, batch_size = 16, verbose = 1)
    preds = preds.argmax(axis = 1)
    # predictions
    preds = preds.astype(int).flatten()
    preds = (labels.inverse_transform((preds)))
    preds = pd.DataFrame({'predictedvalues': preds})

    # Actual labels
    actual = y_test.argmax(axis=1)
    actual = actual.astype(int).flatten()
    actual = (labels.inverse_transform((actual)))
    actual = pd.DataFrame({'actualvalues': actual})

    # Lets combined both of them into a single dataframe
    com_ = actual.join(preds)
    print(com_[170:180])
    com_.to_csv('Predictions_com.csv', index = False)
    com_.groupby('predictedvalues').count()


    com_ = pd.read_csv('Predictions_com.csv')
    classes = com_.actualvalues.unique()
    classes.sort()

    c = confusion_matrix(com_.actualvalues, com_.predictedvalues)
    print(accuracy_score(com_.actualvalues, com_.predictedvalues))
    print_confusion_matrix(c, class_names = classes)

    classes = com_.actualvalues.unique()
    classes.sort()
    print(classification_report(com_.actualvalues, com_.predictedvalues,␣
↪target_names=classes))
```

[0]:
```
load_and_predict_com('Model_1.h5', 'model_1_json.json')
```

```
Loaded model from disk
accuracy: 15.21%
4720/4720 [==============================] - 7s 1ms/step
      actualvalues  predictedvalues
170  male_surprise        male_fear
171   male_disgust        male_fear
172   male_neutral     female_happy
173  male_surprise  female_surprise
174     male_happy        male_fear
175       male_sad       male_happy
176      male_fear        male_fear
177       male_sad   female_disgust
```

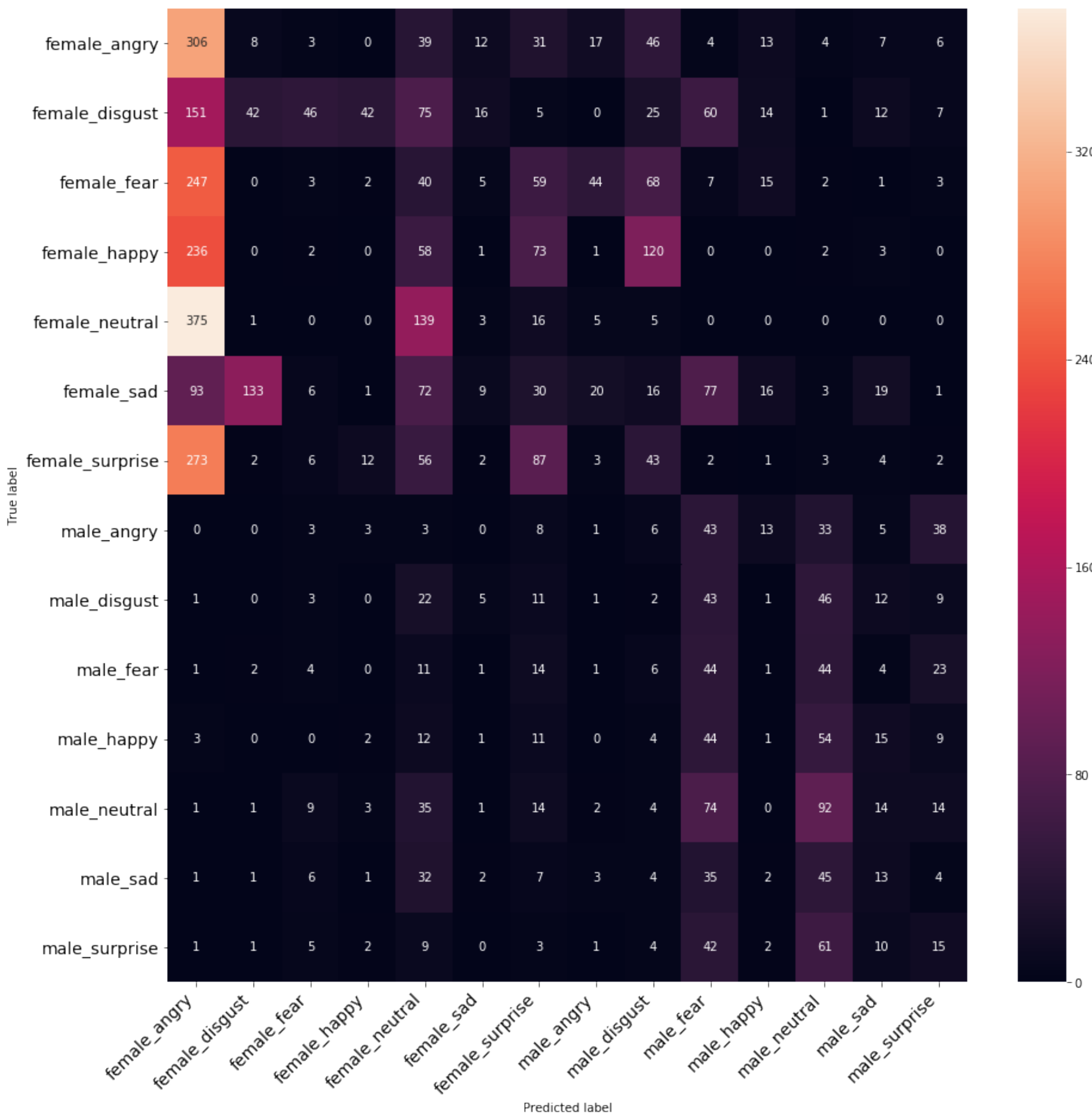
```
178   male_neutral         male_sad
179     male_angry        male_fear
0.1521186440677966
                 precision    recall  f1-score   support

   female_angry       0.11      0.10      0.10       496
 female_disgust       0.22      0.11      0.15       496
    female_fear       0.53      0.02      0.03       496
   female_happy       0.27      0.19      0.22       496
 female_neutral       0.23      0.29      0.26       544
     female_sad       0.33      0.01      0.01       496
female_surprise       0.26      0.25      0.26       496
     male_angry       0.00      0.00      0.00       156
   male_disgust       0.03      0.03      0.03       156
      male_fear       0.13      0.37      0.19       156
     male_happy       0.04      0.11      0.06       156
   male_neutral       0.34      0.31      0.33       264
       male_sad       0.04      0.29      0.07       156
  male_surprise       0.14      0.10      0.12       156

       accuracy                           0.15      4720
      macro avg       0.19      0.16      0.13      4720
   weighted avg       0.24      0.15      0.14      4720
```




```
[0]: load_and_predict_com('Model_2.h5', 'model_2_json.json')
```

```
Loaded model from disk
accuracy: 15.97%
4720/4720 [==============================] - 2s 361us/step
      actualvalues  predictedvalues
170  male_surprise        male_fear
171   male_disgust        male_fear
172   male_neutral     female_happy
173  male_surprise     male_disgust
174     male_happy        male_fear
175       male_sad     male_disgust
176      male_fear        male_fear
```

```
177       male_sad  female_surprise
178   male_neutral     male_disgust
179     male_angry        male_fear
0.15974576271186441
                 precision    recall  f1-score   support

   female_angry       0.18      0.62      0.28       496
 female_disgust       0.22      0.08      0.12       496
    female_fear       0.03      0.01      0.01       496
   female_happy       0.00      0.00      0.00       496
 female_neutral       0.23      0.26      0.24       544
     female_sad       0.16      0.02      0.03       496
female_surprise       0.24      0.18      0.20       496
     male_angry       0.01      0.01      0.01       156
   male_disgust       0.01      0.01      0.01       156
      male_fear       0.09      0.28      0.14       156
     male_happy       0.01      0.01      0.01       156
   male_neutral       0.24      0.35      0.28       264
       male_sad       0.11      0.08      0.09       156
  male_surprise       0.11      0.10      0.10       156

       accuracy                           0.16      4720
      macro avg       0.12      0.14      0.11      4720
   weighted avg       0.14      0.16      0.12      4720
```



4 Experiment 3: Train the models on Combined dataset and check for the performance on the same

Back to Experiments and Results

4.1 Now we trained the two models on combined data split into train and test set

[0]:
```
X_train, y_train, X_test, y_test, labels = process_data(combined_df, 0.3)
```

[0]:
```
model_1_combined = build_model_1()
model_1_combined_history=model_1_combined.fit(X_train, y_train, batch_size=16,␣
↪epochs=100, validation_data=(X_test, y_test))
```

```
Model: "sequential_11"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_29 (Conv1D)           (None, 216, 256)          2304

_________________________________________________________________
activation_40 (Activation)   (None, 216, 256)          0

_________________________________________________________________
conv1d_30 (Conv1D)           (None, 216, 256)          524544

_________________________________________________________________
batch_normalization_5 (Batch (None, 216, 256)          1024

_________________________________________________________________
activation_41 (Activation)   (None, 216, 256)          0

_________________________________________________________________
dropout_14 (Dropout)         (None, 216, 256)          0

_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 27, 256)           0

_________________________________________________________________
conv1d_31 (Conv1D)           (None, 27, 128)           262272

_________________________________________________________________
activation_42 (Activation)   (None, 27, 128)           0

_________________________________________________________________
conv1d_32 (Conv1D)           (None, 27, 128)           131200

_________________________________________________________________
activation_43 (Activation)   (None, 27, 128)           0

_________________________________________________________________
conv1d_33 (Conv1D)           (None, 27, 128)           131200

_________________________________________________________________
activation_44 (Activation)   (None, 27, 128)           0

_________________________________________________________________
conv1d_34 (Conv1D)           (None, 27, 128)           131200

_________________________________________________________________
batch_normalization_6 (Batch (None, 27, 128)           512

_________________________________________________________________
activation_45 (Activation)   (None, 27, 128)           0

_________________________________________________________________
dropout_15 (Dropout)         (None, 27, 128)           0

_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 3, 128)            0

_________________________________________________________________
conv1d_35 (Conv1D)           (None, 3, 64)             65600

_________________________________________________________________
activation_46 (Activation)   (None, 3, 64)             0
```

```
_________________________________________________________________
conv1d_36 (Conv1D)           (None, 3, 64)             32832
_________________________________________________________________
activation_47 (Activation)   (None, 3, 64)             0
_________________________________________________________________
flatten_5 (Flatten)          (None, 192)               0
_________________________________________________________________
dense_12 (Dense)             (None, 14)                2702
_________________________________________________________________
activation_48 (Activation)   (None, 14)                0
=================================================================
Total params: 1,285,390
Trainable params: 1,284,622
Non-trainable params: 768
_________________________________________________________________
Train on 3304 samples, validate on 1416 samples
Epoch 1/100
3304/3304 [==============================] - 24s 7ms/step - loss: 2.3324 -
accuracy: 0.2536 - val_loss: 2.5883 - val_accuracy: 0.2373
Epoch 2/100
3304/3304 [==============================] - 23s 7ms/step - loss: 1.9762 -
accuracy: 0.3732 - val_loss: 2.4298 - val_accuracy: 0.2719
Epoch 3/100
3304/3304 [==============================] - 24s 7ms/step - loss: 1.8044 -
accuracy: 0.4210 - val_loss: 2.0784 - val_accuracy: 0.3644
Epoch 4/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.6723 -
accuracy: 0.4782 - val_loss: 1.8810 - val_accuracy: 0.4675
Epoch 5/100
3304/3304 [==============================] - 26s 8ms/step - loss: 1.5728 -
accuracy: 0.5185 - val_loss: 1.7722 - val_accuracy: 0.5056
Epoch 6/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.4844 -
accuracy: 0.5512 - val_loss: 1.6919 - val_accuracy: 0.5346
Epoch 7/100
3304/3304 [==============================] - 25s 7ms/step - loss: 1.3960 -
accuracy: 0.5769 - val_loss: 1.6197 - val_accuracy: 0.5650
Epoch 8/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.3344 -
accuracy: 0.6002 - val_loss: 1.5562 - val_accuracy: 0.5770
Epoch 9/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.2597 -
accuracy: 0.6159 - val_loss: 1.5113 - val_accuracy: 0.5791
Epoch 10/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.2184 -
accuracy: 0.6271 - val_loss: 1.4730 - val_accuracy: 0.5897
Epoch 11/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.1602 -
```

```
accuracy: 0.6456 - val_loss: 1.4111 - val_accuracy: 0.6045
Epoch 12/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.1090 -
accuracy: 0.6580 - val_loss: 1.3638 - val_accuracy: 0.6264
Epoch 13/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.0714 -
accuracy: 0.6707 - val_loss: 1.3368 - val_accuracy: 0.6109
Epoch 14/100
3304/3304 [==============================] - 25s 8ms/step - loss: 1.0281 -
accuracy: 0.6828 - val_loss: 1.3290 - val_accuracy: 0.6081
Epoch 15/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.9957 -
accuracy: 0.6925 - val_loss: 1.2867 - val_accuracy: 0.6342
Epoch 16/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.9693 -
accuracy: 0.7001 - val_loss: 1.2396 - val_accuracy: 0.6292
Epoch 17/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.9366 -
accuracy: 0.7091 - val_loss: 1.2232 - val_accuracy: 0.6356
Epoch 18/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.9103 -
accuracy: 0.7222 - val_loss: 1.1878 - val_accuracy: 0.6490
Epoch 19/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.8966 -
accuracy: 0.7197 - val_loss: 1.1830 - val_accuracy: 0.6525
Epoch 20/100
3304/3304 [==============================] - 35s 10ms/step - loss: 0.8708 -
accuracy: 0.7273 - val_loss: 1.1575 - val_accuracy: 0.6631
Epoch 21/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.8455 -
accuracy: 0.7306 - val_loss: 1.1314 - val_accuracy: 0.6674
Epoch 22/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.8274 -
accuracy: 0.7446 - val_loss: 1.1191 - val_accuracy: 0.6857
Epoch 23/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.8126 -
accuracy: 0.7446 - val_loss: 1.1085 - val_accuracy: 0.6610
Epoch 24/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.7970 -
accuracy: 0.7476 - val_loss: 1.1166 - val_accuracy: 0.6610
Epoch 25/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.7739 -
accuracy: 0.7548 - val_loss: 1.0901 - val_accuracy: 0.6681
Epoch 26/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.7637 -
accuracy: 0.7545 - val_loss: 1.0711 - val_accuracy: 0.6780
Epoch 27/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.7493 -
```

```
accuracy: 0.7676 - val_loss: 1.0581 - val_accuracy: 0.6780
Epoch 28/100
3304/3304 [==============================] - 29s 9ms/step - loss: 0.7316 -
accuracy: 0.7630 - val_loss: 1.0507 - val_accuracy: 0.6773
Epoch 29/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.7171 -
accuracy: 0.7766 - val_loss: 1.0331 - val_accuracy: 0.6857
Epoch 30/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.7054 -
accuracy: 0.7791 - val_loss: 1.0294 - val_accuracy: 0.6893
Epoch 31/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.6875 -
accuracy: 0.7912 - val_loss: 1.0183 - val_accuracy: 0.6935
Epoch 32/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.6750 -
accuracy: 0.7915 - val_loss: 1.0101 - val_accuracy: 0.6942
Epoch 33/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.6509 -
accuracy: 0.8018 - val_loss: 0.9953 - val_accuracy: 0.6963
Epoch 34/100
3304/3304 [==============================] - 36s 11ms/step - loss: 0.6525 -
accuracy: 0.7981 - val_loss: 0.9749 - val_accuracy: 0.6900
Epoch 35/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.6390 -
accuracy: 0.8057 - val_loss: 0.9928 - val_accuracy: 0.6949
Epoch 36/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.6298 -
accuracy: 0.8069 - val_loss: 0.9806 - val_accuracy: 0.7034
Epoch 37/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.6171 -
accuracy: 0.8039 - val_loss: 0.9869 - val_accuracy: 0.7013
Epoch 38/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.6049 -
accuracy: 0.8105 - val_loss: 0.9637 - val_accuracy: 0.7090
Epoch 39/100
3304/3304 [==============================] - 27s 8ms/step - loss: 0.5987 -
accuracy: 0.8208 - val_loss: 0.9721 - val_accuracy: 0.6893
Epoch 40/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.5862 -
accuracy: 0.8130 - val_loss: 0.9516 - val_accuracy: 0.7126
Epoch 41/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.5680 -
accuracy: 0.8278 - val_loss: 0.9707 - val_accuracy: 0.6794
Epoch 42/100
3304/3304 [==============================] - 27s 8ms/step - loss: 0.5567 -
accuracy: 0.8381 - val_loss: 0.9462 - val_accuracy: 0.6999
Epoch 43/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.5452 -
```

```
accuracy: 0.8402 - val_loss: 0.9738 - val_accuracy: 0.6921
Epoch 44/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.5363 -
accuracy: 0.8441 - val_loss: 0.9294 - val_accuracy: 0.7119
Epoch 45/100
3304/3304 [==============================] - 27s 8ms/step - loss: 0.5283 -
accuracy: 0.8417 - val_loss: 0.9260 - val_accuracy: 0.7055
Epoch 46/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.5171 -
accuracy: 0.8484 - val_loss: 0.9094 - val_accuracy: 0.7147
Epoch 47/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.5116 -
accuracy: 0.8447 - val_loss: 0.9193 - val_accuracy: 0.7133
Epoch 48/100
3304/3304 [==============================] - 27s 8ms/step - loss: 0.5005 -
accuracy: 0.8559 - val_loss: 0.9243 - val_accuracy: 0.6970
Epoch 49/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.4872 -
accuracy: 0.8644 - val_loss: 0.9109 - val_accuracy: 0.7105
Epoch 50/100
3304/3304 [==============================] - 34s 10ms/step - loss: 0.4760 -
accuracy: 0.8620 - val_loss: 0.8998 - val_accuracy: 0.7083
Epoch 51/100
3304/3304 [==============================] - 28s 8ms/step - loss: 0.4684 -
accuracy: 0.8620 - val_loss: 0.9298 - val_accuracy: 0.6949
Epoch 52/100
3304/3304 [==============================] - 34s 10ms/step - loss: 0.4611 -
accuracy: 0.8689 - val_loss: 0.8953 - val_accuracy: 0.7175
Epoch 53/100
3304/3304 [==============================] - 30s 9ms/step - loss: 0.4510 -
accuracy: 0.8665 - val_loss: 0.8899 - val_accuracy: 0.7161
Epoch 54/100
3304/3304 [==============================] - 35s 11ms/step - loss: 0.4445 -
accuracy: 0.8777 - val_loss: 0.9028 - val_accuracy: 0.6977
Epoch 55/100
3304/3304 [==============================] - 30s 9ms/step - loss: 0.4389 -
accuracy: 0.8832 - val_loss: 0.8836 - val_accuracy: 0.7154
Epoch 56/100
3304/3304 [==============================] - 26s 8ms/step - loss: 0.4224 -
accuracy: 0.8820 - val_loss: 0.8872 - val_accuracy: 0.7168
Epoch 57/100
3304/3304 [==============================] - 27s 8ms/step - loss: 0.4170 -
accuracy: 0.8880 - val_loss: 0.8849 - val_accuracy: 0.7196
Epoch 58/100
3304/3304 [==============================] - 28s 8ms/step - loss: 0.4129 -
accuracy: 0.8880 - val_loss: 0.8822 - val_accuracy: 0.7126
Epoch 59/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.3986 -
```

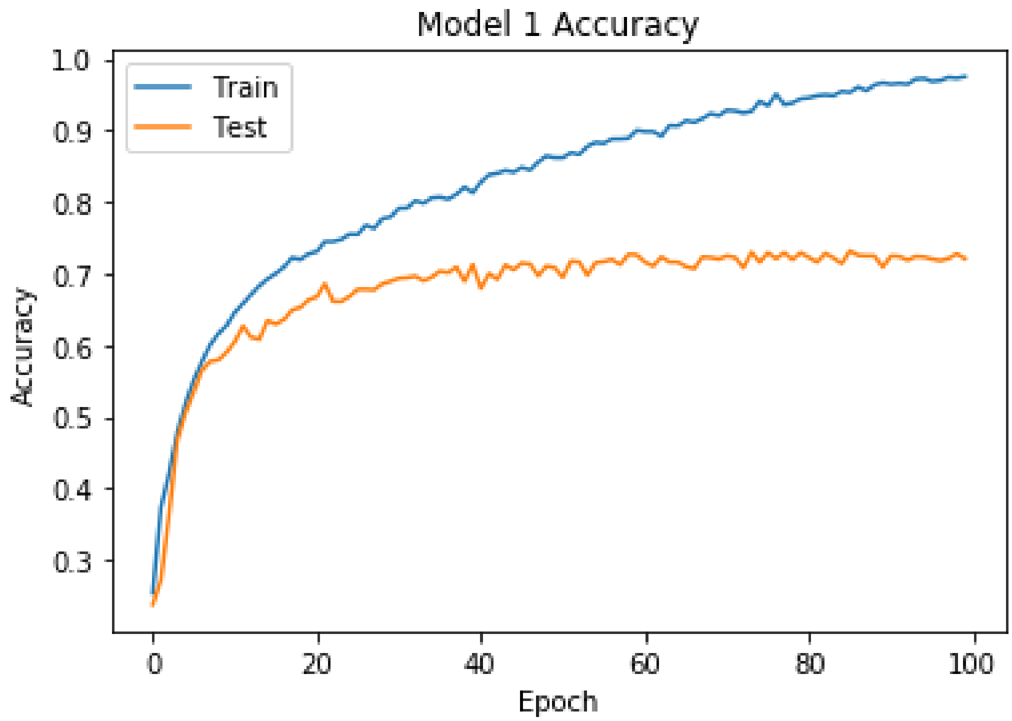
```
accuracy: 0.8889 - val_loss: 0.8889 - val_accuracy: 0.7267
Epoch 60/100
3304/3304 [==============================] - 25s 7ms/step - loss: 0.3862 -
accuracy: 0.9001 - val_loss: 0.8763 - val_accuracy: 0.7260
Epoch 61/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.3825 -
accuracy: 0.8983 - val_loss: 0.8639 - val_accuracy: 0.7161
Epoch 62/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.3807 -
accuracy: 0.8983 - val_loss: 0.8791 - val_accuracy: 0.7105
Epoch 63/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.3778 -
accuracy: 0.8919 - val_loss: 0.8507 - val_accuracy: 0.7232
Epoch 64/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.3592 -
accuracy: 0.9068 - val_loss: 0.8542 - val_accuracy: 0.7161
Epoch 65/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.3515 -
accuracy: 0.9062 - val_loss: 0.8553 - val_accuracy: 0.7161
Epoch 66/100
3304/3304 [==============================] - 25s 7ms/step - loss: 0.3438 -
accuracy: 0.9134 - val_loss: 0.8687 - val_accuracy: 0.7097
Epoch 67/100
3304/3304 [==============================] - 25s 8ms/step - loss: 0.3400 -
accuracy: 0.9113 - val_loss: 0.8747 - val_accuracy: 0.7069
Epoch 68/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.3295 -
accuracy: 0.9165 - val_loss: 0.8555 - val_accuracy: 0.7232
Epoch 69/100
3304/3304 [==============================] - 34s 10ms/step - loss: 0.3176 -
accuracy: 0.9234 - val_loss: 0.8611 - val_accuracy: 0.7225
Epoch 70/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.3168 -
accuracy: 0.9207 - val_loss: 0.8438 - val_accuracy: 0.7203
Epoch 71/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.3134 -
accuracy: 0.9283 - val_loss: 0.8346 - val_accuracy: 0.7246
Epoch 72/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2973 -
accuracy: 0.9274 - val_loss: 0.8464 - val_accuracy: 0.7225
Epoch 73/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.2913 -
accuracy: 0.9243 - val_loss: 0.8509 - val_accuracy: 0.7083
Epoch 74/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2922 -
accuracy: 0.9268 - val_loss: 0.8396 - val_accuracy: 0.7295
Epoch 75/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.2749 -
```

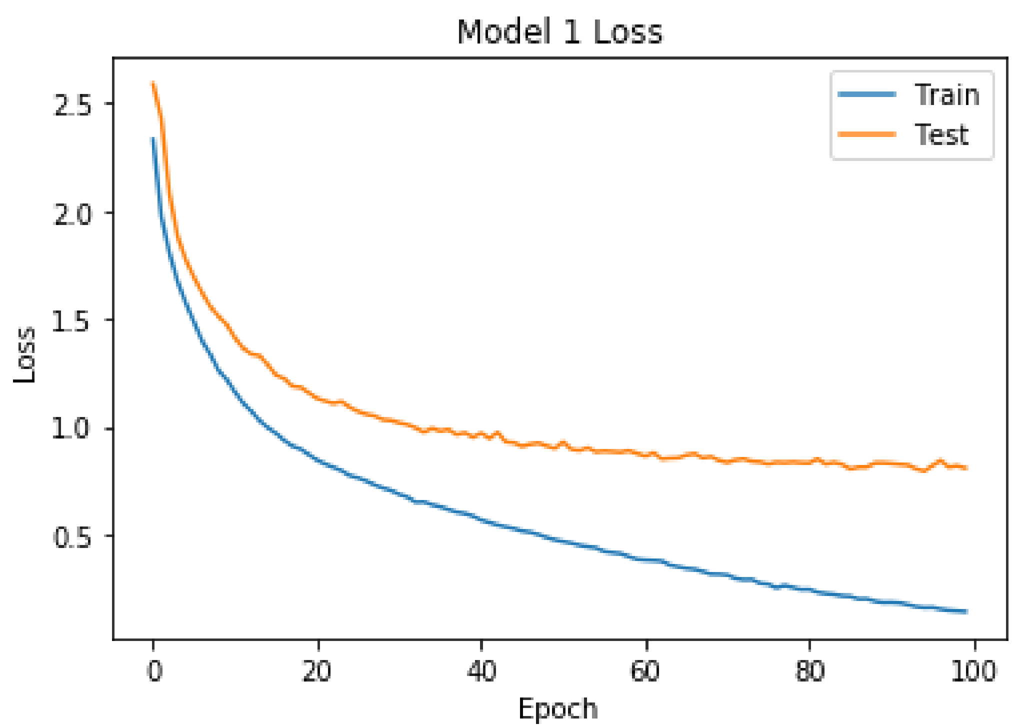
```
accuracy: 0.9407 - val_loss: 0.8368 - val_accuracy: 0.7161
Epoch 76/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.2704 -
accuracy: 0.9346 - val_loss: 0.8275 - val_accuracy: 0.7288
Epoch 77/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.2537 -
accuracy: 0.9507 - val_loss: 0.8352 - val_accuracy: 0.7210
Epoch 78/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2649 -
accuracy: 0.9361 - val_loss: 0.8334 - val_accuracy: 0.7288
Epoch 79/100
3304/3304 [==============================] - 30s 9ms/step - loss: 0.2549 -
accuracy: 0.9386 - val_loss: 0.8360 - val_accuracy: 0.7196
Epoch 80/100
3304/3304 [==============================] - 34s 10ms/step - loss: 0.2460 -
accuracy: 0.9443 - val_loss: 0.8332 - val_accuracy: 0.7288
Epoch 81/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.2466 -
accuracy: 0.9455 - val_loss: 0.8320 - val_accuracy: 0.7225
Epoch 82/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2329 -
accuracy: 0.9485 - val_loss: 0.8514 - val_accuracy: 0.7161
Epoch 83/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2270 -
accuracy: 0.9498 - val_loss: 0.8273 - val_accuracy: 0.7281
Epoch 84/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2230 -
accuracy: 0.9485 - val_loss: 0.8348 - val_accuracy: 0.7218
Epoch 85/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.2158 -
accuracy: 0.9546 - val_loss: 0.8264 - val_accuracy: 0.7140
Epoch 86/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2138 -
accuracy: 0.9531 - val_loss: 0.8046 - val_accuracy: 0.7316
Epoch 87/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2021 -
accuracy: 0.9607 - val_loss: 0.8133 - val_accuracy: 0.7253
Epoch 88/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.2027 -
accuracy: 0.9555 - val_loss: 0.8137 - val_accuracy: 0.7246
Epoch 89/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.1914 -
accuracy: 0.9637 - val_loss: 0.8323 - val_accuracy: 0.7246
Epoch 90/100
3304/3304 [==============================] - 30s 9ms/step - loss: 0.1856 -
accuracy: 0.9667 - val_loss: 0.8329 - val_accuracy: 0.7090
Epoch 91/100
3304/3304 [==============================] - 33s 10ms/step - loss: 0.1863 -
```

```
accuracy: 0.9646 - val_loss: 0.8299 - val_accuracy: 0.7246
Epoch 92/100
3304/3304 [==============================] - 31s 9ms/step - loss: 0.1837 -
accuracy: 0.9658 - val_loss: 0.8264 - val_accuracy: 0.7239
Epoch 93/100
3304/3304 [==============================] - 29s 9ms/step - loss: 0.1757 -
accuracy: 0.9646 - val_loss: 0.8233 - val_accuracy: 0.7189
Epoch 94/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.1663 -
accuracy: 0.9719 - val_loss: 0.8058 - val_accuracy: 0.7239
Epoch 95/100
3304/3304 [==============================] - 32s 10ms/step - loss: 0.1619 -
accuracy: 0.9728 - val_loss: 0.7947 - val_accuracy: 0.7232
Epoch 96/100
3304/3304 [==============================] - 23s 7ms/step - loss: 0.1628 -
accuracy: 0.9685 - val_loss: 0.8205 - val_accuracy: 0.7203
Epoch 97/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.1548 -
accuracy: 0.9697 - val_loss: 0.8434 - val_accuracy: 0.7182
Epoch 98/100
3304/3304 [==============================] - 24s 7ms/step - loss: 0.1493 -
accuracy: 0.9743 - val_loss: 0.8136 - val_accuracy: 0.7210
Epoch 99/100
3304/3304 [==============================] - 29s 9ms/step - loss: 0.1466 -
accuracy: 0.9731 - val_loss: 0.8196 - val_accuracy: 0.7274
Epoch 100/100
3304/3304 [==============================] - 34s 10ms/step - loss: 0.1433 -
accuracy: 0.9752 - val_loss: 0.8090 - val_accuracy: 0.7210
```

[0]:

```
plt.plot(model_1_combined_history.history['accuracy'])
plt.plot(model_1_combined_history.history['val_accuracy'])
plt.title('Model 1 Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
plt.plot(model_1_combined_history.history['loss'])
plt.plot(model_1_combined_history.history['val_loss'])
plt.title('Model 1 Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```


Model 1 Accuracy
Train
Test
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100


Model 1 Loss
Train
Test
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100

[0]:
```python
# Save model and weights
model_name = 'Model_1_combined.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model_1_combined.save(model_path)
print('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model_1_combined.to_json()
with open("model_1_combined_json.json", "w") as json_file:
    json_file.write(model_json)
```

```
Save model and weights at /home/subodh/Second Semester/ML/Random
Project/Audio/saved_models/Model_1_combined.h5
```

[0]:
```python
model_2_combined = build_model_2()
model_2_combined_history=model_2_combined.fit(X_train, y_train, batch_size=16,␣
↪epochs=100, validation_data=(X_test, y_test))
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_43 (Conv1D)           (None, 216, 128)          768
_________________________________________________________________
activation_56 (Activation)   (None, 216, 128)          0
_________________________________________________________________
conv1d_44 (Conv1D)           (None, 216, 128)          82048
_________________________________________________________________
activation_57 (Activation)   (None, 216, 128)          0
_________________________________________________________________
dropout_18 (Dropout)         (None, 216, 128)          0
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, 27, 128)           0
_________________________________________________________________
conv1d_45 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_58 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_46 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_59 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_47 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
```

```
activation_60 (Activation)   (None, 27, 128)          0
_________________________________________________________________
dropout_19 (Dropout)         (None, 27, 128)          0
_________________________________________________________________
conv1d_48 (Conv1D)           (None, 27, 128)          82048
_________________________________________________________________
activation_61 (Activation)   (None, 27, 128)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 3456)             0
_________________________________________________________________
dense_14 (Dense)             (None, 14)               48398
_________________________________________________________________
activation_62 (Activation)   (None, 14)               0
=================================================================
Total params: 459,406
Trainable params: 459,406
Non-trainable params: 0
_________________________________________________________________
Train on 3304 samples, validate on 1416 samples
Epoch 1/100
3304/3304 [==============================] - 6s 2ms/step - loss: 2.6013 -
accuracy: 0.1574 - val_loss: 2.5537 - val_accuracy: 0.1702
Epoch 2/100
3304/3304 [==============================] - 5s 2ms/step - loss: 2.4491 -
accuracy: 0.1855 - val_loss: 2.3839 - val_accuracy: 0.2126
Epoch 3/100
3304/3304 [==============================] - 5s 2ms/step - loss: 2.2564 -
accuracy: 0.2558 - val_loss: 2.1772 - val_accuracy: 0.2903
Epoch 4/100
3304/3304 [==============================] - 8s 2ms/step - loss: 2.0294 -
accuracy: 0.3002 - val_loss: 1.9494 - val_accuracy: 0.3376
Epoch 5/100
3304/3304 [==============================] - 6s 2ms/step - loss: 1.8626 -
accuracy: 0.3232 - val_loss: 1.8114 - val_accuracy: 0.3588
Epoch 6/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.7683 -
accuracy: 0.3462 - val_loss: 1.7461 - val_accuracy: 0.3856
Epoch 7/100
3304/3304 [==============================] - 6s 2ms/step - loss: 1.7125 -
accuracy: 0.3717 - val_loss: 1.7033 - val_accuracy: 0.3941
Epoch 8/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.6868 -
accuracy: 0.3768 - val_loss: 1.6771 - val_accuracy: 0.4004
Epoch 9/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.6524 -
accuracy: 0.4022 - val_loss: 1.6549 - val_accuracy: 0.4209
Epoch 10/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.6284 -
```

```
accuracy: 0.4010 - val_loss: 1.6316 - val_accuracy: 0.4202
Epoch 11/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.6014 -
accuracy: 0.4222 - val_loss: 1.6061 - val_accuracy: 0.4449
Epoch 12/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.5866 -
accuracy: 0.4334 - val_loss: 1.5937 - val_accuracy: 0.4513
Epoch 13/100
3304/3304 [==============================] - 8s 3ms/step - loss: 1.5598 -
accuracy: 0.4428 - val_loss: 1.5716 - val_accuracy: 0.4541
Epoch 14/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.5431 -
accuracy: 0.4434 - val_loss: 1.5577 - val_accuracy: 0.4654
Epoch 15/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.5158 -
accuracy: 0.4540 - val_loss: 1.5394 - val_accuracy: 0.4689
Epoch 16/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.5065 -
accuracy: 0.4613 - val_loss: 1.5266 - val_accuracy: 0.4767
Epoch 17/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.4997 -
accuracy: 0.4558 - val_loss: 1.5186 - val_accuracy: 0.4838
Epoch 18/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.4691 -
accuracy: 0.4803 - val_loss: 1.4991 - val_accuracy: 0.5014
Epoch 19/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.4646 -
accuracy: 0.4840 - val_loss: 1.4898 - val_accuracy: 0.4873
Epoch 20/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.4486 -
accuracy: 0.4918 - val_loss: 1.4773 - val_accuracy: 0.4901
Epoch 21/100
3304/3304 [==============================] - 6s 2ms/step - loss: 1.4308 -
accuracy: 0.4873 - val_loss: 1.4709 - val_accuracy: 0.4979
Epoch 22/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.4204 -
accuracy: 0.5012 - val_loss: 1.4522 - val_accuracy: 0.5092
Epoch 23/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.4075 -
accuracy: 0.4967 - val_loss: 1.4411 - val_accuracy: 0.5205
Epoch 24/100
3304/3304 [==============================] - 6s 2ms/step - loss: 1.4031 -
accuracy: 0.5021 - val_loss: 1.4293 - val_accuracy: 0.5240
Epoch 25/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.3878 -
accuracy: 0.5203 - val_loss: 1.4217 - val_accuracy: 0.5219
Epoch 26/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.3809 -
```

```
accuracy: 0.5163 - val_loss: 1.4141 - val_accuracy: 0.5212
Epoch 27/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.3640 -
accuracy: 0.5236 - val_loss: 1.4023 - val_accuracy: 0.5240
Epoch 28/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.3481 -
accuracy: 0.5272 - val_loss: 1.3972 - val_accuracy: 0.5290
Epoch 29/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.3400 -
accuracy: 0.5288 - val_loss: 1.3892 - val_accuracy: 0.5311
Epoch 30/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.3248 -
accuracy: 0.5406 - val_loss: 1.3819 - val_accuracy: 0.5374
Epoch 31/100
3304/3304 [==============================] - 6s 2ms/step - loss: 1.3182 -
accuracy: 0.5475 - val_loss: 1.3744 - val_accuracy: 0.5325
Epoch 32/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.3096 -
accuracy: 0.5484 - val_loss: 1.3696 - val_accuracy: 0.5353
Epoch 33/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2975 -
accuracy: 0.5566 - val_loss: 1.3643 - val_accuracy: 0.5367
Epoch 34/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2929 -
accuracy: 0.5466 - val_loss: 1.3490 - val_accuracy: 0.5523
Epoch 35/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.2779 -
accuracy: 0.5623 - val_loss: 1.3463 - val_accuracy: 0.5537
Epoch 36/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.2779 -
accuracy: 0.5648 - val_loss: 1.3409 - val_accuracy: 0.5487
Epoch 37/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2623 -
accuracy: 0.5693 - val_loss: 1.3360 - val_accuracy: 0.5516
Epoch 38/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2637 -
accuracy: 0.5645 - val_loss: 1.3325 - val_accuracy: 0.5530
Epoch 39/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.2551 -
accuracy: 0.5726 - val_loss: 1.3177 - val_accuracy: 0.5600
Epoch 40/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.2514 -
accuracy: 0.5639 - val_loss: 1.3140 - val_accuracy: 0.5621
Epoch 41/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2348 -
accuracy: 0.5732 - val_loss: 1.3073 - val_accuracy: 0.5607
Epoch 42/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2281 -
```

```
accuracy: 0.5772 - val_loss: 1.3039 - val_accuracy: 0.5636
Epoch 43/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.2343 -
accuracy: 0.5775 - val_loss: 1.2993 - val_accuracy: 0.5671
Epoch 44/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.2222 -
accuracy: 0.5760 - val_loss: 1.2959 - val_accuracy: 0.5657
Epoch 45/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2117 -
accuracy: 0.5872 - val_loss: 1.2884 - val_accuracy: 0.5650
Epoch 46/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.2113 -
accuracy: 0.5869 - val_loss: 1.2859 - val_accuracy: 0.5727
Epoch 47/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.2002 -
accuracy: 0.5947 - val_loss: 1.2822 - val_accuracy: 0.5742
Epoch 48/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.1872 -
accuracy: 0.5972 - val_loss: 1.2739 - val_accuracy: 0.5699
Epoch 49/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.1860 -
accuracy: 0.5908 - val_loss: 1.2635 - val_accuracy: 0.5756
Epoch 50/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1851 -
accuracy: 0.6011 - val_loss: 1.2747 - val_accuracy: 0.5685
Epoch 51/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1691 -
accuracy: 0.6026 - val_loss: 1.2594 - val_accuracy: 0.5784
Epoch 52/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1662 -
accuracy: 0.5993 - val_loss: 1.2536 - val_accuracy: 0.5777
Epoch 53/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.1550 -
accuracy: 0.5999 - val_loss: 1.2467 - val_accuracy: 0.5833
Epoch 54/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.1527 -
accuracy: 0.6035 - val_loss: 1.2457 - val_accuracy: 0.5791
Epoch 55/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1473 -
accuracy: 0.6008 - val_loss: 1.2386 - val_accuracy: 0.5876
Epoch 56/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1385 -
accuracy: 0.6011 - val_loss: 1.2312 - val_accuracy: 0.5862
Epoch 57/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.1313 -
accuracy: 0.6068 - val_loss: 1.2366 - val_accuracy: 0.5812
Epoch 58/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.1156 -
```

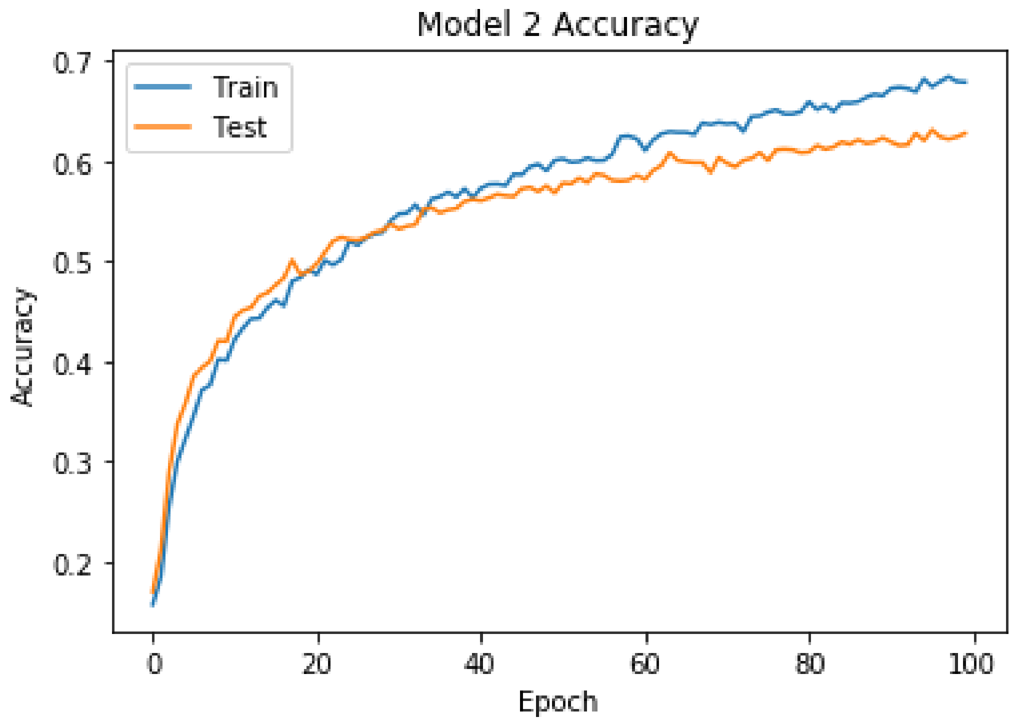
```
accuracy: 0.6244 - val_loss: 1.2342 - val_accuracy: 0.5805
Epoch 59/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.1131 -
accuracy: 0.6253 - val_loss: 1.2297 - val_accuracy: 0.5812
Epoch 60/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.1116 -
accuracy: 0.6226 - val_loss: 1.2238 - val_accuracy: 0.5862
Epoch 61/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.1183 -
accuracy: 0.6111 - val_loss: 1.2215 - val_accuracy: 0.5819
Epoch 62/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0998 -
accuracy: 0.6214 - val_loss: 1.2131 - val_accuracy: 0.5911
Epoch 63/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.0950 -
accuracy: 0.6274 - val_loss: 1.2081 - val_accuracy: 0.5960
Epoch 64/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0856 -
accuracy: 0.6295 - val_loss: 1.1974 - val_accuracy: 0.6088
Epoch 65/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0852 -
accuracy: 0.6289 - val_loss: 1.2015 - val_accuracy: 0.6010
Epoch 66/100
3304/3304 [==============================] - 10s 3ms/step - loss: 1.0811 -
accuracy: 0.6289 - val_loss: 1.1975 - val_accuracy: 0.5996
Epoch 67/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0812 -
accuracy: 0.6268 - val_loss: 1.1957 - val_accuracy: 0.5989
Epoch 68/100
3304/3304 [==============================] - 13s 4ms/step - loss: 1.0667 -
accuracy: 0.6383 - val_loss: 1.1897 - val_accuracy: 0.5989
Epoch 69/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0629 -
accuracy: 0.6368 - val_loss: 1.1920 - val_accuracy: 0.5890
Epoch 70/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.0560 -
accuracy: 0.6389 - val_loss: 1.1794 - val_accuracy: 0.6038
Epoch 71/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0530 -
accuracy: 0.6374 - val_loss: 1.1827 - val_accuracy: 0.5982
Epoch 72/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0495 -
accuracy: 0.6383 - val_loss: 1.1794 - val_accuracy: 0.5946
Epoch 73/100
3304/3304 [==============================] - 8s 3ms/step - loss: 1.0485 -
accuracy: 0.6298 - val_loss: 1.1752 - val_accuracy: 0.6010
Epoch 74/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0385 -
```

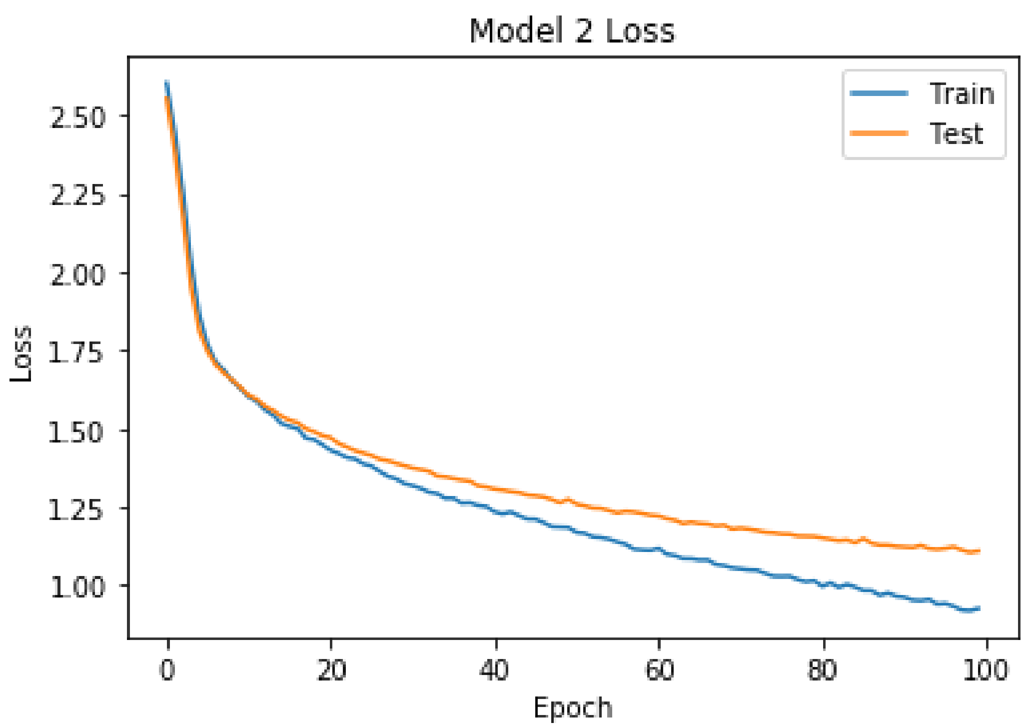
```
accuracy: 0.6447 - val_loss: 1.1695 - val_accuracy: 0.6031
Epoch 75/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.0294 -
accuracy: 0.6453 - val_loss: 1.1676 - val_accuracy: 0.6088
Epoch 76/100
3304/3304 [==============================] - 7s 2ms/step - loss: 1.0288 -
accuracy: 0.6492 - val_loss: 1.1641 - val_accuracy: 0.6010
Epoch 77/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.0294 -
accuracy: 0.6513 - val_loss: 1.1633 - val_accuracy: 0.6116
Epoch 78/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.0184 -
accuracy: 0.6474 - val_loss: 1.1574 - val_accuracy: 0.6123
Epoch 79/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.0117 -
accuracy: 0.6474 - val_loss: 1.1568 - val_accuracy: 0.6116
Epoch 80/100
3304/3304 [==============================] - 5s 2ms/step - loss: 1.0152 -
accuracy: 0.6495 - val_loss: 1.1561 - val_accuracy: 0.6088
Epoch 81/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9968 -
accuracy: 0.6595 - val_loss: 1.1514 - val_accuracy: 0.6095
Epoch 82/100
3304/3304 [==============================] - 9s 3ms/step - loss: 1.0084 -
accuracy: 0.6516 - val_loss: 1.1471 - val_accuracy: 0.6158
Epoch 83/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9933 -
accuracy: 0.6562 - val_loss: 1.1423 - val_accuracy: 0.6123
Epoch 84/100
3304/3304 [==============================] - 8s 2ms/step - loss: 1.0033 -
accuracy: 0.6498 - val_loss: 1.1451 - val_accuracy: 0.6144
Epoch 85/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9945 -
accuracy: 0.6586 - val_loss: 1.1356 - val_accuracy: 0.6194
Epoch 86/100
3304/3304 [==============================] - 7s 2ms/step - loss: 0.9836 -
accuracy: 0.6583 - val_loss: 1.1496 - val_accuracy: 0.6172
Epoch 87/100
3304/3304 [==============================] - 7s 2ms/step - loss: 0.9834 -
accuracy: 0.6595 - val_loss: 1.1342 - val_accuracy: 0.6215
Epoch 88/100
3304/3304 [==============================] - 9s 3ms/step - loss: 0.9682 -
accuracy: 0.6640 - val_loss: 1.1286 - val_accuracy: 0.6179
Epoch 89/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9759 -
accuracy: 0.6671 - val_loss: 1.1283 - val_accuracy: 0.6194
Epoch 90/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9654 -
```

```
accuracy: 0.6656 - val_loss: 1.1242 - val_accuracy: 0.6236
Epoch 91/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9622 -
accuracy: 0.6728 - val_loss: 1.1225 - val_accuracy: 0.6194
Epoch 92/100
3304/3304 [==============================] - 7s 2ms/step - loss: 0.9531 -
accuracy: 0.6740 - val_loss: 1.1208 - val_accuracy: 0.6158
Epoch 93/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9517 -
accuracy: 0.6728 - val_loss: 1.1274 - val_accuracy: 0.6165
Epoch 94/100
3304/3304 [==============================] - 7s 2ms/step - loss: 0.9553 -
accuracy: 0.6695 - val_loss: 1.1175 - val_accuracy: 0.6278
Epoch 95/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9399 -
accuracy: 0.6822 - val_loss: 1.1149 - val_accuracy: 0.6208
Epoch 96/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9420 -
accuracy: 0.6746 - val_loss: 1.1180 - val_accuracy: 0.6314
Epoch 97/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9331 -
accuracy: 0.6798 - val_loss: 1.1228 - val_accuracy: 0.6243
Epoch 98/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9213 -
accuracy: 0.6846 - val_loss: 1.1125 - val_accuracy: 0.6222
Epoch 99/100
3304/3304 [==============================] - 8s 2ms/step - loss: 0.9194 -
accuracy: 0.6798 - val_loss: 1.1053 - val_accuracy: 0.6243
Epoch 100/100
3304/3304 [==============================] - 7s 2ms/step - loss: 0.9264 -
accuracy: 0.6792 - val_loss: 1.1096 - val_accuracy: 0.6278
```

[0]:

```
plt.plot(model_2_combined_history.history['accuracy'])
plt.plot(model_2_combined_history.history['val_accuracy'])
plt.title('Model 2 Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
plt.plot(model_2_combined_history.history['loss'])
plt.plot(model_2_combined_history.history['val_loss'])
plt.title('Model 2 Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```


Model 2 Accuracy
Train
Test
Accuracy
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
Epoch


Model 2 Loss
Train
Test
Loss
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0
20
40
60
80
100
Epoch

[0]:
```
# Save model and weights
model_name = 'Model_2_combined.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model_2_combined.save(model_path)
print('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model_2_combined.to_json()
with open("model_2_combined_json.json", "w") as json_file:
    json_file.write(model_json)
```

```
Save model and weights at /home/subodh/Second Semester/ML/Random
Project/Audio/saved_models/Model_2_combined.h5
```

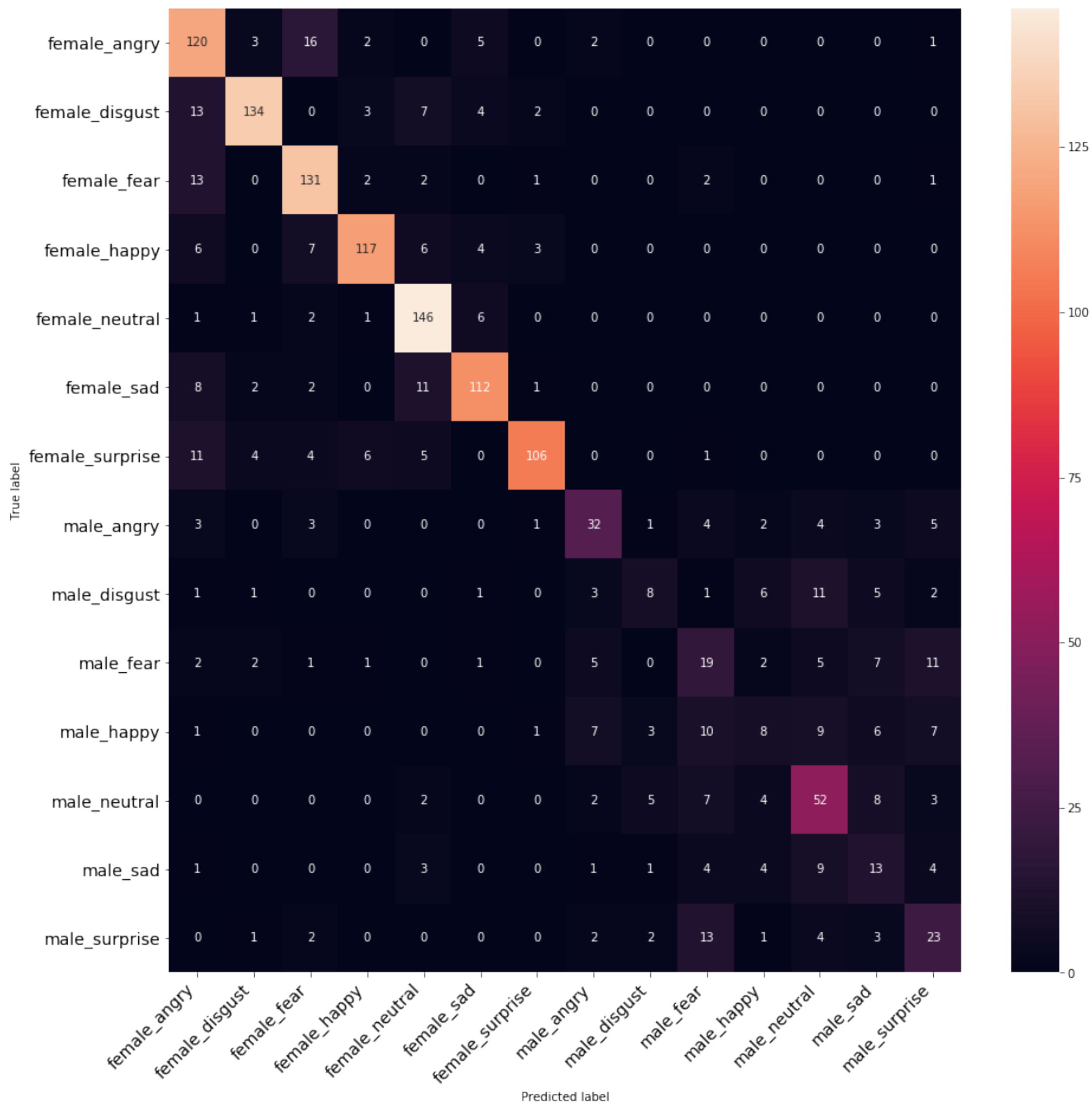
4.2 Now that we have saved the models, we can see how they perform on the test data

[0]:
```
def load_and_print_results(filename, json_filename):
    # loading json and model architecture
    json_file = open(json_filename, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights("saved_models/" + filename)
    print("Loaded model from disk")

    # Keras optimiser
    opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
    loaded_model.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics=['accuracy'])
    score = loaded_model.evaluate(X_test, y_test, verbose=0)
    print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))


    #print(labels.classes_)
    preds = loaded_model.predict(X_test, batch_size = 16, verbose = 1)
    preds = preds.argmax(axis = 1)
    # predictions
    preds = preds.astype(int).flatten()
    preds = (labels.inverse_transform((preds)))
    preds = pd.DataFrame({'predictedvalues': preds})
```

```
    # Actual labels
    actual=y_test.argmax(axis=1)
    actual = actual.astype(int).flatten()
    actual = (labels.inverse_transform((actual)))
    actual = pd.DataFrame({'actualvalues': actual})

    # Lets combined both of them into a single dataframe
    com_finaldf = actual.join(preds)
    print(com_finaldf[170:180])
    com_finaldf.to_csv('Predictions_combined.csv', index = False)
    com_finaldf.groupby('predictedvalues').count()


    com_finaldf = pd.read_csv('Predictions_combined.csv')
    classes = com_finaldf.actualvalues.unique()
    classes.sort()

    c = confusion_matrix(com_finaldf.actualvalues, com_finaldf.predictedvalues)
    print(accuracy_score(com_finaldf.actualvalues, com_finaldf.predictedvalues))
    print_confusion_matrix(c, class_names = classes)

    classes = com_finaldf.actualvalues.unique()
    classes.sort()
    print(classification_report(com_finaldf.actualvalues, com_finaldf.
↪predictedvalues, target_names=classes))
    return com_finaldf

com_finaldf = load_and_print_results('Model_1_combined.h5',␣
↪'model_1_combined_json.json')
```

```
Loaded model from disk
accuracy: 72.10%
1416/1416 [==============================] - 2s 1ms/step
        actualvalues  predictedvalues
170     female_angry   female_disgust
171   female_neutral   female_neutral
172     male_neutral     male_neutral
173     male_neutral     male_neutral
174     female_happy     female_happy
175     male_neutral     male_neutral
176      female_fear      female_fear
177     female_angry     female_angry
178     male_disgust     male_neutral
179  female_surprise  female_surprise
0.721045197740113
                 precision    recall  f1-score   support
```

female_angry	0.67	0.81	0.73	149
female_disgust	0.91	0.82	0.86	163
female_fear	0.78	0.86	0.82	152
female_happy	0.89	0.82	0.85	143
female_neutral	0.80	0.93	0.86	157
female_sad	0.84	0.82	0.83	136
female_surprise	0.92	0.77	0.84	137
male_angry	0.59	0.55	0.57	58
male_disgust	0.40	0.21	0.27	39
male_fear	0.31	0.34	0.32	56
male_happy	0.30	0.15	0.20	52
male_neutral	0.55	0.63	0.59	83
male_sad	0.29	0.33	0.31	40
male_surprise	0.40	0.45	0.43	51
accuracy			0.72	1416
macro avg	0.62	0.61	0.61	1416
weighted avg	0.72	0.72	0.72	1416



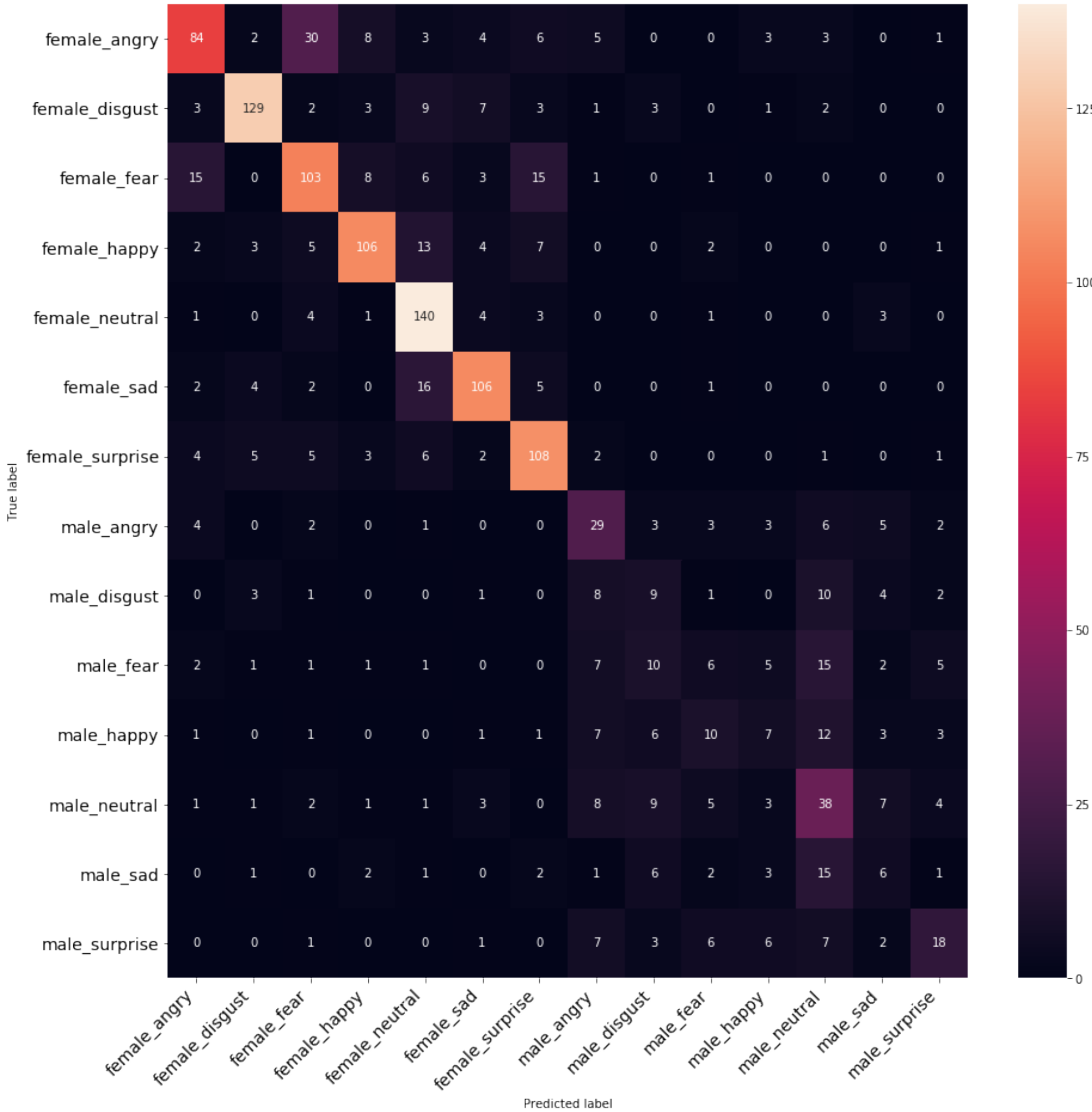

```
[0]: load_and_print_results('Model_2_combined.h5', 'model_2_combined_json.json')
```

```
Loaded model from disk
accuracy: 62.78%
1416/1416 [==============================] - 1s 363us/step
        actualvalues  predictedvalues
170     female_angry       female_sad
171   female_neutral   female_neutral
172     male_neutral     male_neutral
173     male_neutral     male_neutral
174     female_happy     female_happy
175     male_neutral     male_neutral
176      female_fear      female_fear
```

```
177     female_angry     female_angry
178     male_disgust        male_fear
179  female_surprise  female_surprise
0.6278248587570622
                 precision    recall  f1-score   support

   female_angry       0.71      0.56      0.63       149
 female_disgust       0.87      0.79      0.83       163
    female_fear       0.65      0.68      0.66       152
   female_happy       0.80      0.74      0.77       143
 female_neutral       0.71      0.89      0.79       157
     female_sad       0.78      0.78      0.78       136
female_surprise       0.72      0.79      0.75       137
     male_angry       0.38      0.50      0.43        58
   male_disgust       0.18      0.23      0.20        39
      male_fear       0.16      0.11      0.13        56
     male_happy       0.23      0.13      0.17        52
   male_neutral       0.35      0.46      0.40        83
       male_sad       0.19      0.15      0.17        40
  male_surprise       0.47      0.35      0.40        51

       accuracy                           0.63      1416
      macro avg       0.51      0.51      0.51      1416
   weighted avg       0.63      0.63      0.62      1416
```

[0]:

```
         actualvalues  predictedvalues
0     female_surprise  female_surprise
1        female_angry       male_angry
2        male_neutral     male_neutral
3           male_fear    male_surprise
4      female_disgust   female_disgust
…                 …                …
1411       male_happy     male_neutral
1412     female_happy     female_happy
1413   female_neutral   female_neutral
1414        male_fear     male_neutral
1415  female_surprise  female_surprise

[1416 rows x 2 columns]
```



[0]:

```
```

4.3 Now, let's group the gender and check for the results for the combined data

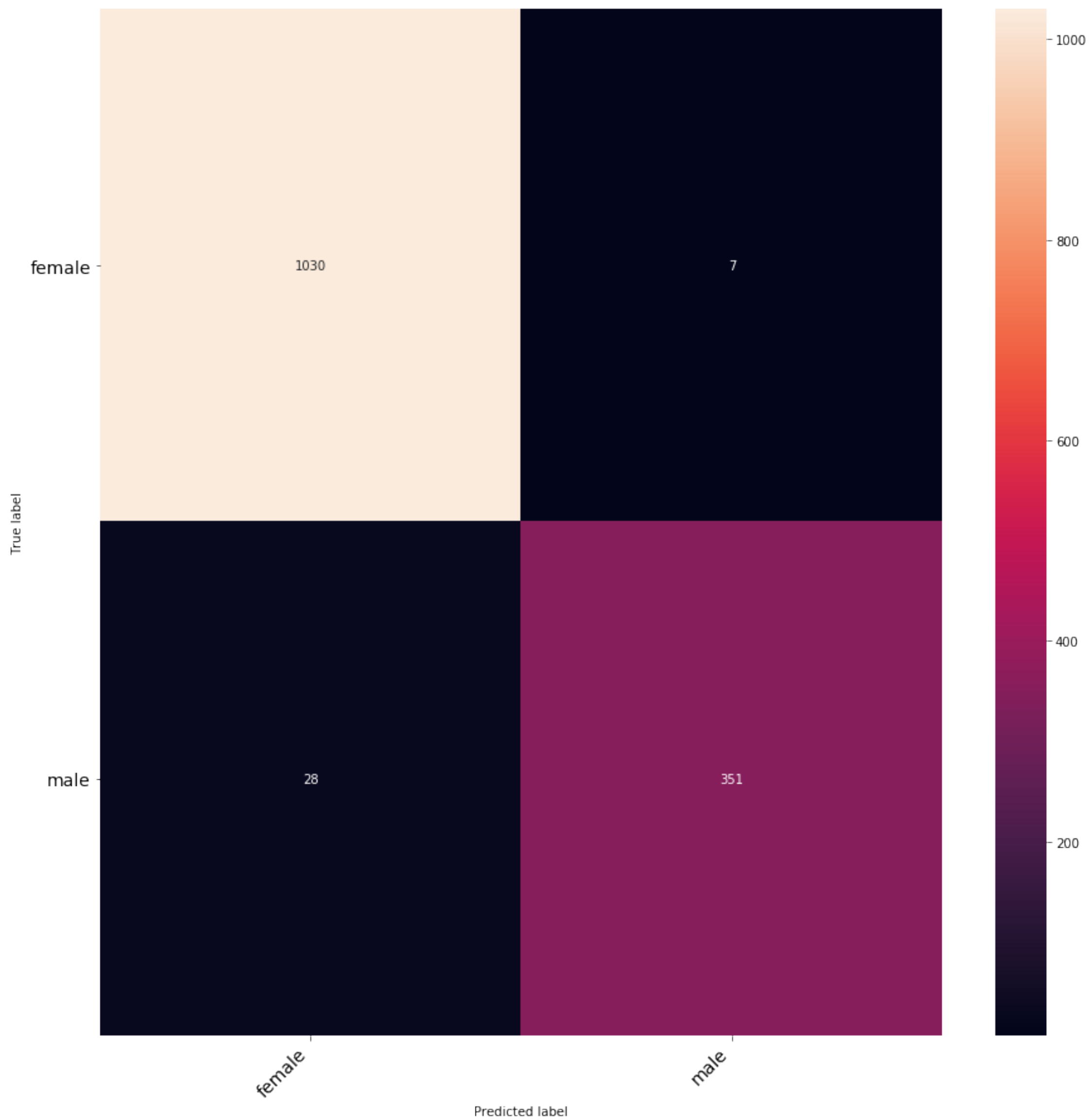
[0]:

```
modidf = com_finaldf
modidf['actualvalues'] = com_finaldf.actualvalues.replace({'female_angry':
↪'female'
                                          , 'female_disgust':'female'
                                          , 'female_fear':'female'
                                          , 'female_happy':'female'
                                          , 'female_sad':'female'
                                          , 'female_surprise':'female'
```

```
                                      , 'female_neutral':'female'
                                      , 'male_angry':'male'
                                      , 'male_fear':'male'
                                      , 'male_happy':'male'
                                      , 'male_sad':'male'
                                      , 'male_surprise':'male'
                                      , 'male_neutral':'male'
                                      , 'male_disgust':'male'
                                     })

modidf['predictedvalues'] = com_finaldf.predictedvalues.replace({'female_angry':
 ↪'female'
                                      , 'female_disgust':'female'
                                      , 'female_fear':'female'
                                      , 'female_happy':'female'
                                      , 'female_sad':'female'
                                      , 'female_surprise':'female'
                                      , 'female_neutral':'female'
                                      , 'male_angry':'male'
                                      , 'male_fear':'male'
                                      , 'male_happy':'male'
                                      , 'male_sad':'male'
                                      , 'male_surprise':'male'
                                      , 'male_neutral':'male'
                                      , 'male_disgust':'male'
                                     })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
0.9752824858757062
              precision    recall  f1-score   support

      female       0.97      0.99      0.98      1037
        male       0.98      0.93      0.95       379

    accuracy                           0.98      1416
   macro avg       0.98      0.96      0.97      1416
```

```
weighted avg       0.98      0.98      0.98      1416
```



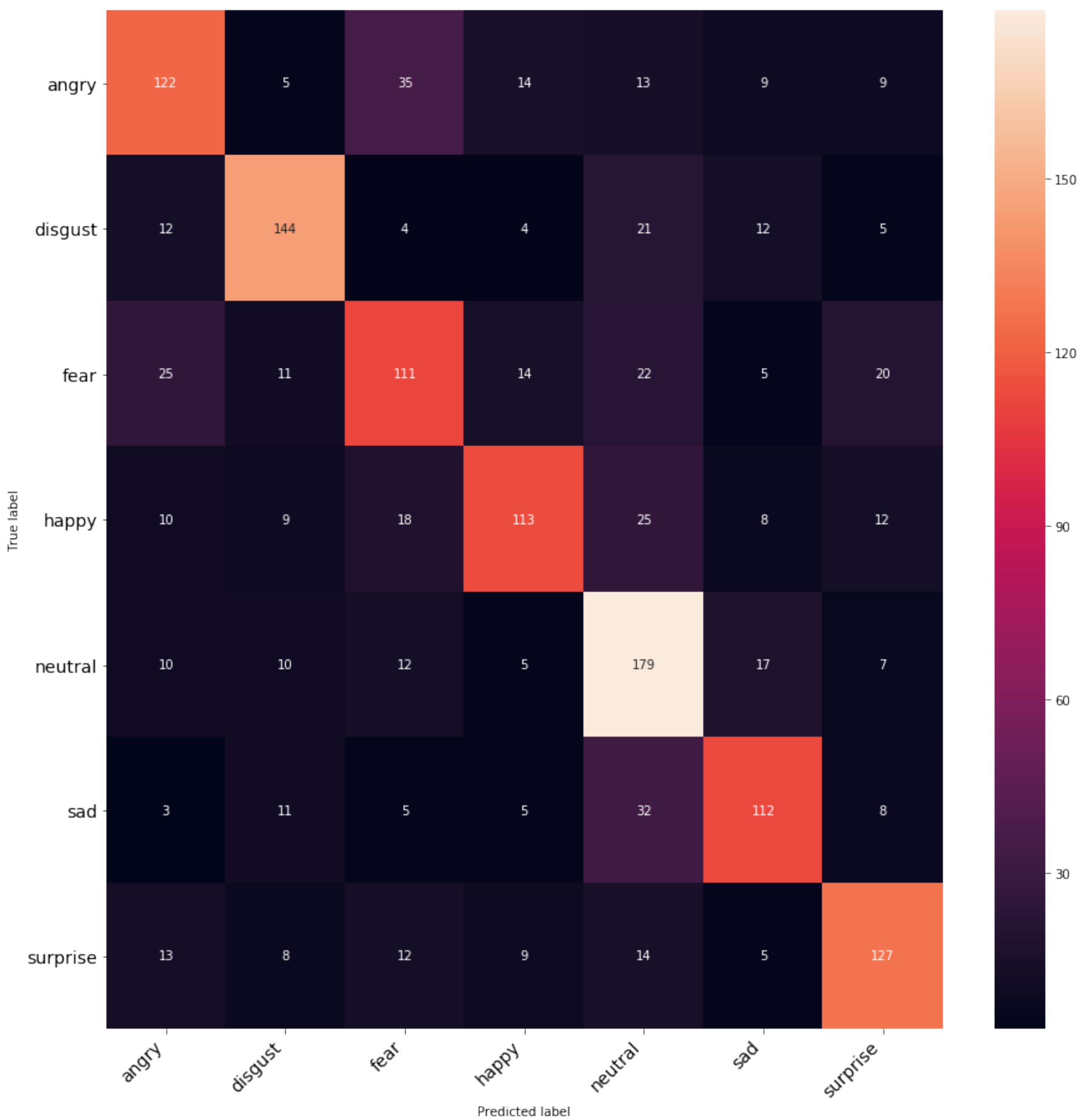
4.4 Let's group together the emotions and look for the performance for the combined data

[0]:
```
modidf = pd.read_csv("Predictions_combined.csv")
modidf['actualvalues'] = modidf.actualvalues.replace({'female_angry':'angry'
                                       , 'female_disgust':'disgust'
                                       , 'female_fear':'fear'
                                       , 'female_happy':'happy'
                                       , 'female_sad':'sad'
```

```
                                   , 'female_surprise':'surprise'
                                   , 'female_neutral':'neutral'
                                   , 'male_angry':'angry'
                                   , 'male_fear':'fear'
                                   , 'male_happy':'happy'
                                   , 'male_sad':'sad'
                                   , 'male_surprise':'surprise'
                                   , 'male_neutral':'neutral'
                                   , 'male_disgust':'disgust'
                                  })

modidf['predictedvalues'] = modidf.predictedvalues.replace({'female_angry':
 ↪'angry'
                                   , 'female_disgust':'disgust'
                                   , 'female_fear':'fear'
                                   , 'female_happy':'happy'
                                   , 'female_sad':'sad'
                                   , 'female_surprise':'surprise'
                                   , 'female_neutral':'neutral'
                                   , 'male_angry':'angry'
                                   , 'male_fear':'fear'
                                   , 'male_happy':'happy'
                                   , 'male_sad':'sad'
                                   , 'male_surprise':'surprise'
                                   , 'male_neutral':'neutral'
                                   , 'male_disgust':'disgust'
                                  })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)

# Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
0.6412429378531074
              precision    recall  f1-score   support

       angry       0.63      0.59      0.61       207
     disgust       0.73      0.71      0.72       202
```

fear	0.56	0.53	0.55	208
happy	0.69	0.58	0.63	195
neutral	0.58	0.75	0.66	240
sad	0.67	0.64	0.65	176
surprise	0.68	0.68	0.68	188
accuracy			0.64	1416
macro avg	0.65	0.64	0.64	1416
weighted avg	0.64	0.64	0.64	1416



4.4.1 Model 1 gave us accuracy on the combined data of 72.10%, where as Model 2 gave accuracy of 62.78%, so from here on we will only work with Model 1.

5 Experiment 4: Now we experiment with the combined data with some augmentation methods

Back to Experiments and Results

5.0.1 This is a major comparison experiment. We will try to compare the contributions done by augmentation methods in the accuracy

This experiment is heavily influenced by Edward Ma

5.1 A. Load the combined dataset

[0]:
```
from tqdm import tqdm
df = pd.read_csv('./Data_combined.csv')
df.head()
```

[0]:
```
       labels source                      path
0   male_fear  SAVEE   ./SAVEE_used/KL_f04.wav
1  male_angry  SAVEE   ./SAVEE_used/KL_a11.wav
2   male_fear  SAVEE   ./SAVEE_used/JE_f11.wav
3    male_sad  SAVEE  ./SAVEE_used/DC_sa11.wav
4   male_fear  SAVEE   ./SAVEE_used/KL_f08.wav
```

[0]:
```

```

5.2 B. Define Augmentation Methods

5.2.1 We define some augmentation methods

[0]:
```
def noise(data):
    """
    Adding White Noise.
    """
    noise_amp = 0.05*np.random.uniform()*np.amax(data)    # more noise reduce␣
↪the value to 0.5
    data = data.astype('float64') + noise_amp * np.random.normal(size=data.
↪shape[0])
    return data

def shift(data):
    """
    Random Shifting.
    """
    s_range = int(np.random.uniform(low=-5, high = 5)*1000)  #default at 500
    return np.roll(data, s_range)
```

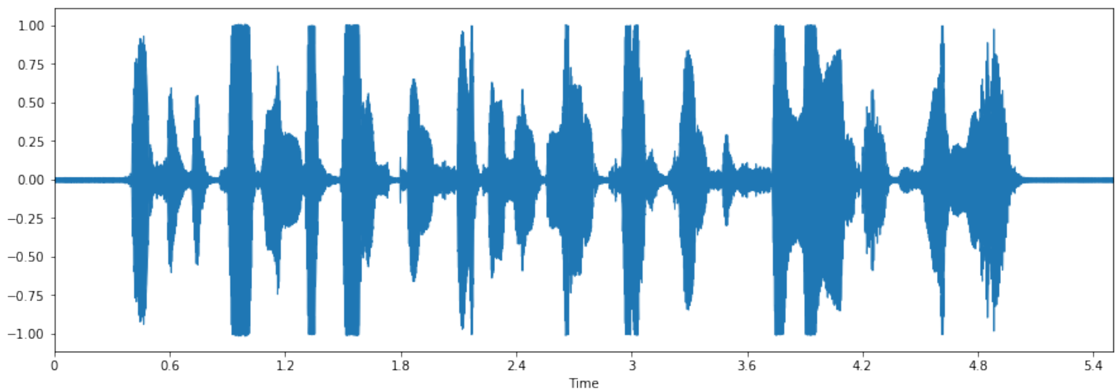
```python
def stretch(data, rate=0.8):
    """
    Streching the Sound. Note that this expands the dataset slightly
    """
    data = librosa.effects.time_stretch(data, rate)
    return data

def pitch(data, sample_rate):
    """
    Pitch Tuning.
    """
    bins_per_octave = 12
    pitch_pm = 2
    pitch_change =  pitch_pm * 2*(np.random.uniform())
    data = librosa.effects.pitch_shift(data.astype('float64'),
                                      sample_rate, n_steps=pitch_change,
                                      bins_per_octave=bins_per_octave)
    return data

def dyn_change(data):
    """
    Random Value Change.
    """
    dyn_change = np.random.uniform(low=-0.5 ,high=7)  # default low = 1.5, high␣
 ↪= 3
    return (data * dyn_change)

def speedNpitch(data):
    """
    peed and Pitch Tuning.
    """
    length_change = np.random.uniform(low=0.8, high = 1)
    speed_fac = 1.2  / length_change # try changing 1.0 to 2.0 ... =D
    tmp = np.interp(np.arange(0,len(data),speed_fac),np.
 ↪arange(0,len(data)),data)
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data
```

5.3 C. Explore the methods

5.3.1 Let's take some data and try these methods

[0]:
```
fname = './SAVEE_used/JK_f12.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)

# Play it again to refresh our memory
ipd.Audio(data, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>

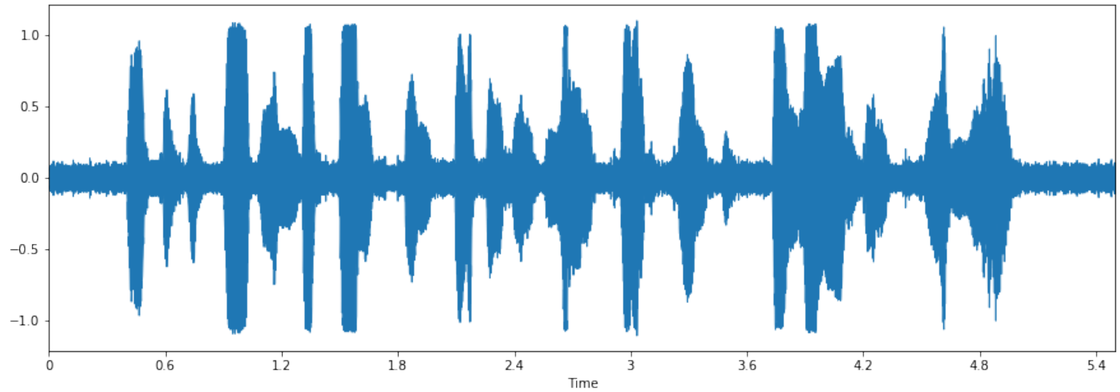


5.4 1. Static Noise

5.4.1 First we will add static noise in the background and see how it sounds

[0]:
```
x = noise(data)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(x, sr=sampling_rate)
ipd.Audio(x, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>

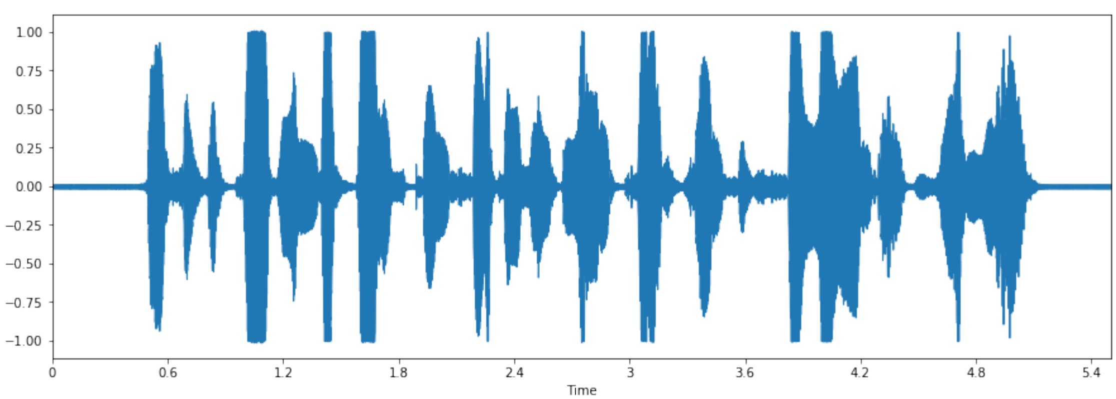


5.5 2. Shift

What we did here is we shifted the audio randomly either to the left or the right direction, within fixed audio duration. This is similar to the original plot except there's a tiny bit of delay before the speaker speaks

[0]:
```
x = shift(data)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(x, sr=sampling_rate)
ipd.Audio(x, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>



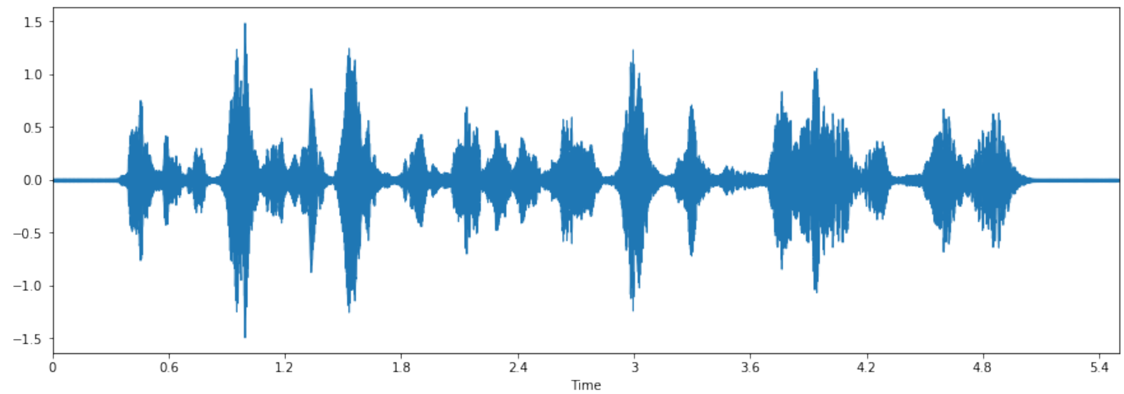
[0]:
```

```

5.6 3. Pitch

This method stretches the audio, because of which the duration is longer but the audio wave gets stretched too

[0]:
```
x = pitch(data, sampling_rate)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(x, sr=sampling_rate)
ipd.Audio(x, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>

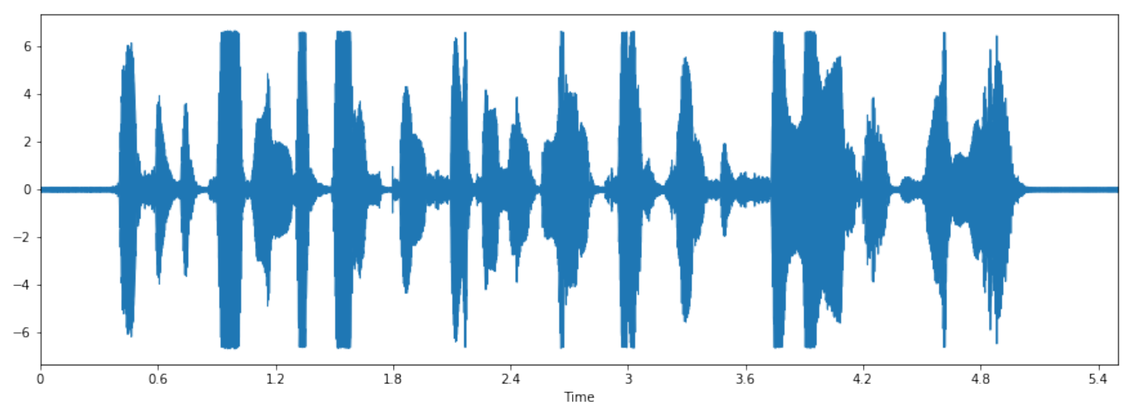


5.7 4. Dynamic Change

We try to do some dynamic changes in the audio file

[0]:
```
x = dyn_change(data)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(x, sr=sampling_rate)
ipd.Audio(x, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>

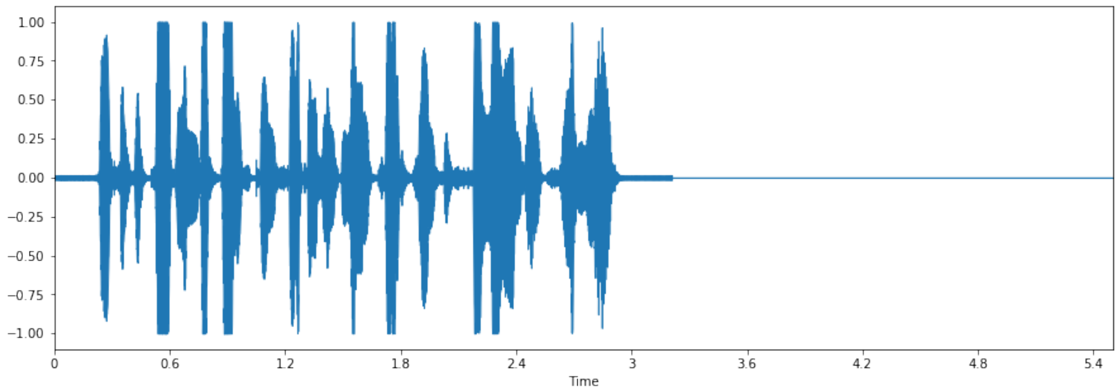


5.8 5. Speed and Pitch

As the name suggests we try to play with the speed and speech of the qudio file

[0]:
```
x = speedNpitch(data)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(x, sr=sampling_rate)
ipd.Audio(x, rate=sampling_rate)
```

[0]: <IPython.lib.display.Audio object>



5.9 D. Data Preprocessing

[0]:
```
data = pd.read_csv('./Data_combined.csv')
data.head()
```

[0]:

	labels	source	path
0	male_fear	SAVEE	./SAVEE_used/KL_f04.wav
1	male_angry	SAVEE	./SAVEE_used/KL_a11.wav
2	male_fear	SAVEE	./SAVEE_used/JE_f11.wav
3	male_sad	SAVEE	./SAVEE_used/DC_sa11.wav
4	male_fear	SAVEE	./SAVEE_used/KL_f08.wav

[0]:
```
data.shape
```

[0]: (4720, 3)

[0]:
```
df = pd.DataFrame(columns=['feature'])
df_noise = pd.DataFrame(columns=['feature'])
df_speedpitch = pd.DataFrame(columns=['feature'])
cnt = 0
# feature extraction
for i in tqdm(data.path):
    X, sample_rate = librosa.load(i, res_type='kaiser_fast'
                                  , duration=2.5
```

```
                                    , sr=44100
                                    , offset=0.5
                                    )

    mfccs = np.mean(librosa.feature.mfcc(y=X,
                                        sr=np.array(sample_rate),
                                        n_mfcc=13),
                                        axis=0)
    df.loc[cnt] = [mfccs]
    aug = noise(X)
    aug = np.mean(librosa.feature.mfcc(y=aug,
                                    sr=np.array(sample_rate),
                                    n_mfcc=13),
                                    axis=0)
    df_noise.loc[cnt] = [aug]

    # speed pitch
    aug = speedNpitch(X)
    aug = np.mean(librosa.feature.mfcc(y=aug,
                                    sr=np.array(sample_rate),
                                    n_mfcc=13),
                    axis=0)
    df_speedpitch.loc[cnt] = [aug]

    cnt += 1

df.head()
```

```
100%|      | 4720/4720 [05:25<00:00, 14.50it/s]
```

[0]:

	feature
0	[-30.205614, -28.294016, -27.90939, -28.50983,…
1	[-40.3034, -37.9193, -36.6451, -28.010498, -24…
2	[-21.392101, -21.662266, -22.259338, -24.44498…
3	[-24.90076, -24.354008, -23.842062, -23.96136,…
4	[-35.30234, -35.13114, -36.651817, -40.39294, …

[0]:
```
# combine
df = pd.concat([data,pd.DataFrame(df['feature'].values.tolist())],axis=1)
df_noise = pd.concat([data,pd.DataFrame(df_noise['feature'].values.
 ↪tolist())],axis=1)
df_speedpitch = pd.concat([data,pd.DataFrame(df_speedpitch['feature'].values.
 ↪tolist())],axis=1)
print(df.shape,df_noise.shape,df_speedpitch.shape)
```

```
(4720, 219) (4720, 219) (4720, 219)
```

[0]:
```
df = pd.concat([df,df_noise,df_speedpitch],axis=0,sort=False)
df=df.fillna(0)
del df_noise, df_speedpitch

df.head()
```

[0]:
```
       labels source                      path          0          1  \
0   male_fear  SAVEE   ./SAVEE_used/KL_f04.wav -30.205614 -28.294016
1  male_angry  SAVEE   ./SAVEE_used/KL_a11.wav -40.303398 -37.919300
2   male_fear  SAVEE   ./SAVEE_used/JE_f11.wav -21.392101 -21.662266
3    male_sad  SAVEE  ./SAVEE_used/DC_sa11.wav -24.900761 -24.354008
4   male_fear  SAVEE   ./SAVEE_used/KL_f08.wav -35.302341 -35.131142

           2          3          4          5          6  …        206  \
0 -27.909389 -28.509830 -28.120195 -28.570707 -29.546034  …   0.000000
1 -36.645100 -28.010498 -24.288029 -22.791922 -23.459490  …   0.000000
2 -22.259338 -24.444984 -23.050682 -23.140684 -22.903954  … -22.763048
3 -23.842062 -23.961361 -21.653095 -21.758453 -23.224234  … -25.985544
4 -36.651817 -40.392941 -40.899864 -39.890297 -37.871014  … -38.050453

         207        208        209        210        211        212  \
0   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
2 -21.701115 -11.972590  -9.322908 -10.145143 -12.772147 -14.352438
3 -19.891569 -15.183666 -13.054301 -12.797897 -11.673220  -9.116754
4 -36.393749 -36.336716 -37.973526 -32.837669 -29.188053 -29.468193

         213        214        215
0   0.000000   0.000000   0.000000
1   0.000000   0.000000   0.000000
2 -16.328117 -12.141912  -6.584519
3  -8.331745  -8.620239  -7.165553
4 -31.124041 -26.687956 -22.051907

[5 rows x 219 columns]
```

[0]:
```
X_train, X_test, y_train, y_test = train_test_split(df.
 ↪drop(['path','labels','source'],axis=1)
                                                    , df.labels
                                                    , test_size=0.25
                                                    , shuffle=True
                                                    , random_state=42
                                                   )

# Lets see how the data present itself before normalisation
X_train[150:160]
```

[0]:

	0	1	2	3	4	5
3153	-19.676512	-23.396910	-34.020939	-33.329849	-29.642826	-29.498608
1406	-46.517067	-46.517067	-46.517067	-46.517067	-46.517067	-46.517067
3863	-31.171440	-32.247330	-33.870255	-33.905083	-33.786892	-35.060898
3076	-23.684439	-25.834816	-36.033062	-37.161190	-35.740543	-34.186390
1387	-60.710445	-60.710445	-60.710445	-60.710445	-60.710445	-60.710445
1539	-42.335976	-39.704937	-39.089779	-38.157162	-38.366306	-38.743790
852	-41.479099	-40.854713	-40.902458	-40.164665	-39.799412	-39.881203
3164	-19.752104	-21.232588	-24.323814	-23.761780	-23.164494	-23.660288
1909	-58.269451	-58.269451	-58.269451	-58.269451	-58.269451	-58.269451
3948	0.516810	-2.895117	-9.617130	-10.407608	-11.494343	-11.967142

	6	7	8	9	…	206	207
3153	-29.849100	-29.572702	-30.188053	-31.178677	…	0.000000	0.000000
1406	-46.517067	-46.517067	-46.517067	-46.517067	…	-31.490894	-31.275990
3863	-34.166695	-34.314499	-36.433384	-36.262596	…	0.000000	0.000000
3076	-35.416485	-37.115284	-38.435276	-36.491867	…	0.000000	0.000000
1387	-60.710445	-60.710445	-60.710445	-60.710445	…	-60.710445	-60.710445
1539	-39.434391	-38.587292	-39.972046	-42.647263	…	-40.005379	-40.241177
852	-41.160500	-40.562630	-38.974007	-38.688774	…	-25.137627	-25.724409
3164	-23.135611	-23.077129	-23.061636	-23.905149	…	0.000000	0.000000
1909	-58.269451	-58.269451	-58.542385	-57.179745	…	-58.269451	-58.269451
3948	-11.295981	-11.034580	-13.457041	-11.022572	…	0.000000	0.000000

	208	209	210	211	212	213
3153	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1406	-31.752699	-33.640179	-32.239552	-31.140381	-31.129709	-34.016731
3863	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3076	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1387	-60.710445	-60.710445	-60.710445	-60.710445	-60.710445	-60.710445
1539	-40.178467	-36.949825	-36.158806	-38.596119	-40.459488	-37.236595
852	-26.515440	-26.058014	-25.218809	-25.929684	-26.585421	-26.452057
3164	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1909	-58.269451	-58.269451	-58.269451	-58.269451	-58.269451	-58.269451
3948	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

	214	215
3153	0.000000	0.000000
1406	-23.477995	-14.838339
3863	0.000000	0.000000
3076	0.000000	0.000000
1387	-60.710445	-60.710445
1539	-37.271175	-39.963707
852	-26.296618	-24.204966
3164	0.000000	0.000000
1909	-58.269451	-58.269451
3948	0.000000	0.000000

```
[10 rows x 216 columns]
```

[0]:
```
# Let's do data normalization
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std

# Check the dataset now
X_train[150:160]
```

[0]:
```
             0         1         2         3         4         5         6  \
3153  0.389000  0.288465 -0.042269 -0.000866  0.252767  0.256418  0.225351
1406 -1.113550 -1.107112 -0.941076 -0.942721 -0.944816 -0.945453 -0.947116
3863 -0.254493 -0.245763 -0.031430 -0.041951 -0.041341 -0.136400 -0.078360
3076  0.164634  0.141308 -0.186994 -0.274507 -0.179994 -0.074640 -0.166273
1387 -1.908104 -1.963851 -1.961962 -1.956437 -1.952135 -1.947813 -1.945515
1539 -0.879490 -0.695919 -0.406855 -0.345641 -0.366347 -0.396491 -0.448903
852  -0.831522 -0.765321 -0.537235 -0.489021 -0.468057 -0.476817 -0.570322
3164  0.384768  0.419107  0.655215  0.682502  0.712542  0.668730  0.697595
1909 -1.771455 -1.816508 -1.786389 -1.782096 -1.778895 -1.775426 -1.773809
3948  1.519434  1.525992  1.713022  1.636280  1.540785  1.494519  1.530425

             7         8         9  …       206       207       208  \
3153  0.239746  0.192685  0.117809  …  0.705770  0.705920  0.704998
1406 -0.947890 -0.952211 -0.955565  … -0.652127 -0.640370 -0.660505
3863 -0.092608 -0.245202 -0.237961  …  0.705770  0.705920  0.704998
3076 -0.288916 -0.385563 -0.254006  …  0.705770  0.705920  0.704998
1387 -1.942708 -1.947369 -1.948811  … -1.912083 -1.907389 -1.905813
1539 -0.392089 -0.493312 -0.684758  … -1.019274 -1.026280 -1.022850
852  -0.530541 -0.423335 -0.407744  … -0.378172 -0.401399 -0.435280
3164  0.695023  0.692348  0.626807  …  0.705770  0.705920  0.704998
1909 -1.771618 -1.795357 -1.701734  … -1.806826 -1.802316 -1.800840
3948  1.539089  1.365767  1.528324  …  0.705770  0.705920  0.704998

           209       210       211       212       213       214       215
3153  0.704688  0.704279  0.704222  0.704197  0.704350  0.698001  0.690906
1406 -0.740194 -0.678716 -0.630417 -0.627605 -0.747961 -0.304138  0.060526
3863  0.704688  0.704279  0.704222  0.704197  0.704350  0.698001  0.690906
3076  0.704688  0.704279  0.704222  0.704197  0.704350  0.698001  0.690906
1387 -1.902892 -1.900046 -1.897754 -1.893139 -1.887623 -1.893375 -1.888270
1539 -0.882347 -0.846842 -0.949961 -1.026755 -0.885430 -0.892889 -1.006881
852  -0.414532 -0.377544 -0.407093 -0.433190 -0.424995 -0.424449 -0.337399
3164  0.704688  0.704279  0.704222  0.704197  0.704350  0.698001  0.690906
1909 -1.798048 -1.795333 -1.793136 -1.788708 -1.783407 -1.789183 -1.784568
```

```
3948  0.704688  0.704279  0.704222  0.704197  0.704350  0.698001  0.690906

[10 rows x 216 columns]
```

[0]:
```python
# Lets do few preparation steps to get it into the correct format for Keras
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

print(X_train.shape)
print(lb.classes_)

# Pickel the lb object for future use
filename = 'labels'
outfile = open(filename,'wb')
pickle.dump(lb,outfile)
outfile.close()
```

```
(10620, 216)
['female_angry' 'female_disgust' 'female_fear' 'female_happy'
 'female_neutral' 'female_sad' 'female_surprise' 'male_angry'
 'male_disgust' 'male_fear' 'male_happy' 'male_neutral' 'male_sad'
 'male_surprise']
```

[0]:
```python
X_train = np.expand_dims(X_train, axis=2)
X_test = np.expand_dims(X_test, axis=2)
X_train.shape
```

[0]:
```
(10620, 216, 1)
```

[0]:
```python
model = Sequential()
model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))  #␣
 ↪X_train.shape[1] = No. of Columns
model.add(Activation('relu'))
model.add(Conv1D(256, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
```

```
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(14)) # Target class number
model.add(Activation('softmax'))
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_57 (Conv1D)           (None, 216, 256)          2304
_________________________________________________________________
activation_72 (Activation)   (None, 216, 256)          0
_________________________________________________________________
conv1d_58 (Conv1D)           (None, 216, 256)          524544
_________________________________________________________________
batch_normalization_9 (Batch (None, 216, 256)          1024
_________________________________________________________________
activation_73 (Activation)   (None, 216, 256)          0
_________________________________________________________________
dropout_22 (Dropout)         (None, 216, 256)          0
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 27, 256)           0
_________________________________________________________________
conv1d_59 (Conv1D)           (None, 27, 128)           262272
_________________________________________________________________
activation_74 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_60 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
activation_75 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_61 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
```

```
activation_76 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_62 (Conv1D)           (None, 27, 128)           131200
_________________________________________________________________
batch_normalization_10 (Batc (None, 27, 128)           512
_________________________________________________________________
activation_77 (Activation)   (None, 27, 128)           0
_________________________________________________________________
dropout_23 (Dropout)         (None, 27, 128)           0
_________________________________________________________________
max_pooling1d_14 (MaxPooling (None, 3, 128)            0
_________________________________________________________________
conv1d_63 (Conv1D)           (None, 3, 64)             65600
_________________________________________________________________
activation_78 (Activation)   (None, 3, 64)             0
_________________________________________________________________
conv1d_64 (Conv1D)           (None, 3, 64)             32832
_________________________________________________________________
activation_79 (Activation)   (None, 3, 64)             0
_________________________________________________________________
flatten_9 (Flatten)          (None, 192)               0
_________________________________________________________________
dense_16 (Dense)             (None, 14)                2702
_________________________________________________________________
activation_80 (Activation)   (None, 14)                0
=================================================================
Total params: 1,285,390
Trainable params: 1,284,622
Non-trainable params: 768
_________________________________________________________________
```

[0]:
```python
model.compile(loss='categorical_crossentropy',
 ↪optimizer=opt,metrics=['accuracy'])
model_history=model.fit(X_train, y_train, batch_size=16, epochs=150,
 ↪validation_data=(X_test, y_test))
```

```
Train on 10620 samples, validate on 3540 samples
Epoch 1/150
10620/10620 [==============================] - 82s 8ms/step - loss: 1.8040 -
accuracy: 0.3830 - val_loss: 1.8671 - val_accuracy: 0.3986
Epoch 2/150
10620/10620 [==============================] - 85s 8ms/step - loss: 1.6535 -
accuracy: 0.4325 - val_loss: 1.7403 - val_accuracy: 0.4418
Epoch 3/150
10620/10620 [==============================] - 83s 8ms/step - loss: 1.5376 -
accuracy: 0.4782 - val_loss: 1.6651 - val_accuracy: 0.5079
Epoch 4/150
10620/10620 [==============================] - 86s 8ms/step - loss: 1.4397 -
```

```
accuracy: 0.5223 - val_loss: 1.5794 - val_accuracy: 0.5475
Epoch 5/150
10620/10620 [==============================] - 86s 8ms/step - loss: 1.3564 -
accuracy: 0.5454 - val_loss: 1.4983 - val_accuracy: 0.5647
Epoch 6/150
10620/10620 [==============================] - 85s 8ms/step - loss: 1.2932 -
accuracy: 0.5734 - val_loss: 1.4360 - val_accuracy: 0.5816
Epoch 7/150
10620/10620 [==============================] - 86s 8ms/step - loss: 1.2354 -
accuracy: 0.5892 - val_loss: 1.3774 - val_accuracy: 0.6017
Epoch 8/150
10620/10620 [==============================] - 80s 8ms/step - loss: 1.1860 -
accuracy: 0.6022 - val_loss: 1.3426 - val_accuracy: 0.6079
Epoch 9/150
10620/10620 [==============================] - 77s 7ms/step - loss: 1.1415 -
accuracy: 0.6186 - val_loss: 1.2949 - val_accuracy: 0.6189
Epoch 10/150
10620/10620 [==============================] - 76s 7ms/step - loss: 1.1015 -
accuracy: 0.6345 - val_loss: 1.2699 - val_accuracy: 0.6136
Epoch 11/150
10620/10620 [==============================] - 96s 9ms/step - loss: 1.0736 -
accuracy: 0.6371 - val_loss: 1.2210 - val_accuracy: 0.6367
Epoch 12/150
10620/10620 [==============================] - 83s 8ms/step - loss: 1.0412 -
accuracy: 0.6535 - val_loss: 1.1977 - val_accuracy: 0.6345
Epoch 13/150
10620/10620 [==============================] - 83s 8ms/step - loss: 1.0086 -
accuracy: 0.6559 - val_loss: 1.1666 - val_accuracy: 0.6483
Epoch 14/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.9875 -
accuracy: 0.6692 - val_loss: 1.1523 - val_accuracy: 0.6463
Epoch 15/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.9628 -
accuracy: 0.6728 - val_loss: 1.1200 - val_accuracy: 0.6446
Epoch 16/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.9454 -
accuracy: 0.6775 - val_loss: 1.1337 - val_accuracy: 0.6497
Epoch 17/150
10620/10620 [==============================] - 85s 8ms/step - loss: 0.9248 -
accuracy: 0.6863 - val_loss: 1.0951 - val_accuracy: 0.6641
Epoch 18/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.9077 -
accuracy: 0.6917 - val_loss: 1.0768 - val_accuracy: 0.6695
Epoch 19/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.8832 -
accuracy: 0.6980 - val_loss: 1.0613 - val_accuracy: 0.6698
Epoch 20/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.8665 -
```

```
accuracy: 0.7043 - val_loss: 1.0468 - val_accuracy: 0.6768
Epoch 21/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.8559 -
accuracy: 0.7079 - val_loss: 1.0315 - val_accuracy: 0.6814
Epoch 22/150
10620/10620 [==============================] - 98s 9ms/step - loss: 0.8354 -
accuracy: 0.7174 - val_loss: 1.0320 - val_accuracy: 0.6706
Epoch 23/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.8213 -
accuracy: 0.7185 - val_loss: 1.0098 - val_accuracy: 0.6799
Epoch 24/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.8111 -
accuracy: 0.7230 - val_loss: 0.9943 - val_accuracy: 0.6907
Epoch 25/150
10620/10620 [==============================] - 95s 9ms/step - loss: 0.7925 -
accuracy: 0.7337 - val_loss: 0.9727 - val_accuracy: 0.6921
Epoch 26/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.7825 -
accuracy: 0.7338 - val_loss: 1.0037 - val_accuracy: 0.6794
Epoch 27/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.7693 -
accuracy: 0.7364 - val_loss: 0.9732 - val_accuracy: 0.6839
Epoch 28/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.7565 -
accuracy: 0.7428 - val_loss: 0.9888 - val_accuracy: 0.6780
Epoch 29/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.7431 -
accuracy: 0.7523 - val_loss: 0.9520 - val_accuracy: 0.7020
Epoch 30/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.7307 -
accuracy: 0.7524 - val_loss: 0.9661 - val_accuracy: 0.6946
Epoch 31/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.7188 -
accuracy: 0.7589 - val_loss: 0.9526 - val_accuracy: 0.6969
Epoch 32/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.7088 -
accuracy: 0.7628 - val_loss: 0.9468 - val_accuracy: 0.6960
Epoch 33/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.6965 -
accuracy: 0.7644 - val_loss: 0.9318 - val_accuracy: 0.7014
Epoch 34/150
10620/10620 [==============================] - 88s 8ms/step - loss: 0.6816 -
accuracy: 0.7701 - val_loss: 0.9028 - val_accuracy: 0.7167
Epoch 35/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.6710 -
accuracy: 0.7713 - val_loss: 0.9245 - val_accuracy: 0.6941
Epoch 36/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.6645 -
```

```
accuracy: 0.7751 - val_loss: 0.9020 - val_accuracy: 0.7085
Epoch 37/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.6488 -
accuracy: 0.7830 - val_loss: 0.9291 - val_accuracy: 0.6893
Epoch 38/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.6361 -
accuracy: 0.7887 - val_loss: 0.8958 - val_accuracy: 0.7065
Epoch 39/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.6361 -
accuracy: 0.7845 - val_loss: 0.9065 - val_accuracy: 0.7031
Epoch 40/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.6251 -
accuracy: 0.7911 - val_loss: 0.8913 - val_accuracy: 0.7124
Epoch 41/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.6058 -
accuracy: 0.7997 - val_loss: 0.8739 - val_accuracy: 0.7136
Epoch 42/150
10620/10620 [==============================] - 97s 9ms/step - loss: 0.5986 -
accuracy: 0.7999 - val_loss: 0.8582 - val_accuracy: 0.7226
Epoch 43/150
10620/10620 [==============================] - 92s 9ms/step - loss: 0.5922 -
accuracy: 0.8036 - val_loss: 0.8754 - val_accuracy: 0.7031
Epoch 44/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.5787 -
accuracy: 0.8110 - val_loss: 0.8639 - val_accuracy: 0.7203
Epoch 45/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.5719 -
accuracy: 0.8126 - val_loss: 0.8744 - val_accuracy: 0.7110
Epoch 46/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.5628 -
accuracy: 0.8124 - val_loss: 0.8657 - val_accuracy: 0.7096
Epoch 47/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.5547 -
accuracy: 0.8170 - val_loss: 0.8601 - val_accuracy: 0.7079
Epoch 48/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.5393 -
accuracy: 0.8244 - val_loss: 0.8437 - val_accuracy: 0.7243
Epoch 49/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.5356 -
accuracy: 0.8249 - val_loss: 0.8408 - val_accuracy: 0.7169
Epoch 50/150
10620/10620 [==============================] - 104s 10ms/step - loss: 0.5144 -
accuracy: 0.8341 - val_loss: 0.8514 - val_accuracy: 0.7141
Epoch 51/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.5177 -
accuracy: 0.8267 - val_loss: 0.8344 - val_accuracy: 0.7209
Epoch 52/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.5102 -
```

```
accuracy: 0.8277 - val_loss: 0.8286 - val_accuracy: 0.7271
Epoch 53/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.4992 -
accuracy: 0.8408 - val_loss: 0.8790 - val_accuracy: 0.6938
Epoch 54/150
10620/10620 [==============================] - 103s 10ms/step - loss: 0.4907 -
accuracy: 0.8397 - val_loss: 0.8345 - val_accuracy: 0.7198
Epoch 55/150
10620/10620 [==============================] - 99s 9ms/step - loss: 0.4826 -
accuracy: 0.8445 - val_loss: 0.8307 - val_accuracy: 0.7257
Epoch 56/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.4772 -
accuracy: 0.8429 - val_loss: 0.8163 - val_accuracy: 0.7257
Epoch 57/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.4612 -
accuracy: 0.8527 - val_loss: 0.8233 - val_accuracy: 0.7319
Epoch 58/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.4510 -
accuracy: 0.8548 - val_loss: 0.8315 - val_accuracy: 0.7201
Epoch 59/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.4506 -
accuracy: 0.8576 - val_loss: 0.8186 - val_accuracy: 0.7251
Epoch 60/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.4408 -
accuracy: 0.8610 - val_loss: 0.8786 - val_accuracy: 0.6918
Epoch 61/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.4288 -
accuracy: 0.8647 - val_loss: 0.8238 - val_accuracy: 0.7234
Epoch 62/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.4218 -
accuracy: 0.8665 - val_loss: 0.8138 - val_accuracy: 0.7291
Epoch 63/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.4176 -
accuracy: 0.8626 - val_loss: 0.8302 - val_accuracy: 0.7155
Epoch 64/150
10620/10620 [==============================] - 99s 9ms/step - loss: 0.4058 -
accuracy: 0.8702 - val_loss: 0.8084 - val_accuracy: 0.7311
Epoch 65/150
10620/10620 [==============================] - 103s 10ms/step - loss: 0.3969 -
accuracy: 0.8744 - val_loss: 0.8081 - val_accuracy: 0.7195
Epoch 66/150
10620/10620 [==============================] - 85s 8ms/step - loss: 0.3928 -
accuracy: 0.8760 - val_loss: 0.8084 - val_accuracy: 0.7266
Epoch 67/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.3855 -
accuracy: 0.8793 - val_loss: 0.7772 - val_accuracy: 0.7463
Epoch 68/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.3727 -
```

```
accuracy: 0.8831 - val_loss: 0.8033 - val_accuracy: 0.7282
Epoch 69/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.3676 -
accuracy: 0.8851 - val_loss: 0.8113 - val_accuracy: 0.7246
Epoch 70/150
10620/10620 [==============================] - 98s 9ms/step - loss: 0.3606 -
accuracy: 0.8878 - val_loss: 0.7802 - val_accuracy: 0.7407
Epoch 71/150
10620/10620 [==============================] - 101s 9ms/step - loss: 0.3665 -
accuracy: 0.8844 - val_loss: 0.8000 - val_accuracy: 0.7277
Epoch 72/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.3454 -
accuracy: 0.8933 - val_loss: 0.7889 - val_accuracy: 0.7333
Epoch 73/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.3387 -
accuracy: 0.8953 - val_loss: 0.8155 - val_accuracy: 0.7186
Epoch 74/150
10620/10620 [==============================] - 87s 8ms/step - loss: 0.3328 -
accuracy: 0.8987 - val_loss: 0.7820 - val_accuracy: 0.7362
Epoch 75/150
10620/10620 [==============================] - 106s 10ms/step - loss: 0.3232 -
accuracy: 0.9019 - val_loss: 0.7713 - val_accuracy: 0.7373
Epoch 76/150
10620/10620 [==============================] - 104s 10ms/step - loss: 0.3190 -
accuracy: 0.9028 - val_loss: 0.7775 - val_accuracy: 0.7398
Epoch 77/150
10620/10620 [==============================] - 104s 10ms/step - loss: 0.3140 -
accuracy: 0.9015 - val_loss: 0.7688 - val_accuracy: 0.7353
Epoch 78/150
10620/10620 [==============================] - 86s 8ms/step - loss: 0.3100 -
accuracy: 0.9076 - val_loss: 0.7599 - val_accuracy: 0.7390
Epoch 79/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.3056 -
accuracy: 0.9056 - val_loss: 0.7716 - val_accuracy: 0.7379
Epoch 80/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.2935 -
accuracy: 0.9105 - val_loss: 0.8139 - val_accuracy: 0.7141
Epoch 81/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2878 -
accuracy: 0.9092 - val_loss: 0.8004 - val_accuracy: 0.7285
Epoch 82/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2778 -
accuracy: 0.9162 - val_loss: 0.7694 - val_accuracy: 0.7384
Epoch 83/150
10620/10620 [==============================] - 93s 9ms/step - loss: 0.2796 -
accuracy: 0.9169 - val_loss: 0.7965 - val_accuracy: 0.7249
Epoch 84/150
10620/10620 [==============================] - 106s 10ms/step - loss: 0.2700 -
```

```
accuracy: 0.9167 - val_loss: 0.8011 - val_accuracy: 0.7232
Epoch 85/150
10620/10620 [==============================] - 105s 10ms/step - loss: 0.2662 -
accuracy: 0.9215 - val_loss: 0.7558 - val_accuracy: 0.7415
Epoch 86/150
10620/10620 [==============================] - 96s 9ms/step - loss: 0.2620 -
accuracy: 0.9225 - val_loss: 0.7722 - val_accuracy: 0.7393
Epoch 87/150
10620/10620 [==============================] - 81s 8ms/step - loss: 0.2545 -
accuracy: 0.9231 - val_loss: 0.7921 - val_accuracy: 0.7254
Epoch 88/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2509 -
accuracy: 0.9241 - val_loss: 0.7711 - val_accuracy: 0.7379
Epoch 89/150
10620/10620 [==============================] - 97s 9ms/step - loss: 0.2407 -
accuracy: 0.9289 - val_loss: 0.7736 - val_accuracy: 0.7328
Epoch 90/150
10620/10620 [==============================] - 88s 8ms/step - loss: 0.2390 -
accuracy: 0.9282 - val_loss: 0.7362 - val_accuracy: 0.7460
Epoch 91/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.2387 -
accuracy: 0.9271 - val_loss: 0.7532 - val_accuracy: 0.7438
Epoch 92/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2225 -
accuracy: 0.9365 - val_loss: 0.7647 - val_accuracy: 0.7421
Epoch 93/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.2221 -
accuracy: 0.9356 - val_loss: 0.7882 - val_accuracy: 0.7311
Epoch 94/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2109 -
accuracy: 0.9391 - val_loss: 0.7565 - val_accuracy: 0.7441
Epoch 95/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2113 -
accuracy: 0.9389 - val_loss: 0.7515 - val_accuracy: 0.7379
Epoch 96/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2071 -
accuracy: 0.9415 - val_loss: 0.7605 - val_accuracy: 0.7415
Epoch 97/150
10620/10620 [==============================] - 106s 10ms/step - loss: 0.1988 -
accuracy: 0.9438 - val_loss: 0.7718 - val_accuracy: 0.7359
Epoch 98/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.2037 -
accuracy: 0.9403 - val_loss: 0.7506 - val_accuracy: 0.7469
Epoch 99/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.1910 -
accuracy: 0.9458 - val_loss: 0.7634 - val_accuracy: 0.7415
Epoch 100/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1886 -
```

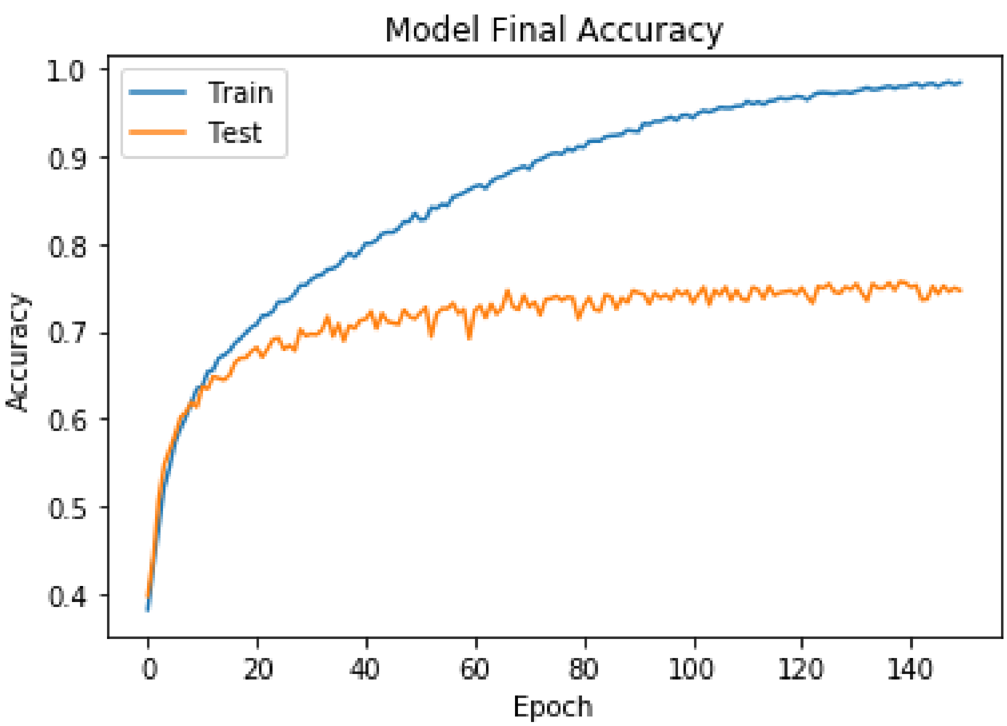
```
accuracy: 0.9461 - val_loss: 0.7420 - val_accuracy: 0.7458
Epoch 101/150
10620/10620 [==============================] - 94s 9ms/step - loss: 0.1892 -
accuracy: 0.9431 - val_loss: 0.7684 - val_accuracy: 0.7322
Epoch 102/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1819 -
accuracy: 0.9483 - val_loss: 0.7628 - val_accuracy: 0.7418
Epoch 103/150
10620/10620 [==============================] - 93s 9ms/step - loss: 0.1774 -
accuracy: 0.9511 - val_loss: 0.7509 - val_accuracy: 0.7503
Epoch 104/150
10620/10620 [==============================] - 93s 9ms/step - loss: 0.1713 -
accuracy: 0.9495 - val_loss: 0.8109 - val_accuracy: 0.7299
Epoch 105/150
10620/10620 [==============================] - 93s 9ms/step - loss: 0.1684 -
accuracy: 0.9518 - val_loss: 0.7462 - val_accuracy: 0.7466
Epoch 106/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.1625 -
accuracy: 0.9548 - val_loss: 0.7591 - val_accuracy: 0.7412
Epoch 107/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1644 -
accuracy: 0.9542 - val_loss: 0.7429 - val_accuracy: 0.7506
Epoch 108/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1602 -
accuracy: 0.9542 - val_loss: 0.7894 - val_accuracy: 0.7359
Epoch 109/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1597 -
accuracy: 0.9564 - val_loss: 0.7554 - val_accuracy: 0.7466
Epoch 110/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1503 -
accuracy: 0.9565 - val_loss: 0.7740 - val_accuracy: 0.7367
Epoch 111/150
10620/10620 [==============================] - 92s 9ms/step - loss: 0.1439 -
accuracy: 0.9617 - val_loss: 0.7775 - val_accuracy: 0.7350
Epoch 112/150
10620/10620 [==============================] - 105s 10ms/step - loss: 0.1454 -
accuracy: 0.9587 - val_loss: 0.7452 - val_accuracy: 0.7500
Epoch 113/150
10620/10620 [==============================] - 105s 10ms/step - loss: 0.1423 -
accuracy: 0.9612 - val_loss: 0.7483 - val_accuracy: 0.7472
Epoch 114/150
10620/10620 [==============================] - 85s 8ms/step - loss: 0.1419 -
accuracy: 0.9582 - val_loss: 0.7892 - val_accuracy: 0.7373
Epoch 115/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1349 -
accuracy: 0.9621 - val_loss: 0.7537 - val_accuracy: 0.7508
Epoch 116/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.1306 -
```

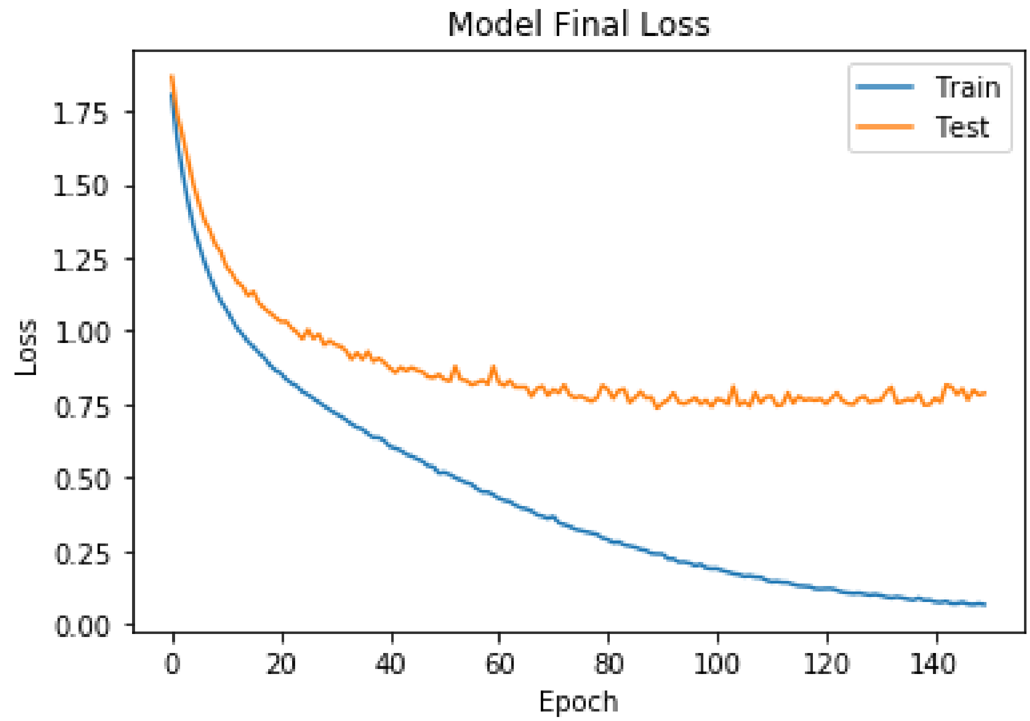
```
accuracy: 0.9628 - val_loss: 0.7771 - val_accuracy: 0.7410
Epoch 117/150
10620/10620 [==============================] - 106s 10ms/step - loss: 0.1293 -
accuracy: 0.9657 - val_loss: 0.7636 - val_accuracy: 0.7441
Epoch 118/150
10620/10620 [==============================] - 87s 8ms/step - loss: 0.1279 -
accuracy: 0.9644 - val_loss: 0.7676 - val_accuracy: 0.7449
Epoch 119/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1205 -
accuracy: 0.9652 - val_loss: 0.7632 - val_accuracy: 0.7463
Epoch 120/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1193 -
accuracy: 0.9675 - val_loss: 0.7683 - val_accuracy: 0.7418
Epoch 121/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.1205 -
accuracy: 0.9668 - val_loss: 0.7517 - val_accuracy: 0.7494
Epoch 122/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1207 -
accuracy: 0.9636 - val_loss: 0.7698 - val_accuracy: 0.7398
Epoch 123/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.1155 -
accuracy: 0.9684 - val_loss: 0.7895 - val_accuracy: 0.7319
Epoch 124/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1089 -
accuracy: 0.9713 - val_loss: 0.7627 - val_accuracy: 0.7511
Epoch 125/150
10620/10620 [==============================] - 102s 10ms/step - loss: 0.1065 -
accuracy: 0.9718 - val_loss: 0.7521 - val_accuracy: 0.7492
Epoch 126/150
10620/10620 [==============================] - 104s 10ms/step - loss: 0.1041 -
accuracy: 0.9711 - val_loss: 0.7488 - val_accuracy: 0.7540
Epoch 127/150
10620/10620 [==============================] - 107s 10ms/step - loss: 0.1060 -
accuracy: 0.9705 - val_loss: 0.7695 - val_accuracy: 0.7446
Epoch 128/150
10620/10620 [==============================] - 105s 10ms/step - loss: 0.1025 -
accuracy: 0.9719 - val_loss: 0.7765 - val_accuracy: 0.7432
Epoch 129/150
10620/10620 [==============================] - 82s 8ms/step - loss: 0.0988 -
accuracy: 0.9722 - val_loss: 0.7561 - val_accuracy: 0.7506
Epoch 130/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.1012 -
accuracy: 0.9713 - val_loss: 0.7625 - val_accuracy: 0.7477
Epoch 131/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.0977 -
accuracy: 0.9734 - val_loss: 0.7623 - val_accuracy: 0.7540
Epoch 132/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.0925 -
```

```
accuracy: 0.9749 - val_loss: 0.7896 - val_accuracy: 0.7438
Epoch 133/150
10620/10620 [==============================] - 98s 9ms/step - loss: 0.0894 -
accuracy: 0.9770 - val_loss: 0.8087 - val_accuracy: 0.7350
Epoch 134/150
10620/10620 [==============================] - 99s 9ms/step - loss: 0.0926 -
accuracy: 0.9749 - val_loss: 0.7547 - val_accuracy: 0.7540
Epoch 135/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.0897 -
accuracy: 0.9754 - val_loss: 0.7588 - val_accuracy: 0.7492
Epoch 136/150
10620/10620 [==============================] - 89s 8ms/step - loss: 0.0851 -
accuracy: 0.9773 - val_loss: 0.7670 - val_accuracy: 0.7492
Epoch 137/150
10620/10620 [==============================] - 88s 8ms/step - loss: 0.0821 -
accuracy: 0.9788 - val_loss: 0.7618 - val_accuracy: 0.7551
Epoch 138/150
10620/10620 [==============================] - 109s 10ms/step - loss: 0.0888 -
accuracy: 0.9765 - val_loss: 0.7848 - val_accuracy: 0.7469
Epoch 139/150
10620/10620 [==============================] - 108s 10ms/step - loss: 0.0813 -
accuracy: 0.9787 - val_loss: 0.7467 - val_accuracy: 0.7565
Epoch 140/150
10620/10620 [==============================] - 90s 9ms/step - loss: 0.0820 -
accuracy: 0.9782 - val_loss: 0.7486 - val_accuracy: 0.7551
Epoch 141/150
10620/10620 [==============================] - 98s 9ms/step - loss: 0.0759 -
accuracy: 0.9803 - val_loss: 0.7702 - val_accuracy: 0.7503
Epoch 142/150
10620/10620 [==============================] - 104s 10ms/step - loss: 0.0742 -
accuracy: 0.9820 - val_loss: 0.7575 - val_accuracy: 0.7523
Epoch 143/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.0779 -
accuracy: 0.9782 - val_loss: 0.8159 - val_accuracy: 0.7359
Epoch 144/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.0699 -
accuracy: 0.9812 - val_loss: 0.8086 - val_accuracy: 0.7367
Epoch 145/150
10620/10620 [==============================] - 84s 8ms/step - loss: 0.0700 -
accuracy: 0.9820 - val_loss: 0.7834 - val_accuracy: 0.7514
Epoch 146/150
10620/10620 [==============================] - 93s 9ms/step - loss: 0.0748 -
accuracy: 0.9785 - val_loss: 0.8031 - val_accuracy: 0.7424
Epoch 147/150
10620/10620 [==============================] - 106s 10ms/step - loss: 0.0685 -
accuracy: 0.9818 - val_loss: 0.7651 - val_accuracy: 0.7517
Epoch 148/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.0665 -
```

```
accuracy: 0.9836 - val_loss: 0.7977 - val_accuracy: 0.7452
Epoch 149/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.0703 -
accuracy: 0.9808 - val_loss: 0.7814 - val_accuracy: 0.7494
Epoch 150/150
10620/10620 [==============================] - 83s 8ms/step - loss: 0.0663 -
accuracy: 0.9829 - val_loss: 0.7867 - val_accuracy: 0.7466
```

[0]:
```
plt.plot(model_history.history['accuracy'])
plt.plot(model_history.history['val_accuracy'])
plt.title('Model Final Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Final Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```





[0]:
```python
# Save model and weights
model_name = 'Model_final.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Save model and weights at %s ' % model_path)

# Save the model to disk
model_json = model.to_json()
with open("model_final_json.json", "w") as json_file:
    json_file.write(model_json)
```

```
Save model and weights at /home/subodh/Second Semester/ML/Random
Project/Audio/saved_models/Model_final.h5
```

[0]:
```python
# loading json and model architecture
json_file = open('model_final_json.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
```

```python
# load weights into new model
loaded_model.load_weights("saved_models/Model_final.h5")
print("Loaded model from disk")

# Keras optimiser
opt = keras.optimizers.rmsprop(lr=0.00001, decay=1e-6)
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics=['accuracy'])
score = loaded_model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 74.66%
```

[0]:
```python
preds = loaded_model.predict(X_test,
                         batch_size=16,
                         verbose=1)

preds=preds.argmax(axis=1)
preds
```

```
3540/3540 [==============================] - 5s 2ms/step
```

[0]:
```
array([8, 4, 5, …, 3, 2, 4])
```

[0]:
```python
# predictions
preds = preds.astype(int).flatten()
preds = (lb.inverse_transform((preds)))
preds = pd.DataFrame({'predictedvalues': preds})

# Actual labels
actual=y_test.argmax(axis=1)
actual = actual.astype(int).flatten()
actual = (lb.inverse_transform((actual)))
actual = pd.DataFrame({'actualvalues': actual})

# Lets combined both of them into a single dataframe
finaldf = actual.join(preds)
finaldf[170:180]
```

[0]:

	actualvalues	predictedvalues
170	female_angry	female_angry
171	female_neutral	female_neutral
172	female_angry	male_happy
173	female_fear	female_angry
174	male_sad	female_disgust
175	female_angry	female_angry

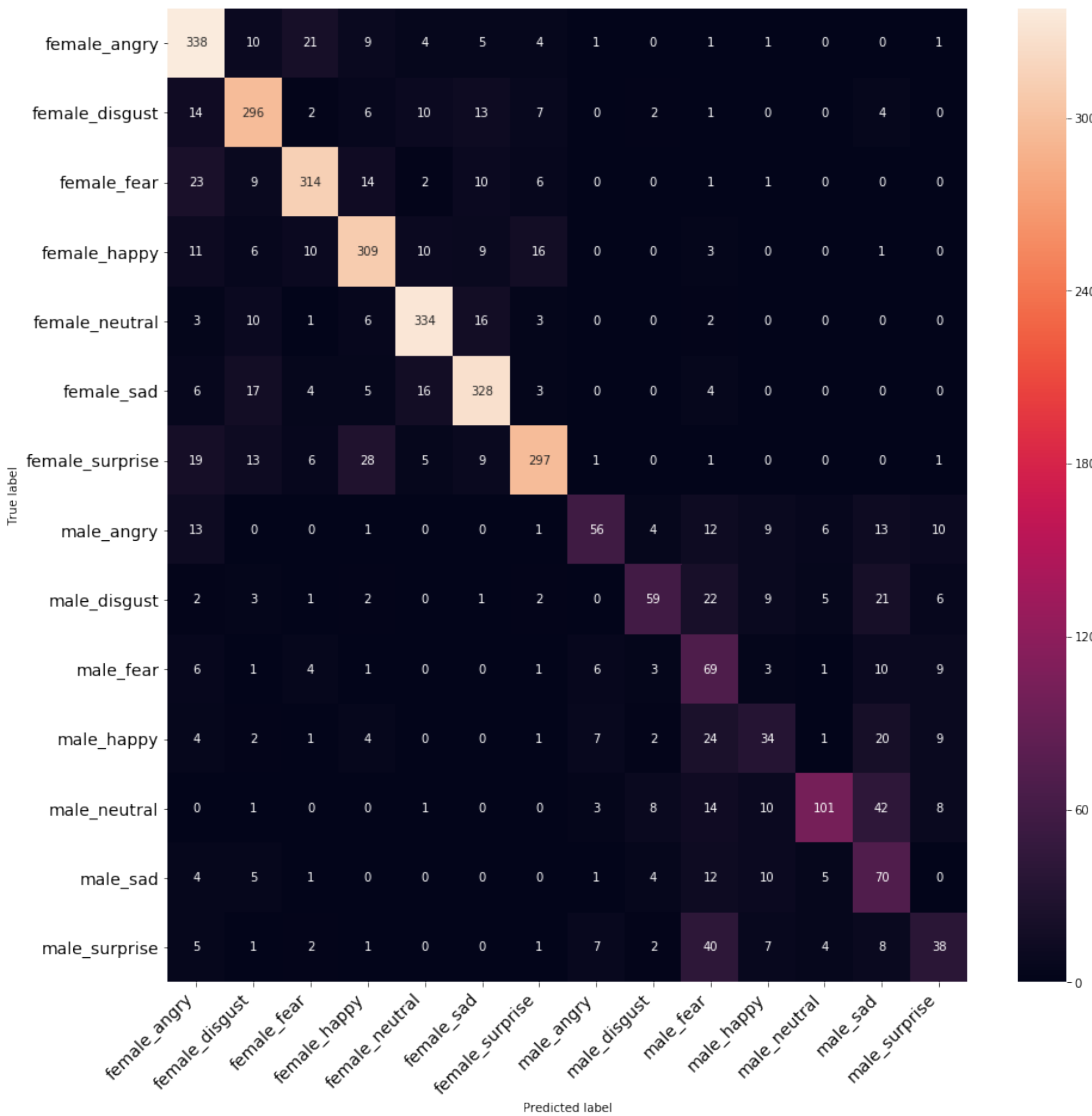
```
176     female_fear     female_fear
177    male_neutral       male_fear
178  female_neutral  female_neutral
179    female_happy    female_happy
```

[0]:
```python
# Write out the predictions to disk
finaldf.to_csv('Predictions_final.csv', index=False)
finaldf.groupby('predictedvalues').count()
```

[0]:
```
                 actualvalues
predictedvalues
female_angry              448
female_disgust            374
female_fear               367
female_happy              386
female_neutral            382
female_sad                391
female_surprise           342
male_angry                 82
male_disgust               84
male_fear                 206
male_happy                 84
male_neutral              123
male_sad                  189
male_surprise              82
```

[0]:
```python
# Get the predictions file
finaldf = pd.read_csv("Predictions_final.csv")
classes = finaldf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(finaldf.actualvalues, finaldf.predictedvalues)
print(accuracy_score(finaldf.actualvalues, finaldf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

```
0.7466101694915255
```



[0]:
```python
# Classification report
classes = finaldf.actualvalues.unique()
classes.sort()
print(classification_report(finaldf.actualvalues, finaldf.predictedvalues,
 ↪target_names=classes))
```

```
                 precision    recall  f1-score   support

   female_angry       0.75      0.86      0.80       395
 female_disgust       0.79      0.83      0.81       355
    female_fear       0.86      0.83      0.84       380
   female_happy       0.80      0.82      0.81       375
 female_neutral       0.87      0.89      0.88       375
```

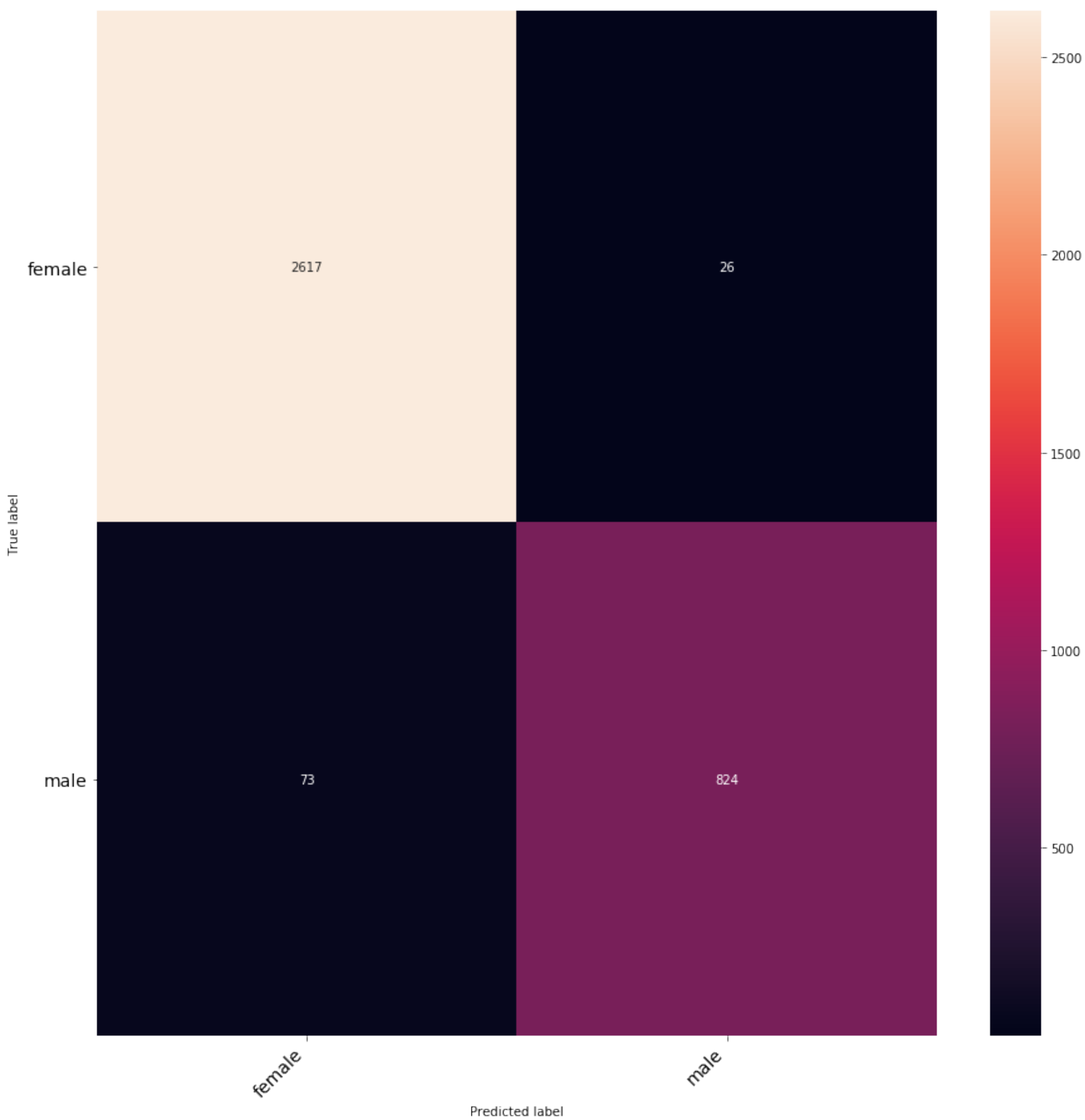
```
     female_sad       0.84      0.86      0.85       383
 female_surprise      0.87      0.78      0.82       380
      male_angry      0.68      0.45      0.54       125
    male_disgust      0.70      0.44      0.54       133
       male_fear      0.33      0.61      0.43       114
      male_happy      0.40      0.31      0.35       109
    male_neutral      0.82      0.54      0.65       188
        male_sad      0.37      0.62      0.47       112
   male_surprise      0.46      0.33      0.38       116

        accuracy                          0.75      3540
       macro avg      0.68      0.65      0.66      3540
    weighted avg      0.76      0.75      0.75      3540
```

[0]:

```
modidf = finaldf
modidf['actualvalues'] = finaldf.actualvalues.replace({'female_angry':'female'
                                       , 'female_disgust':'female'
                                       , 'female_fear':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprise':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_fear':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprise':'male'
                                       , 'male_neutral':'male'
                                       , 'male_disgust':'male'
                                      })

modidf['predictedvalues'] = finaldf.predictedvalues.replace({'female_angry':
 ↪'female'
                                       , 'female_disgust':'female'
                                       , 'female_fear':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprise':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_fear':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprise':'male'
                                       , 'male_neutral':'male'
                                       , 'male_disgust':'male'
```

```
                                    })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

0.9720338983050848



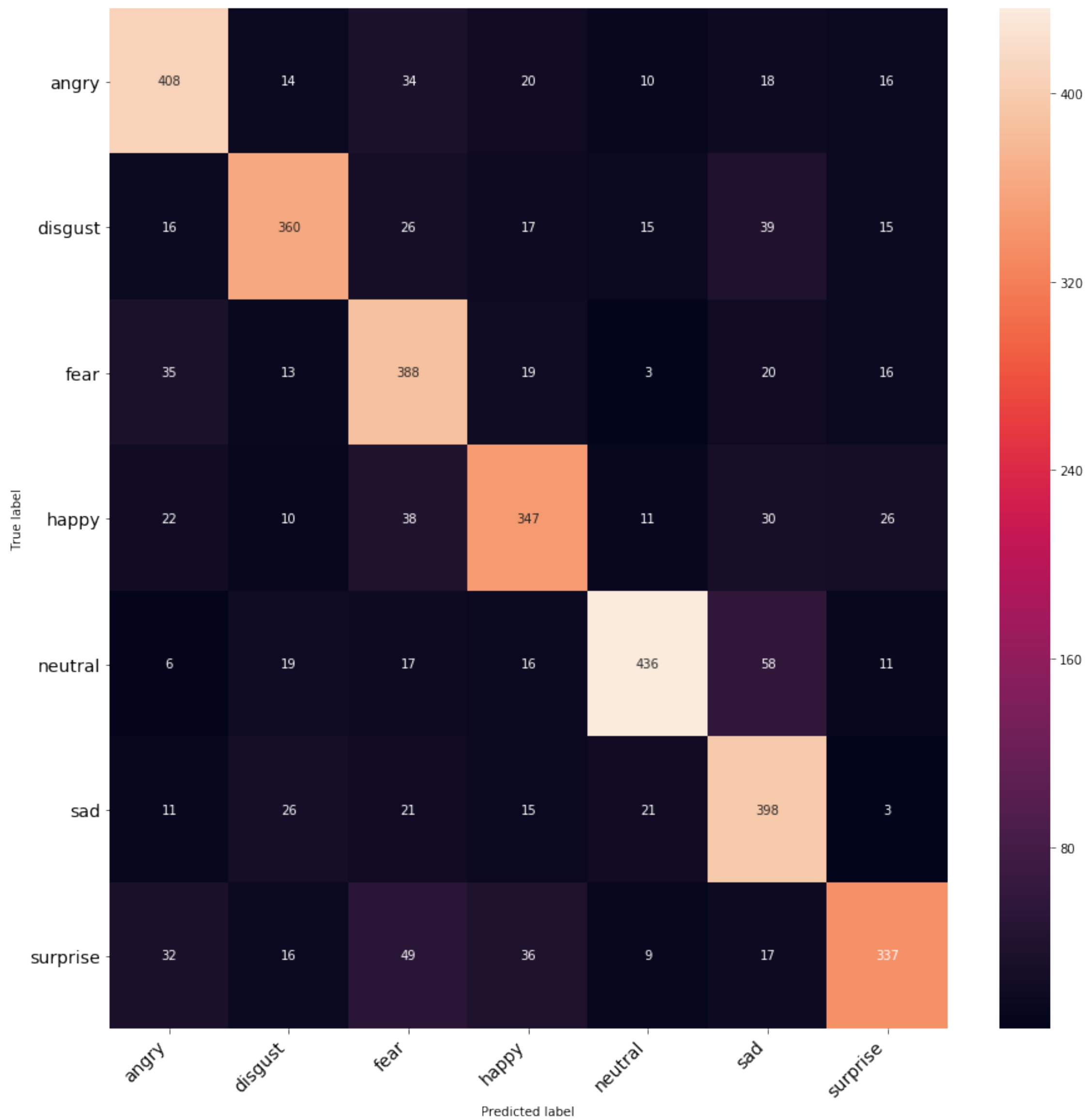
```
[0]: # Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
              precision    recall  f1-score   support

      female       0.97      0.99      0.98      2643
        male       0.97      0.92      0.94       897

    accuracy                           0.97      3540
   macro avg       0.97      0.95      0.96      3540
weighted avg       0.97      0.97      0.97      3540
```

```
[0]: modidf = pd.read_csv("Predictions_final.csv")
modidf['actualvalues'] = modidf.actualvalues.replace({'female_angry':'angry'
                                        , 'female_disgust':'disgust'
                                        , 'female_fear':'fear'
                                        , 'female_happy':'happy'
                                        , 'female_sad':'sad'
                                        , 'female_surprise':'surprise'
                                        , 'female_neutral':'neutral'
                                        , 'male_angry':'angry'
                                        , 'male_fear':'fear'
                                        , 'male_happy':'happy'
                                        , 'male_sad':'sad'
                                        , 'male_surprise':'surprise'
                                        , 'male_neutral':'neutral'
                                        , 'male_disgust':'disgust'
                                       })

modidf['predictedvalues'] = modidf.predictedvalues.replace({'female_angry':
 ↪'angry'
                                        , 'female_disgust':'disgust'
                                        , 'female_fear':'fear'
                                        , 'female_happy':'happy'
                                        , 'female_sad':'sad'
                                        , 'female_surprise':'surprise'
                                        , 'female_neutral':'neutral'
                                        , 'male_angry':'angry'
                                        , 'male_fear':'fear'
                                        , 'male_happy':'happy'
                                        , 'male_sad':'sad'
                                        , 'male_surprise':'surprise'
                                        , 'male_neutral':'neutral'
```

```
                                          , 'male_disgust':'disgust'
                                         })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print(accuracy_score(modidf.actualvalues, modidf.predictedvalues))
print_confusion_matrix(c, class_names = classes)
```

```
0.7553672316384181
```



[0]:
```
# Classification report
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
↪target_names=classes))
```

```
              precision    recall  f1-score   support

       angry       0.77      0.78      0.78       520
     disgust       0.79      0.74      0.76       488
        fear       0.68      0.79      0.73       494
       happy       0.74      0.72      0.73       484
     neutral       0.86      0.77      0.82       563
         sad       0.69      0.80      0.74       495
    surprise       0.79      0.68      0.73       496

    accuracy                           0.76      3540
   macro avg       0.76      0.75      0.75      3540
weighted avg       0.76      0.76      0.76      3540
```

5.9.1 The accuracy we got after augmentation is 74.66% as compared to 72.10% without augmentation.

6 Experiment 5: Train and test on RAVDESS(Not Randomized)

In this experiment we will take the first 20 actors of RAVDESS dataset and train model on them, then we will test it's accuracy on the rest of the actors.

From earlier experiment we have found that augmentation methods helps in improving the accuracy, we will put that to test here as well. Back to Experiments and Results

This experiment is heavily influenced by this experiment by Reza Chu

6.0.1 1. Reading Data

[0]:
```
path_ = './RAVDESS/'
data = os.listdir(path_)
data.sort()
print(data)
```

```
['Actor_01', 'Actor_02', 'Actor_03', 'Actor_04', 'Actor_05', 'Actor_06',
'Actor_07', 'Actor_08', 'Actor_09', 'Actor_10', 'Actor_11', 'Actor_12',
'Actor_13', 'Actor_14', 'Actor_15', 'Actor_16', 'Actor_17', 'Actor_18',
'Actor_19', 'Actor_20', 'Actor_21', 'Actor_22', 'Actor_23', 'Actor_24']
```

[0]:
```
data_df = pd.DataFrame(columns=['path', 'source', 'actor', 'gender',
```

```
                                'intensity', 'statement', 'repetition',␣
↪'emotion'])
count = 0
for i in data:
    file_list = os.listdir(path_ + i)
    for f in file_list:
        nm = f.split('.')[0].split('-')
        path = path_ + i + '/' + f
        src = int(nm[1])
        actor = int(nm[-1])
        emotion = int(nm[2])

        if int(actor)%2 == 0:
            gender = "female"
        else:
            gender = "male"

        if nm[3] == '01':
            intensity = 0
        else:
            intensity = 1

        if nm[4] == '01':
            statement = 0
        else:
            statement = 1

        if nm[5] == '01':
            repeat = 0
        else:
            repeat = 1

        data_df.loc[count] = [path, src, actor, gender, intensity, statement,␣
↪repeat, emotion]
        count += 1
```

[0]:
```
print(len(data_df))
data_df.head()
```

1440

[0]:
```
                                           path  source  actor gender  intensity  \
0  ./RAVDESS/Actor_01/03-01-05-01-01-01-01.wav       1      1   male          0
1  ./RAVDESS/Actor_01/03-01-07-01-02-01-01.wav       1      1   male          0
2  ./RAVDESS/Actor_01/03-01-07-01-01-01-01.wav       1      1   male          0
3  ./RAVDESS/Actor_01/03-01-02-01-02-02-01.wav       1      1   male          0
4  ./RAVDESS/Actor_01/03-01-03-01-01-02-01.wav       1      1   male          0
```

```
   statement  repetition  emotion         label
0          0           0        5    male_angry
1          1           0        7  male_disgust
2          0           0        7  male_disgust
3          1           1        2     male_calm
4          0           1        3    male_happy
```

6.0.2 2. Plotting waveform and Spectogram

[0]:
```python
# We will choose a random file

filename = data_df.path[1057]
print (filename)

samples, sample_rate = librosa.load(filename)
sample_rate, samples
```

```
./RAVDESS/Actor_18/03-01-02-02-01-01-18.wav
```

[0]:
```
(22050, array([ 4.7052872e-11, -6.9230288e-11,  9.6121958e-11, …,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00], dtype=float32))
```

[0]:
```python
len(samples), sample_rate
```

[0]:
```
(79460, 22050)
```

[0]:
```python
# define function to plot spectogram
def log_specgram(audio, sample_rate, window_size=20,
                 step_size=10, eps=1e-10):
    nperseg = int(round(window_size * sample_rate / 1e3))
    noverlap = int(round(step_size * sample_rate / 1e3))
    freqs, times, spec = signal.spectrogram(audio,
                                    fs=sample_rate,
                                    window='hann',
                                    nperseg=nperseg,
                                    noverlap=noverlap,
                                    detrend=False)
    return freqs, times, np.log(spec.T.astype(np.float32) + eps)
```

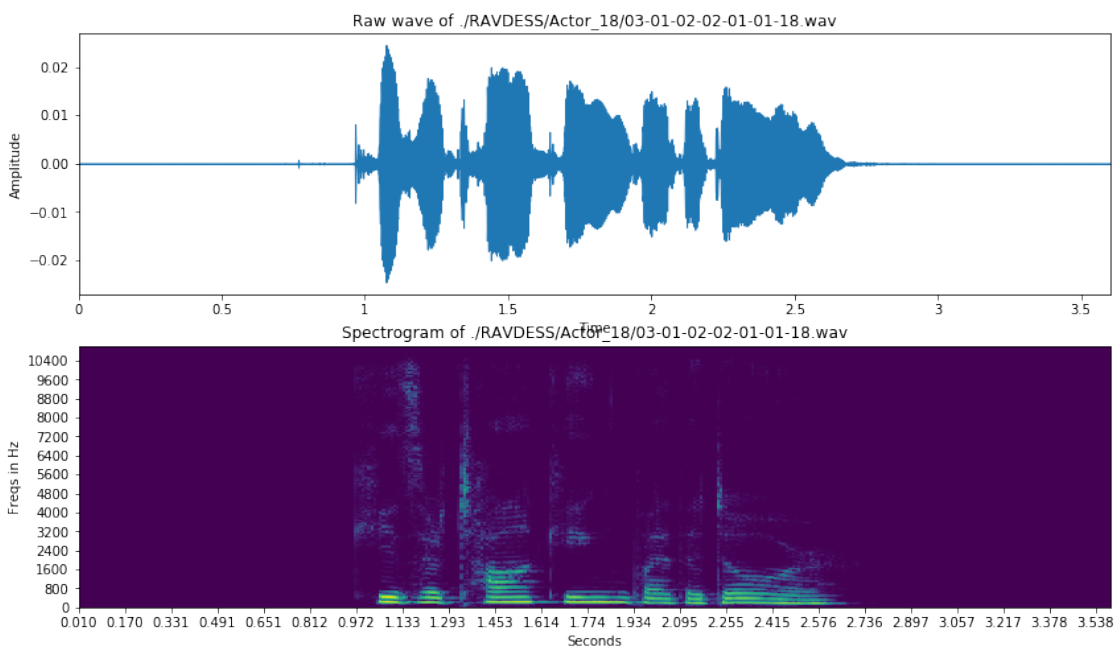
[0]:
```python
sample_rate/len(samples)
```

[0]:
```
0.27749811225773974
```

[0]:
```python
from scipy.fftpack import fft
from scipy import signal
from scipy.io import wavfile
```

```python
from tqdm import tqdm

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

fig = plt.figure(figsize=(14, 8))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + filename)
ax1.set_ylabel('Amplitude')
librosa.display.waveplot(samples, sr=sample_rate)
ax2 = fig.add_subplot(212)
ax2.imshow(spectrogram.T, aspect='auto', origin='lower',
           extent=[times.min(), times.max(), freqs.min(), freqs.max()])
ax2.set_yticks(freqs[::16])
ax2.set_xticks(times[::16])
ax2.set_title('Spectrogram of ' + filename)
ax2.set_ylabel('Freqs in Hz')
ax2.set_xlabel('Seconds')
```

[0]: Text(0.5, 0, 'Seconds')

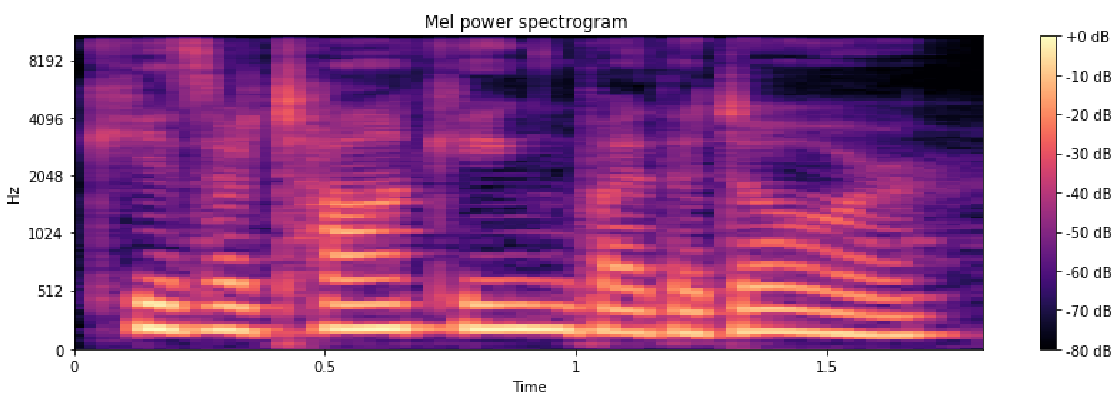


[0]:
```python
mean = np.mean(spectrogram, axis=0)
std = np.std(spectrogram, axis=0)
spectrogram = (spectrogram - mean) / std
```

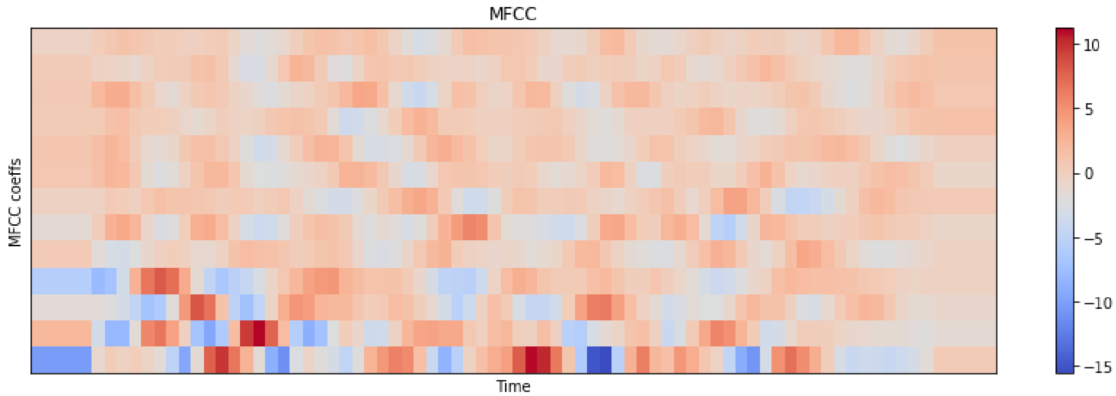
[0]:
```
aa, bb = librosa.effects.trim(samples, top_db = 30)
aa, bb
```

[0]:
```
(array([-4.4609305e-07,  4.4025666e-07, -4.2128403e-07, …,
         9.1547212e-05,  9.2423223e-05,  4.7072081e-05], dtype=float32),
 array([20480, 59904]))
```

[0]:
```
# Plotting Mel Power Spectrogram
S = librosa.feature.melspectrogram(aa, sr=sample_rate, n_mels=128)

# Convert to log scale (dB). We'll use the peak power (max) as reference.
log_S = librosa.power_to_db(S, ref=np.max)

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



[0]:
```
# Plotting MFCC
mfcc = librosa.feature.mfcc(S=log_S, n_mfcc=13)

# Let's pad on the first and second deltas while we're at it
delta2_mfcc = librosa.feature.delta(mfcc, order=2)

plt.figure(figsize=(12, 4))
librosa.display.specshow(delta2_mfcc)
plt.ylabel('MFCC coeffs')
plt.xlabel('Time')
plt.title('MFCC')
plt.colorbar()
plt.tight_layout()
```



[0]:
```
# Let's hear original sound
ipd.Audio(samples, rate = sample_rate)
```

[0]: <IPython.lib.display.Audio object>

[0]:
```
# Let's hear silence-trimmed sound by librosa.effects.trim()
# This will trim
ipd.Audio(aa, rate = sample_rate)
```

[0]: <IPython.lib.display.Audio object>

[0]:
```
# Silence trimmed Sound by manuel trimming
samples_cut = samples[10000:-12500]
ipd.Audio(samples_cut, rate=sample_rate)
```

[0]: <IPython.lib.display.Audio object>

[0]:
```
data_df.emotion.unique()
```

[0]: array([5, 7, 2, 3, 6, 8, 4, 1], dtype=object)

[0]:
```
# All class

label8_list = []
for i in range(len(data_df)):
    if data_df.emotion[i] == 1:
        lb = "_neutral"
    elif data_df.emotion[i] == 2:
        lb = "_calm"
    elif data_df.emotion[i] == 3:
        lb = "_happy"
    elif data_df.emotion[i] == 4:
        lb = "_sad"
```

```
    elif data_df.emotion[i] == 5:
        lb = "_angry"
    elif data_df.emotion[i] == 6:
        lb = "_fearful"
    elif data_df.emotion[i] == 7:
        lb = "_disgust"
    elif data_df.emotion[i] == 8:
        lb = "_surprised"
    else:
        lb = "_none"

    # Add gender to the label
    label8_list.append(data_df.gender[i]  + lb)

len(label8_list)
```

[0]:
```
1440
```

[0]:
```
data_df['label'] = label8_list
data_df.head()
```

[0]:
```
                                           path  source  actor gender  intensity  \
0  ./RAVDESS/Actor_01/03-01-05-01-01-01-01.wav       1      1   male          0
1  ./RAVDESS/Actor_01/03-01-07-01-02-01-01.wav       1      1   male          0
2  ./RAVDESS/Actor_01/03-01-07-01-01-01-01.wav       1      1   male          0
3  ./RAVDESS/Actor_01/03-01-02-01-02-02-01.wav       1      1   male          0
4  ./RAVDESS/Actor_01/03-01-03-01-01-02-01.wav       1      1   male          0

  statement  repetition  emotion         label
0         0           0        5    male_angry
1         1           0        7  male_disgust
2         0           0        7  male_disgust
3         1           1        2     male_calm
4         0           1        3    male_happy
```

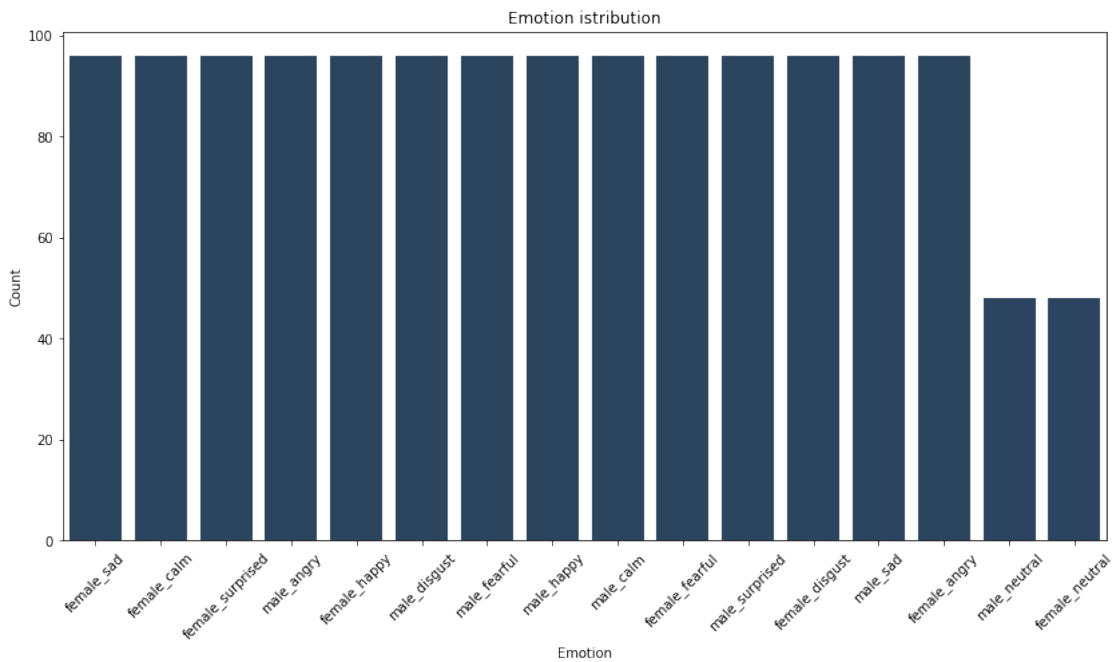
[0]:
```
print (data_df.label.value_counts().keys())
```

```
Index(['female_sad', 'female_calm', 'female_surprised', 'male_angry',
       'female_happy', 'male_disgust', 'male_fearful', 'male_happy',
       'male_calm', 'female_fearful', 'male_surprised', 'female_disgust',
       'male_sad', 'female_angry', 'male_neutral', 'female_neutral'],
      dtype='object')
```

[0]:
```
# Plotting the emotion distribution

def plot_emotion_dist(dist, color_code='#C2185B', title="Plot"):
    """
```

```
    To plot the data distributioin by class.
    Arg:
      dist: pandas series of label count.
    """
    tmp_df = pd.DataFrame()
    tmp_df['Emotion'] = list(dist.keys())
    tmp_df['Count'] = list(dist)
    fig, ax = plt.subplots(figsize=(14, 7))
    ax = sns.barplot(x="Emotion", y='Count', color=color_code, data=tmp_df)
    ax.set_title(title)
    ax.set_xticklabels(ax.get_xticklabels(),rotation=45)
```

[0]:
```
a = data_df.label.value_counts()
plot_emotion_dist(a, "#234567", "Emotion istribution")
```



[0]:
```
data2_df = data_df.copy()
```

[0]:
```
data2_df
```

[0]:
```
                                      path source actor  gender  \
0     ./RAVDESS/Actor_01/03-01-05-01-01-01-01.wav      1     1    male
1     ./RAVDESS/Actor_01/03-01-07-01-02-01-01.wav      1     1    male
2     ./RAVDESS/Actor_01/03-01-07-01-01-01-01.wav      1     1    male
3     ./RAVDESS/Actor_01/03-01-02-01-02-02-01.wav      1     1    male
4     ./RAVDESS/Actor_01/03-01-03-01-01-02-01.wav      1     1    male
```

```
…                                               …     …    …      …
1435  ./RAVDESS/Actor_24/03-01-05-01-02-02-24.wav      1     24  female
1436  ./RAVDESS/Actor_24/03-01-07-01-01-02-24.wav      1     24  female
1437  ./RAVDESS/Actor_24/03-01-04-02-02-02-24.wav      1     24  female
1438  ./RAVDESS/Actor_24/03-01-06-01-02-01-24.wav      1     24  female
1439  ./RAVDESS/Actor_24/03-01-02-02-02-01-24.wav      1     24  female

     intensity statement repetition emotion           label
0            0          0          0       5      male_angry
1            0          1          0       7    male_disgust
2            0          0          0       7    male_disgust
3            0          1          1       2       male_calm
4            0          0          1       3      male_happy
…          …        …          …      …               …
1435         0          1          1       5    female_angry
1436         0          0          1       7  female_disgust
1437         1          1          1       4      female_sad
1438         0          1          0       6  female_fearful
1439         1          1          0       2     female_calm

[1440 rows x 9 columns]
```

[0]:
```
tmp1 = data2_df[data2_df.actor == 21]
tmp2 = data2_df[data2_df.actor == 22]
tmp3 = data2_df[data2_df.actor == 23]
tmp4 = data2_df[data2_df.actor == 24]
```

[0]:
```
data3_df = pd.concat([tmp1, tmp2, tmp3, tmp4],ignore_index=True).
↪reset_index(drop=True)
```

[0]:
```
data3_df
```

[0]:
```
                                            path source actor  gender  \
0    ./RAVDESS/Actor_21/03-01-04-01-01-01-21.wav      1     21    male
1    ./RAVDESS/Actor_21/03-01-05-02-02-01-21.wav      1     21    male
2    ./RAVDESS/Actor_21/03-01-06-02-01-02-21.wav      1     21    male
3    ./RAVDESS/Actor_21/03-01-05-02-01-02-21.wav      1     21    male
4    ./RAVDESS/Actor_21/03-01-05-01-02-01-21.wav      1     21    male
..                                              …     …    …      …
235  ./RAVDESS/Actor_24/03-01-05-01-02-02-24.wav      1     24  female
236  ./RAVDESS/Actor_24/03-01-07-01-01-02-24.wav      1     24  female
237  ./RAVDESS/Actor_24/03-01-04-02-02-02-24.wav      1     24  female
238  ./RAVDESS/Actor_24/03-01-06-01-02-01-24.wav      1     24  female
239  ./RAVDESS/Actor_24/03-01-02-02-02-01-24.wav      1     24  female

    intensity statement repetition emotion           label
0           0          0          0       4        male_sad
```

```
1            1         1          0       5      male_angry
2            1         0          1       6    male_fearful
3            1         0          1       5      male_angry
4            0         1          0       5      male_angry
..         ...       ...        ...     ...             ...
235          0         1          1       5    female_angry
236          0         0          1       7  female_disgust
237          1         1          1       4      female_sad
238          0         1          0       6  female_fearful
239          1         1          0       2     female_calm

[240 rows x 9 columns]
```

[0]:

```python
data2_df = data2_df[data2_df.actor != 21].reset_index(drop=True)
data2_df = data2_df[data2_df.actor != 22].reset_index(drop=True)
data2_df = data2_df[data2_df.actor != 23].reset_index(drop=True)
data2_df = data2_df[data2_df.actor != 24].reset_index(drop=True)
```

[0]:

```python
data2_df
```

[0]:

```
                                              path source actor  gender  \
0     ./RAVDESS/Actor_01/03-01-05-01-01-01-01.wav      1     1    male
1     ./RAVDESS/Actor_01/03-01-07-01-02-01-01.wav      1     1    male
2     ./RAVDESS/Actor_01/03-01-07-01-01-01-01.wav      1     1    male
3     ./RAVDESS/Actor_01/03-01-02-01-02-02-01.wav      1     1    male
4     ./RAVDESS/Actor_01/03-01-03-01-01-02-01.wav      1     1    male
...                                            ...    ...   ...     ...
1195  ./RAVDESS/Actor_20/03-01-02-02-02-01-20.wav      1    20  female
1196  ./RAVDESS/Actor_20/03-01-08-02-01-01-20.wav      1    20  female
1197  ./RAVDESS/Actor_20/03-01-02-01-01-02-20.wav      1    20  female
1198  ./RAVDESS/Actor_20/03-01-04-02-02-01-20.wav      1    20  female
1199  ./RAVDESS/Actor_20/03-01-06-01-02-01-20.wav      1    20  female

     intensity statement repetition emotion             label
0            0         0          0       5        male_angry
1            0         1          0       7      male_disgust
2            0         0          0       7      male_disgust
3            0         1          1       2         male_calm
4            0         0          1       3        male_happy
...        ...       ...        ...     ...               ...
1195         1         1          0       2       female_calm
1196         1         0          0       8  female_surprised
1197         0         0          1       2       female_calm
1198         1         1          0       4        female_sad
1199         0         1          0       6    female_fearful

[1200 rows x 9 columns]
```

[0]:
```
input_duration = 3
data = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i],␣
 ↪res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
    # X = X[10000:90000]
    sample_rate = np.array(sample_rate)
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),␣
 ↪axis=0)
    feature = mfccs
    data.loc[i] = [feature]

data.head()
```

```
100%|          | 1200/1200 [01:35<00:00, 12.51it/s]
```

[0]:
```
                                             feature
0  [-55.507362, -55.729572, -55.716793, -55.83580…
1  [-69.40937, -69.40937, -69.40937, -69.40937, -…
2  [-54.985146, -54.91457, -54.93782, -56.227646,…
3  [-69.0514, -69.0514, -69.0514, -69.0514, -69.0…
4  [-60.369045, -60.083717, -60.978924, -60.95245…
```

[0]:
```
df3 = pd.DataFrame(data['feature'].values.tolist())
labels = data2_df.label
```

[0]:
```
df3.head()
```

[0]:
```
         0          1          2          3          4          5    \
0 -55.507362 -55.729572 -55.716793 -55.835808 -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370
2 -54.985146 -54.914570 -54.937820 -56.227646 -56.685261 -57.022507
3 -69.051399 -69.051399 -69.051399 -69.051399 -69.051399 -68.754860
4 -60.369045 -60.083717 -60.978924 -60.952457 -60.982483 -60.983948

         6          7          8          9    …        249        250  \
0 -55.932133 -55.932133 -55.932133 -55.932133  … -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370  … -63.676579 -57.841850
2 -58.089943 -58.376122 -58.420403 -56.623604  … -63.795944 -64.638062
3 -69.051399 -69.051399 -69.051399 -68.359085  … -65.446953 -68.552094
4 -60.981255 -60.981255 -60.981255 -60.249615  … -60.981255 -60.981255

         251        252        253        254        255        256  \
0 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133
1 -48.709694 -44.560093 -44.730862 -51.467548 -53.909016 -47.980164
2 -65.028267 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340
3 -69.051399 -69.051399 -69.051399 -68.688614 -69.051399        NaN
```

```
4 -60.981255 -60.981255 -60.981255 -60.981255 -60.981255        NaN

         257        258
0 -55.932133 -55.932133
1 -43.389336 -43.327263
2 -65.215340 -65.215340
3        NaN        NaN
4        NaN        NaN

[5 rows x 259 columns]
```

[0]:
```python
labels.unique()
```

[0]:
```
array(['male_angry', 'male_disgust', 'male_calm', 'male_happy',
       'male_fearful', 'male_surprised', 'male_sad', 'male_neutral',
       'female_calm', 'female_happy', 'female_fearful', 'female_angry',
       'female_surprised', 'female_sad', 'female_disgust',
       'female_neutral'], dtype=object)
```

[0]:
```python
newdf = pd.concat([df3,labels], axis=1)
rnewdf = newdf.rename(index=str, columns={"0": "label"})
print(len(rnewdf))
rnewdf.head()
```

```
1200
```

[0]:
```
           0          1          2          3          4          5  \
0 -55.507362 -55.729572 -55.716793 -55.835808 -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370
2 -54.985146 -54.914570 -54.937820 -56.227646 -56.685261 -57.022507
3 -69.051399 -69.051399 -69.051399 -69.051399 -69.051399 -68.754860
4 -60.369045 -60.083717 -60.978924 -60.952457 -60.982483 -60.983948

           6          7          8          9  …        250        251  \
0 -55.932133 -55.932133 -55.932133 -55.932133  … -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370  … -57.841850 -48.709694
2 -58.089943 -58.376122 -58.420403 -56.623604  … -64.638062 -65.028267
3 -69.051399 -69.051399 -69.051399 -68.359085  … -68.552094 -69.051399
4 -60.981255 -60.981255 -60.981255 -60.249615  … -60.981255 -60.981255

         252        253        254        255        256        257  \
0 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133
1 -44.560093 -44.730862 -51.467548 -53.909016 -47.980164 -43.389336
2 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340
3 -69.051399 -69.051399 -68.688614 -69.051399        NaN        NaN
4 -60.981255 -60.981255 -60.981255 -60.981255        NaN        NaN
```

```
         258          label
0 -55.932133     male_angry
1 -43.327263   male_disgust
2 -65.215340   male_disgust
3        NaN      male_calm
4        NaN     male_happy

[5 rows x 260 columns]
```

[0]:
```
rnewdf.isnull().sum().sum()
```

[0]: 5104

[0]:
```
rnewdf = rnewdf.fillna(0)
rnewdf.head()
```

[0]:
```
          0          1          2          3          4          5  \
0 -55.507362 -55.729572 -55.716793 -55.835808 -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370
2 -54.985146 -54.914570 -54.937820 -56.227646 -56.685261 -57.022507
3 -69.051399 -69.051399 -69.051399 -69.051399 -69.051399 -68.754860
4 -60.369045 -60.083717 -60.978924 -60.952457 -60.982483 -60.983948

          6          7          8          9  …        250        251  \
0 -55.932133 -55.932133 -55.932133 -55.932133  … -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370  … -57.841850 -48.709694
2 -58.089943 -58.376122 -58.420403 -56.623604  … -64.638062 -65.028267
3 -69.051399 -69.051399 -69.051399 -68.359085  … -68.552094 -69.051399
4 -60.981255 -60.981255 -60.981255 -60.249615  … -60.981255 -60.981255

         252        253        254        255        256        257  \
0 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133
1 -44.560093 -44.730862 -51.467548 -53.909016 -47.980164 -43.389336
2 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340
3 -69.051399 -69.051399 -68.688614 -69.051399   0.000000   0.000000
4 -60.981255 -60.981255 -60.981255 -60.981255   0.000000   0.000000

         258          label
0 -55.932133     male_angry
1 -43.327263   male_disgust
2 -65.215340   male_disgust
3   0.000000      male_calm
4   0.000000     male_happy

[5 rows x 260 columns]
```

[0]:
```
rnewdf.isnull().sum().sum()
```

[0]: 0

6.0.3 Creating audio files with augmentation methods

[0]:
```
# Augmentation Method 1
import random
syn_data1 = pd.DataFrame(columns=['feature', 'label'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i],␣
 ↪res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
    if data2_df.label[i]:
#     if data2_df.label[i] == "male_positive":
        X = noise(X)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),␣
 ↪axis=0)
        feature = mfccs
        a = random.uniform(0, 1)
        syn_data1.loc[i] = [feature, data2_df.label[i]]
```

100%| | 1200/1200 [01:19<00:00, 15.10it/s]

[0]:
```
# Augmentation Method 2

syn_data2 = pd.DataFrame(columns=['feature', 'label'])
for i in tqdm(range(len(data2_df))):
    X, sample_rate = librosa.load(data2_df.path[i],␣
 ↪res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
    if data2_df.label[i]:
#     if data2_df.label[i] == "male_positive":
        X = pitch(X, sample_rate)
        sample_rate = np.array(sample_rate)
        mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),␣
 ↪axis=0)
        feature = mfccs
        a = random.uniform(0, 1)
        syn_data2.loc[i] = [feature, data2_df.label[i]]
```

100%| | 1200/1200 [04:39<00:00, 4.29it/s]

[0]:
```
len(syn_data1), len(syn_data2)
```

[0]: (1200, 1200)

[0]:
```
syn_data1 = syn_data1.reset_index(drop=True)
syn_data2 = syn_data2.reset_index(drop=True)
```

[0]:
```
df4 = pd.DataFrame(syn_data1['feature'].values.tolist())
labels4 = syn_data1.label
syndf1 = pd.concat([df4,labels4], axis=1)
syndf1 = syndf1.rename(index=str, columns={"0": "label"})
syndf1 = syndf1.fillna(0)
len(syndf1)
```

[0]: 1200

[0]:
```
```

[0]:
```
df4 = pd.DataFrame(syn_data2['feature'].values.tolist())
labels4 = syn_data2.label
syndf2 = pd.concat([df4,labels4], axis=1)
syndf2 = syndf2.rename(index=str, columns={"0": "label"})
syndf2 = syndf2.fillna(0)
print(len(syndf2))
syndf2.head()
```

```
1200
```

[0]:
```
           0          1          2          3          4          5  \
0 -57.081936 -57.812729 -57.975456 -58.220848 -58.200806 -58.320297
1 -71.871422 -71.871422 -71.871422 -71.871422 -71.871422 -71.871422
2 -55.857418 -56.224628 -57.640255 -58.666267 -59.475227 -59.460384
3 -70.618378 -70.618378 -70.618378 -70.618378 -70.618378 -70.618378
4 -62.943687 -62.951511 -64.301682 -64.608696 -64.551857 -64.523506

           6          7          8          9  …        250        251  \
0 -58.320297 -58.320297 -58.320297 -58.320297  … -58.320297 -58.320297
1 -71.871422 -71.871422 -71.871422 -71.871422  … -54.810707 -50.857159
2 -59.371342 -60.084358 -60.724937 -59.392990  … -67.250877 -67.609222
3 -70.618378 -70.618378 -70.618378 -70.618378  … -69.138916 -70.618378
4 -64.523506 -64.523506 -64.523506 -62.884731  … -64.523506 -64.523506

         252        253        254        255        256        257  \
0 -58.320297 -58.320297 -58.320297 -58.320297 -58.320297 -58.320297
1 -50.030987 -49.503311 -51.263157 -52.093075 -49.458767 -50.452084
2 -67.609222 -67.609222 -67.609222 -67.609222 -67.609222 -67.609222
3 -70.618378 -70.618378 -70.618378 -70.618378   0.000000   0.000000
4 -64.523506 -64.523506 -64.523506 -64.523506   0.000000   0.000000

         258         label
0 -58.320297    male_angry
1 -50.590515  male_disgust
2 -67.609222  male_disgust
3   0.000000     male_calm
```

```
4   0.000000    male_happy

[5 rows x 260 columns]
```

[0]:

```

```

[0]:

```python
# Combining the Augmented data with original
combined_df = pd.concat([rnewdf, syndf1, syndf2], ignore_index=True)
combined_df = combined_df.fillna(0)
combined_df.head()
```

[0]:

```
          0          1          2          3          4          5  \
0 -55.507362 -55.729572 -55.716793 -55.835808 -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370 -69.409370
2 -54.985146 -54.914570 -54.937820 -56.227646 -56.685261 -57.022507
3 -69.051399 -69.051399 -69.051399 -69.051399 -69.051399 -68.754860
4 -60.369045 -60.083717 -60.978924 -60.952457 -60.982483 -60.983948

          6          7          8          9  …        250        251  \
0 -55.932133 -55.932133 -55.932133 -55.932133  … -55.932133 -55.932133
1 -69.409370 -69.409370 -69.409370 -69.409370  … -57.841850 -48.709694
2 -58.089943 -58.376122 -58.420403 -56.623604  … -64.638062 -65.028267
3 -69.051399 -69.051399 -69.051399 -68.359085  … -68.552094 -69.051399
4 -60.981255 -60.981255 -60.981255 -60.249615  … -60.981255 -60.981255

        252        253        254        255        256        257  \
0 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133 -55.932133
1 -44.560093 -44.730862 -51.467548 -53.909016 -47.980164 -43.389336
2 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340 -65.215340
3 -69.051399 -69.051399 -68.688614 -69.051399   0.000000   0.000000
4 -60.981255 -60.981255 -60.981255 -60.981255   0.000000   0.000000

        258        label
0 -55.932133   male_angry
1 -43.327263 male_disgust
2 -65.215340 male_disgust
3   0.000000    male_calm
4   0.000000   male_happy

[5 rows x 260 columns]
```

[0]:

```python
len(combined_df)
```

[0]: 3600

[0]:

```python
from sklearn.model_selection import StratifiedShuffleSplit
#  Stratified Shuffle Split
```

```python
X = combined_df.drop(['label'], axis=1)
y = combined_df.label
xxx = StratifiedShuffleSplit(1, test_size=0.2, random_state=12)
for train_index, test_index in xxx.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

[0]:
```python
print(y_train.value_counts())
```

```
female_sad          192
female_calm         192
male_disgust        192
male_fearful        192
female_fearful      192
male_surprised      192
female_angry        192
female_surprised    192
male_angry          192
female_happy        192
male_happy          192
male_calm           192
female_disgust      192
male_sad            192
male_neutral         96
female_neutral       96
Name: label, dtype: int64
```

[0]:
```python
print(y_test.value_counts())
```

```
female_sad          48
female_calm         48
male_happy          48
female_surprised    48
male_angry          48
male_calm           48
female_happy        48
female_fearful      48
male_disgust        48
male_fearful        48
male_surprised      48
female_disgust      48
male_sad            48
female_angry        48
female_neutral      24
male_neutral        24
Name: label, dtype: int64
```

[0]:
```
X_train.isna().sum().sum()
```

[0]: 0

[0]:
```
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))
```

[0]:
```
X_train.shape, y_train.shape
```

[0]: ((2880, 259), (2880, 16))

[0]:
```
X_test.shape, y_test.shape
```

[0]: ((720, 259), (720, 16))

[0]:
```
x_traincnn = np.expand_dims(X_train, axis=2)
x_testcnn = np.expand_dims(X_test, axis=2)
```

[0]:
```
x_traincnn.shape, x_testcnn.shape
```

[0]: ((2880, 259, 1), (720, 259, 1))

[0]:
```
# Set up Keras util functions

from keras import backend as K

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def fscore(y_true, y_pred):
    if K.sum(K.round(K.clip(y_true, 0, 1))) == 0:
```

```
        return 0

    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    f_score = 2 * (p * r) / (p + r + K.epsilon())
    return f_score

def get_lr_metric(optimizer):
    def lr(y_true, y_pred):
        return optimizer.lr
    return lr
```

[0]:
```
model = Sequential()
model.add(Conv1D(256, 8, padding='same',input_shape=(X_train.shape[1],1)))
model.add(Activation('relu'))
model.add(Conv1D(256, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(128, 8, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(MaxPooling1D(pool_size=(8)))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Conv1D(64, 8, padding='same'))
model.add(Activation('relu'))
model.add(Flatten())
# Edit according to target class no.
model.add(Dense(16))
model.add(Activation('softmax'))
opt = keras.optimizers.SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_73 (Conv1D)           (None, 259, 256)          2304
```

```
_________________________________________________________________
activation_90 (Activation)   (None, 259, 256)          0
_________________________________________________________________
conv1d_74 (Conv1D)           (None, 259, 256)          524544
_________________________________________________________________
batch_normalization_13 (Batc (None, 259, 256)          1024
_________________________________________________________________
activation_91 (Activation)   (None, 259, 256)          0
_________________________________________________________________
dropout_26 (Dropout)         (None, 259, 256)          0
_________________________________________________________________
max_pooling1d_17 (MaxPooling (None, 32, 256)           0
_________________________________________________________________
conv1d_75 (Conv1D)           (None, 32, 128)           262272
_________________________________________________________________
activation_92 (Activation)   (None, 32, 128)           0
_________________________________________________________________
conv1d_76 (Conv1D)           (None, 32, 128)           131200
_________________________________________________________________
activation_93 (Activation)   (None, 32, 128)           0
_________________________________________________________________
conv1d_77 (Conv1D)           (None, 32, 128)           131200
_________________________________________________________________
activation_94 (Activation)   (None, 32, 128)           0
_________________________________________________________________
conv1d_78 (Conv1D)           (None, 32, 128)           131200
_________________________________________________________________
batch_normalization_14 (Batc (None, 32, 128)           512
_________________________________________________________________
activation_95 (Activation)   (None, 32, 128)           0
_________________________________________________________________
dropout_27 (Dropout)         (None, 32, 128)           0
_________________________________________________________________
max_pooling1d_18 (MaxPooling (None, 4, 128)            0
_________________________________________________________________
conv1d_79 (Conv1D)           (None, 4, 64)             65600
_________________________________________________________________
activation_96 (Activation)   (None, 4, 64)             0
_________________________________________________________________
conv1d_80 (Conv1D)           (None, 4, 64)             32832
_________________________________________________________________
activation_97 (Activation)   (None, 4, 64)             0
_________________________________________________________________
flatten_11 (Flatten)         (None, 256)               0
_________________________________________________________________
dense_18 (Dense)             (None, 16)                4112
_________________________________________________________________
activation_98 (Activation)   (None, 16)                0
```

```
=================================================================
Total params: 1,286,800
Trainable params: 1,286,032
Non-trainable params: 768
_________________________________________________________________
```

[0]:

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

[0]:

```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler,␣
↪EarlyStopping
from keras.callbacks import  History, ReduceLROnPlateau, CSVLogger
# Model Training

lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.9, patience=20,␣
↪min_lr=0.000001)

# Please change the model name accordingly.
mcp_save = ModelCheckpoint('./saved_models/Experiment_5_model.h5',␣
↪save_best_only=True, monitor='val_loss', mode='min')
cnnhistory=model.fit(x_traincnn, y_train, batch_size=16, epochs=700,
                     validation_data=(x_testcnn, y_test), callbacks=[mcp_save,␣
↪lr_reduce])
```

```
Train on 2880 samples, validate on 720 samples
Epoch 1/700
2880/2880 [==============================] - 25s 9ms/step - loss: 2.7136 -
accuracy: 0.1139 - val_loss: 2.7111 - val_accuracy: 0.1444
Epoch 2/700
2880/2880 [==============================] - 23s 8ms/step - loss: 2.6600 -
accuracy: 0.1337 - val_loss: 2.6630 - val_accuracy: 0.1569
Epoch 3/700
2880/2880 [==============================] - 23s 8ms/step - loss: 2.6176 -
accuracy: 0.1660 - val_loss: 2.6354 - val_accuracy: 0.1778
Epoch 4/700
2880/2880 [==============================] - 26s 9ms/step - loss: 2.5890 -
accuracy: 0.1806 - val_loss: 2.6164 - val_accuracy: 0.1833
Epoch 5/700
2880/2880 [==============================] - 31s 11ms/step - loss: 2.5614 -
accuracy: 0.1726 - val_loss: 2.6076 - val_accuracy: 0.1875
Epoch 6/700
2880/2880 [==============================] - 25s 9ms/step - loss: 2.5404 -
accuracy: 0.1889 - val_loss: 2.5787 - val_accuracy: 0.1806
Epoch 7/700
2880/2880 [==============================] - 25s 9ms/step - loss: 2.5143 -
accuracy: 0.1979 - val_loss: 2.5653 - val_accuracy: 0.2014
Epoch 8/700
```

```
2880/2880 [==============================] - 25s 9ms/step - loss: 2.4936 -
accuracy: 0.2052 - val_loss: 2.5527 - val_accuracy: 0.2097
Epoch 9/700
2880/2880 [==============================] - 26s 9ms/step - loss: 2.4770 -
accuracy: 0.2191 - val_loss: 2.5435 - val_accuracy: 0.2125
Epoch 10/700
2880/2880 [==============================] - 33s 11ms/step - loss: 2.4663 -
accuracy: 0.2118 - val_loss: 2.5203 - val_accuracy: 0.2153
Epoch 11/700
2880/2880 [==============================] - 32s 11ms/step - loss: 2.4413 -
accuracy: 0.2222 - val_loss: 2.5172 - val_accuracy: 0.2208
Epoch 12/700
2880/2880 [==============================] - 29s 10ms/step - loss: 2.4250 -
accuracy: 0.2215 - val_loss: 2.5107 - val_accuracy: 0.2083
Epoch 13/700
2880/2880 [==============================] - 25s 9ms/step - loss: 2.4043 -
accuracy: 0.2330 - val_loss: 2.4951 - val_accuracy: 0.2458
Epoch 14/700
2880/2880 [==============================] - 26s 9ms/step - loss: 2.3957 -
accuracy: 0.2313 - val_loss: 2.4770 - val_accuracy: 0.2306
Epoch 15/700
2880/2880 [==============================] - 28s 10ms/step - loss: 2.3691 -
accuracy: 0.2458 - val_loss: 2.4747 - val_accuracy: 0.2403
Epoch 16/700
2880/2880 [==============================] - 32s 11ms/step - loss: 2.3627 -
accuracy: 0.2476 - val_loss: 2.4636 - val_accuracy: 0.2417
Epoch 17/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.3459 -
accuracy: 0.2458 - val_loss: 2.4525 - val_accuracy: 0.2347
Epoch 18/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.3233 -
accuracy: 0.2576 - val_loss: 2.4413 - val_accuracy: 0.2403
Epoch 19/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.3157 -
accuracy: 0.2663 - val_loss: 2.4228 - val_accuracy: 0.2514
Epoch 20/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.3021 -
accuracy: 0.2535 - val_loss: 2.4074 - val_accuracy: 0.2458
Epoch 21/700
2880/2880 [==============================] - 36s 12ms/step - loss: 2.2879 -
accuracy: 0.2622 - val_loss: 2.4178 - val_accuracy: 0.2292
Epoch 22/700
2880/2880 [==============================] - 33s 12ms/step - loss: 2.2812 -
accuracy: 0.2632 - val_loss: 2.3952 - val_accuracy: 0.2625
Epoch 23/700
2880/2880 [==============================] - 33s 11ms/step - loss: 2.2689 -
accuracy: 0.2653 - val_loss: 2.3940 - val_accuracy: 0.2750
Epoch 24/700
```

```
2880/2880 [==============================] - 32s 11ms/step - loss: 2.2448 -
accuracy: 0.2788 - val_loss: 2.3723 - val_accuracy: 0.2708
Epoch 25/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.2386 -
accuracy: 0.2743 - val_loss: 2.3611 - val_accuracy: 0.2694
Epoch 26/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.2212 -
accuracy: 0.2847 - val_loss: 2.3725 - val_accuracy: 0.2514
Epoch 27/700
2880/2880 [==============================] - 30s 10ms/step - loss: 2.2086 -
accuracy: 0.2861 - val_loss: 2.3449 - val_accuracy: 0.2847
Epoch 28/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.1979 -
accuracy: 0.2819 - val_loss: 2.3691 - val_accuracy: 0.2556
Epoch 29/700
2880/2880 [==============================] - 36s 12ms/step - loss: 2.1853 -
accuracy: 0.2889 - val_loss: 2.3338 - val_accuracy: 0.2889
Epoch 30/700
2880/2880 [==============================] - 33s 11ms/step - loss: 2.1670 -
accuracy: 0.3017 - val_loss: 2.2985 - val_accuracy: 0.2806
Epoch 31/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.1587 -
accuracy: 0.2941 - val_loss: 2.3149 - val_accuracy: 0.2931
Epoch 32/700
2880/2880 [==============================] - 36s 12ms/step - loss: 2.1542 -
accuracy: 0.3010 - val_loss: 2.2934 - val_accuracy: 0.2931
Epoch 33/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.1414 -
accuracy: 0.2986 - val_loss: 2.2940 - val_accuracy: 0.2986
Epoch 34/700
2880/2880 [==============================] - 33s 11ms/step - loss: 2.1163 -
accuracy: 0.3184 - val_loss: 2.3043 - val_accuracy: 0.2833
Epoch 35/700
2880/2880 [==============================] - 36s 12ms/step - loss: 2.1081 -
accuracy: 0.3045 - val_loss: 2.2633 - val_accuracy: 0.3250
Epoch 36/700
2880/2880 [==============================] - 32s 11ms/step - loss: 2.0986 -
accuracy: 0.3149 - val_loss: 2.2902 - val_accuracy: 0.2903
Epoch 37/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.0941 -
accuracy: 0.3153 - val_loss: 2.2518 - val_accuracy: 0.3236
Epoch 38/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.0716 -
accuracy: 0.3361 - val_loss: 2.2203 - val_accuracy: 0.3361
Epoch 39/700
2880/2880 [==============================] - 36s 12ms/step - loss: 2.0606 -
accuracy: 0.3403 - val_loss: 2.2319 - val_accuracy: 0.3278
Epoch 40/700
```

```
2880/2880 [==============================] - 35s 12ms/step - loss: 2.0523 -
accuracy: 0.3372 - val_loss: 2.2348 - val_accuracy: 0.3069
Epoch 41/700
2880/2880 [==============================] - 34s 12ms/step - loss: 2.0375 -
accuracy: 0.3444 - val_loss: 2.2071 - val_accuracy: 0.3389
Epoch 42/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.0351 -
accuracy: 0.3288 - val_loss: 2.2084 - val_accuracy: 0.3042
Epoch 43/700
2880/2880 [==============================] - 36s 13ms/step - loss: 2.0284 -
accuracy: 0.3274 - val_loss: 2.2672 - val_accuracy: 0.2681
Epoch 44/700
2880/2880 [==============================] - 35s 12ms/step - loss: 2.0013 -
accuracy: 0.3396 - val_loss: 2.2015 - val_accuracy: 0.3181
Epoch 45/700
2880/2880 [==============================] - 33s 11ms/step - loss: 2.0029 -
accuracy: 0.3486 - val_loss: 2.1653 - val_accuracy: 0.3569
Epoch 46/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9929 -
accuracy: 0.3375 - val_loss: 2.1974 - val_accuracy: 0.3222
Epoch 47/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9820 -
accuracy: 0.3444 - val_loss: 2.2011 - val_accuracy: 0.3139
Epoch 48/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.9775 -
accuracy: 0.3497 - val_loss: 2.1374 - val_accuracy: 0.3375
Epoch 49/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.9573 -
accuracy: 0.3580 - val_loss: 2.1696 - val_accuracy: 0.3375
Epoch 50/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.9495 -
accuracy: 0.3622 - val_loss: 2.1462 - val_accuracy: 0.3417
Epoch 51/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9491 -
accuracy: 0.3587 - val_loss: 2.1208 - val_accuracy: 0.3431
Epoch 52/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9231 -
accuracy: 0.3712 - val_loss: 2.1739 - val_accuracy: 0.3208
Epoch 53/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9211 -
accuracy: 0.3649 - val_loss: 2.1833 - val_accuracy: 0.3083
Epoch 54/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.9126 -
accuracy: 0.3715 - val_loss: 2.1314 - val_accuracy: 0.3444
Epoch 55/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.9049 -
accuracy: 0.3823 - val_loss: 2.0746 - val_accuracy: 0.3653
Epoch 56/700
```

```
2880/2880 [==============================] - 34s 12ms/step - loss: 1.8979 -
accuracy: 0.3719 - val_loss: 2.1282 - val_accuracy: 0.3403
Epoch 57/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.8900 -
accuracy: 0.3802 - val_loss: 2.1105 - val_accuracy: 0.3361
Epoch 58/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.8842 -
accuracy: 0.3722 - val_loss: 2.0915 - val_accuracy: 0.3403
Epoch 59/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.8603 -
accuracy: 0.3767 - val_loss: 2.1440 - val_accuracy: 0.3194
Epoch 60/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.8604 -
accuracy: 0.3910 - val_loss: 2.1024 - val_accuracy: 0.3597
Epoch 61/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.8567 -
accuracy: 0.3927 - val_loss: 2.0668 - val_accuracy: 0.3681
Epoch 62/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.8436 -
accuracy: 0.3875 - val_loss: 2.0465 - val_accuracy: 0.3750
Epoch 63/700
2880/2880 [==============================] - 33s 11ms/step - loss: 1.8345 -
accuracy: 0.3924 - val_loss: 2.0584 - val_accuracy: 0.3569
Epoch 64/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.8279 -
accuracy: 0.3944 - val_loss: 2.0427 - val_accuracy: 0.3681
Epoch 65/700
2880/2880 [==============================] - 33s 12ms/step - loss: 1.8174 -
accuracy: 0.3969 - val_loss: 2.1930 - val_accuracy: 0.2889
Epoch 66/700
2880/2880 [==============================] - 32s 11ms/step - loss: 1.8117 -
accuracy: 0.4056 - val_loss: 2.0018 - val_accuracy: 0.3764
Epoch 67/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.8048 -
accuracy: 0.4069 - val_loss: 2.0149 - val_accuracy: 0.3833
Epoch 68/700
2880/2880 [==============================] - 40s 14ms/step - loss: 1.7970 -
accuracy: 0.3983 - val_loss: 2.2376 - val_accuracy: 0.2486
Epoch 69/700
2880/2880 [==============================] - 30s 10ms/step - loss: 1.7950 -
accuracy: 0.4083 - val_loss: 2.0364 - val_accuracy: 0.3597
Epoch 70/700
2880/2880 [==============================] - 25s 9ms/step - loss: 1.7758 -
accuracy: 0.4139 - val_loss: 1.9857 - val_accuracy: 0.3903
Epoch 71/700
2880/2880 [==============================] - 28s 10ms/step - loss: 1.7619 -
accuracy: 0.4215 - val_loss: 2.1240 - val_accuracy: 0.2972
Epoch 72/700
```

```
2880/2880 [==============================] - 28s 10ms/step - loss: 1.7694 -
accuracy: 0.4128 - val_loss: 2.0314 - val_accuracy: 0.3653
Epoch 73/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.7630 -
accuracy: 0.4267 - val_loss: 1.9800 - val_accuracy: 0.3819
Epoch 74/700
2880/2880 [==============================] - 28s 10ms/step - loss: 1.7508 -
accuracy: 0.4229 - val_loss: 2.0387 - val_accuracy: 0.3486
Epoch 75/700
2880/2880 [==============================] - 30s 11ms/step - loss: 1.7442 -
accuracy: 0.4313 - val_loss: 1.9533 - val_accuracy: 0.3958
Epoch 76/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.7312 -
accuracy: 0.4340 - val_loss: 1.9950 - val_accuracy: 0.3750
Epoch 77/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.7330 -
accuracy: 0.4191 - val_loss: 2.1039 - val_accuracy: 0.2931
Epoch 78/700
2880/2880 [==============================] - 31s 11ms/step - loss: 1.7115 -
accuracy: 0.4375 - val_loss: 2.0335 - val_accuracy: 0.3500
Epoch 79/700
2880/2880 [==============================] - 27s 10ms/step - loss: 1.7187 -
accuracy: 0.4316 - val_loss: 1.9590 - val_accuracy: 0.3917
Epoch 80/700
2880/2880 [==============================] - 27s 10ms/step - loss: 1.7156 -
accuracy: 0.4389 - val_loss: 1.9517 - val_accuracy: 0.4097
Epoch 81/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.6952 -
accuracy: 0.4455 - val_loss: 1.9594 - val_accuracy: 0.3972
Epoch 82/700
2880/2880 [==============================] - 43s 15ms/step - loss: 1.6946 -
accuracy: 0.4368 - val_loss: 1.9981 - val_accuracy: 0.3472
Epoch 83/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.6911 -
accuracy: 0.4382 - val_loss: 1.9979 - val_accuracy: 0.3681
Epoch 84/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.6794 -
accuracy: 0.4437 - val_loss: 1.9426 - val_accuracy: 0.3917
Epoch 85/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.6793 -
accuracy: 0.4410 - val_loss: 1.9018 - val_accuracy: 0.4125
Epoch 86/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.6508 -
accuracy: 0.4601 - val_loss: 1.9747 - val_accuracy: 0.3764
Epoch 87/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.6621 -
accuracy: 0.4587 - val_loss: 1.9241 - val_accuracy: 0.3917
Epoch 88/700
```

```
2880/2880 [==============================] - 36s 13ms/step - loss: 1.6520 -
accuracy: 0.4559 - val_loss: 1.9187 - val_accuracy: 0.4208
Epoch 89/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.6489 -
accuracy: 0.4573 - val_loss: 1.9961 - val_accuracy: 0.3528
Epoch 90/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.6301 -
accuracy: 0.4660 - val_loss: 1.8996 - val_accuracy: 0.4083
Epoch 91/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.6175 -
accuracy: 0.4764 - val_loss: 1.9251 - val_accuracy: 0.3875
Epoch 92/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.6271 -
accuracy: 0.4726 - val_loss: 1.8742 - val_accuracy: 0.4292
Epoch 93/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.6174 -
accuracy: 0.4628 - val_loss: 1.8849 - val_accuracy: 0.4250
Epoch 94/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.6136 -
accuracy: 0.4701 - val_loss: 1.9057 - val_accuracy: 0.3917
Epoch 95/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.6100 -
accuracy: 0.4708 - val_loss: 2.0864 - val_accuracy: 0.3056
Epoch 96/700
2880/2880 [==============================] - 42s 15ms/step - loss: 1.6014 -
accuracy: 0.4774 - val_loss: 1.8518 - val_accuracy: 0.4472
Epoch 97/700
2880/2880 [==============================] - 31s 11ms/step - loss: 1.5994 -
accuracy: 0.4826 - val_loss: 1.8618 - val_accuracy: 0.4111
Epoch 98/700
2880/2880 [==============================] - 28s 10ms/step - loss: 1.5812 -
accuracy: 0.4861 - val_loss: 1.8894 - val_accuracy: 0.4014
Epoch 99/700
2880/2880 [==============================] - 30s 10ms/step - loss: 1.5730 -
accuracy: 0.4899 - val_loss: 1.8999 - val_accuracy: 0.4069
Epoch 100/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5724 -
accuracy: 0.4858 - val_loss: 1.9033 - val_accuracy: 0.3972
Epoch 101/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5654 -
accuracy: 0.4976 - val_loss: 2.1013 - val_accuracy: 0.3028
Epoch 102/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5588 -
accuracy: 0.4851 - val_loss: 1.9787 - val_accuracy: 0.3694
Epoch 103/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5511 -
accuracy: 0.4962 - val_loss: 2.0769 - val_accuracy: 0.2986
Epoch 104/700
```

```
2880/2880 [==============================] - 30s 10ms/step - loss: 1.5463 -
accuracy: 0.4990 - val_loss: 2.2994 - val_accuracy: 0.2278
Epoch 105/700
2880/2880 [==============================] - 30s 11ms/step - loss: 1.5424 -
accuracy: 0.4885 - val_loss: 2.0319 - val_accuracy: 0.3278
Epoch 106/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5279 -
accuracy: 0.5031 - val_loss: 1.9336 - val_accuracy: 0.3736
Epoch 107/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5382 -
accuracy: 0.5115 - val_loss: 1.8815 - val_accuracy: 0.4028
Epoch 108/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5300 -
accuracy: 0.5094 - val_loss: 2.1509 - val_accuracy: 0.2736
Epoch 109/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5151 -
accuracy: 0.5066 - val_loss: 1.8458 - val_accuracy: 0.4278
Epoch 110/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5024 -
accuracy: 0.5118 - val_loss: 1.8086 - val_accuracy: 0.4153
Epoch 111/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.5048 -
accuracy: 0.5167 - val_loss: 1.9037 - val_accuracy: 0.3597
Epoch 112/700
2880/2880 [==============================] - 30s 10ms/step - loss: 1.4906 -
accuracy: 0.5236 - val_loss: 1.9635 - val_accuracy: 0.3569
Epoch 113/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.4861 -
accuracy: 0.5170 - val_loss: 1.9145 - val_accuracy: 0.3764
Epoch 114/700
2880/2880 [==============================] - 30s 10ms/step - loss: 1.4714 -
accuracy: 0.5319 - val_loss: 1.7855 - val_accuracy: 0.4431
Epoch 115/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.4649 -
accuracy: 0.5392 - val_loss: 1.8264 - val_accuracy: 0.4194
Epoch 116/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.4639 -
accuracy: 0.5257 - val_loss: 1.8094 - val_accuracy: 0.4319
Epoch 117/700
2880/2880 [==============================] - 40s 14ms/step - loss: 1.4654 -
accuracy: 0.5264 - val_loss: 1.8980 - val_accuracy: 0.3875
Epoch 118/700
2880/2880 [==============================] - 42s 15ms/step - loss: 1.4577 -
accuracy: 0.5333 - val_loss: 1.8321 - val_accuracy: 0.4167
Epoch 119/700
2880/2880 [==============================] - 28s 10ms/step - loss: 1.4546 -
accuracy: 0.5333 - val_loss: 1.7723 - val_accuracy: 0.4417
Epoch 120/700
```

```
2880/2880 [==============================] - 28s 10ms/step - loss: 1.4475 -
accuracy: 0.5260 - val_loss: 1.7717 - val_accuracy: 0.4486
Epoch 121/700
2880/2880 [==============================] - 48s 17ms/step - loss: 1.4280 -
accuracy: 0.5330 - val_loss: 1.8748 - val_accuracy: 0.4042
Epoch 122/700
2880/2880 [==============================] - 43s 15ms/step - loss: 1.4396 -
accuracy: 0.5285 - val_loss: 1.8920 - val_accuracy: 0.3722
Epoch 123/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.4237 -
accuracy: 0.5514 - val_loss: 1.8689 - val_accuracy: 0.4139
Epoch 124/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.4104 -
accuracy: 0.5483 - val_loss: 1.8309 - val_accuracy: 0.4194
Epoch 125/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.4211 -
accuracy: 0.5472 - val_loss: 1.9210 - val_accuracy: 0.3500
Epoch 126/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.4128 -
accuracy: 0.5469 - val_loss: 1.7593 - val_accuracy: 0.4403
Epoch 127/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.4038 -
accuracy: 0.5524 - val_loss: 1.7416 - val_accuracy: 0.4514
Epoch 128/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.3942 -
accuracy: 0.5535 - val_loss: 1.7512 - val_accuracy: 0.4556
Epoch 129/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3899 -
accuracy: 0.5639 - val_loss: 1.7886 - val_accuracy: 0.4361
Epoch 130/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.3798 -
accuracy: 0.5615 - val_loss: 1.8858 - val_accuracy: 0.3792
Epoch 131/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.3784 -
accuracy: 0.5622 - val_loss: 1.7978 - val_accuracy: 0.4236
Epoch 132/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.3877 -
accuracy: 0.5562 - val_loss: 1.7387 - val_accuracy: 0.4486
Epoch 133/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.3619 -
accuracy: 0.5677 - val_loss: 1.7844 - val_accuracy: 0.4042
Epoch 134/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.3561 -
accuracy: 0.5677 - val_loss: 1.7781 - val_accuracy: 0.4361
Epoch 135/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.3583 -
accuracy: 0.5736 - val_loss: 1.9704 - val_accuracy: 0.3375
Epoch 136/700
```

```
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3515 -
accuracy: 0.5729 - val_loss: 1.7252 - val_accuracy: 0.4514
Epoch 137/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3378 -
accuracy: 0.5736 - val_loss: 1.9713 - val_accuracy: 0.3444
Epoch 138/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3246 -
accuracy: 0.5823 - val_loss: 1.8815 - val_accuracy: 0.3792
Epoch 139/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3171 -
accuracy: 0.5896 - val_loss: 1.7496 - val_accuracy: 0.4403
Epoch 140/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3164 -
accuracy: 0.5872 - val_loss: 1.8396 - val_accuracy: 0.4069
Epoch 141/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3145 -
accuracy: 0.5872 - val_loss: 1.8634 - val_accuracy: 0.3931
Epoch 142/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.3153 -
accuracy: 0.5896 - val_loss: 1.6925 - val_accuracy: 0.4569
Epoch 143/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.3107 -
accuracy: 0.5830 - val_loss: 1.7167 - val_accuracy: 0.4514
Epoch 144/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2935 -
accuracy: 0.5955 - val_loss: 1.8569 - val_accuracy: 0.3722
Epoch 145/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2960 -
accuracy: 0.6021 - val_loss: 1.7075 - val_accuracy: 0.4764
Epoch 146/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.2833 -
accuracy: 0.6062 - val_loss: 1.7107 - val_accuracy: 0.4639
Epoch 147/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2850 -
accuracy: 0.6059 - val_loss: 1.8082 - val_accuracy: 0.4125
Epoch 148/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2836 -
accuracy: 0.5878 - val_loss: 1.7242 - val_accuracy: 0.4472
Epoch 149/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.2701 -
accuracy: 0.6066 - val_loss: 1.7791 - val_accuracy: 0.4139
Epoch 150/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.2615 -
accuracy: 0.6049 - val_loss: 1.6911 - val_accuracy: 0.4764
Epoch 151/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.2675 -
accuracy: 0.6014 - val_loss: 1.7548 - val_accuracy: 0.4250
Epoch 152/700
```

```
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2595 -
accuracy: 0.6118 - val_loss: 1.7157 - val_accuracy: 0.4375
Epoch 153/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.2512 -
accuracy: 0.6153 - val_loss: 1.7678 - val_accuracy: 0.4403
Epoch 154/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2446 -
accuracy: 0.6149 - val_loss: 1.6734 - val_accuracy: 0.4500
Epoch 155/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.2283 -
accuracy: 0.6292 - val_loss: 1.6778 - val_accuracy: 0.4694
Epoch 156/700
2880/2880 [==============================] - 33s 11ms/step - loss: 1.2437 -
accuracy: 0.6135 - val_loss: 1.6147 - val_accuracy: 0.5083
Epoch 157/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.2040 -
accuracy: 0.6441 - val_loss: 1.7890 - val_accuracy: 0.4153
Epoch 158/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.2226 -
accuracy: 0.6253 - val_loss: 2.0082 - val_accuracy: 0.3389
Epoch 159/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2095 -
accuracy: 0.6295 - val_loss: 1.6430 - val_accuracy: 0.4861
Epoch 160/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.1943 -
accuracy: 0.6344 - val_loss: 1.7616 - val_accuracy: 0.4292
Epoch 161/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2080 -
accuracy: 0.6205 - val_loss: 1.6906 - val_accuracy: 0.4639
Epoch 162/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.2058 -
accuracy: 0.6316 - val_loss: 1.7007 - val_accuracy: 0.4625
Epoch 163/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.1860 -
accuracy: 0.6347 - val_loss: 1.7022 - val_accuracy: 0.4431
Epoch 164/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1822 -
accuracy: 0.6389 - val_loss: 1.6807 - val_accuracy: 0.4597
Epoch 165/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1768 -
accuracy: 0.6389 - val_loss: 1.6134 - val_accuracy: 0.5042
Epoch 166/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1860 -
accuracy: 0.6403 - val_loss: 1.7204 - val_accuracy: 0.4556
Epoch 167/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.1720 -
accuracy: 0.6326 - val_loss: 1.6713 - val_accuracy: 0.4750
Epoch 168/700
```

```
2880/2880 [==============================] - 36s 13ms/step - loss: 1.1632 -
accuracy: 0.6521 - val_loss: 1.7640 - val_accuracy: 0.4306
Epoch 169/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1484 -
accuracy: 0.6517 - val_loss: 1.6796 - val_accuracy: 0.4722
Epoch 170/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1522 -
accuracy: 0.6500 - val_loss: 1.7547 - val_accuracy: 0.4139
Epoch 171/700
2880/2880 [==============================] - 36s 12ms/step - loss: 1.1360 -
accuracy: 0.6514 - val_loss: 1.7128 - val_accuracy: 0.4389
Epoch 172/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1490 -
accuracy: 0.6444 - val_loss: 1.6014 - val_accuracy: 0.5083
Epoch 173/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1453 -
accuracy: 0.6628 - val_loss: 1.7265 - val_accuracy: 0.4597
Epoch 174/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1262 -
accuracy: 0.6622 - val_loss: 1.6729 - val_accuracy: 0.4403
Epoch 175/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1255 -
accuracy: 0.6618 - val_loss: 1.7419 - val_accuracy: 0.4278
Epoch 176/700
2880/2880 [==============================] - 33s 11ms/step - loss: 1.1166 -
accuracy: 0.6649 - val_loss: 1.6536 - val_accuracy: 0.4736
Epoch 177/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.1100 -
accuracy: 0.6788 - val_loss: 1.6900 - val_accuracy: 0.4333
Epoch 178/700
2880/2880 [==============================] - 32s 11ms/step - loss: 1.1051 -
accuracy: 0.6622 - val_loss: 1.6833 - val_accuracy: 0.4597
Epoch 179/700
2880/2880 [==============================] - 33s 12ms/step - loss: 1.1066 -
accuracy: 0.6750 - val_loss: 1.6641 - val_accuracy: 0.4681
Epoch 180/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.0964 -
accuracy: 0.6674 - val_loss: 1.6331 - val_accuracy: 0.4861
Epoch 181/700
2880/2880 [==============================] - 33s 12ms/step - loss: 1.0858 -
accuracy: 0.6691 - val_loss: 1.5772 - val_accuracy: 0.4986
Epoch 182/700
2880/2880 [==============================] - 33s 12ms/step - loss: 1.0863 -
accuracy: 0.6757 - val_loss: 1.6090 - val_accuracy: 0.5042
Epoch 183/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.0798 -
accuracy: 0.6771 - val_loss: 1.6395 - val_accuracy: 0.4653
Epoch 184/700
```

```
2880/2880 [==============================] - 34s 12ms/step - loss: 1.0781 -
accuracy: 0.6719 - val_loss: 1.7222 - val_accuracy: 0.4403
Epoch 185/700
2880/2880 [==============================] - 38s 13ms/step - loss: 1.0666 -
accuracy: 0.6792 - val_loss: 1.6370 - val_accuracy: 0.4611
Epoch 186/700
2880/2880 [==============================] - 33s 12ms/step - loss: 1.0619 -
accuracy: 0.6830 - val_loss: 1.6531 - val_accuracy: 0.4653
Epoch 187/700
2880/2880 [==============================] - 35s 12ms/step - loss: 1.0619 -
accuracy: 0.6896 - val_loss: 1.5862 - val_accuracy: 0.4944
Epoch 188/700
2880/2880 [==============================] - 36s 13ms/step - loss: 1.0420 -
accuracy: 0.6979 - val_loss: 1.5283 - val_accuracy: 0.5306
Epoch 189/700
2880/2880 [==============================] - 34s 12ms/step - loss: 1.0430 -
accuracy: 0.6965 - val_loss: 1.6012 - val_accuracy: 0.4903
Epoch 190/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.0369 -
accuracy: 0.6924 - val_loss: 1.5728 - val_accuracy: 0.4944
Epoch 191/700
2880/2880 [==============================] - 44s 15ms/step - loss: 1.0391 -
accuracy: 0.6892 - val_loss: 1.6916 - val_accuracy: 0.4444
Epoch 192/700
2880/2880 [==============================] - 37s 13ms/step - loss: 1.0251 -
accuracy: 0.7069 - val_loss: 1.5149 - val_accuracy: 0.5097
Epoch 193/700
2880/2880 [==============================] - 28s 10ms/step - loss: 1.0338 -
accuracy: 0.7024 - val_loss: 1.6323 - val_accuracy: 0.4653
Epoch 194/700
2880/2880 [==============================] - 29s 10ms/step - loss: 1.0275 -
accuracy: 0.7021 - val_loss: 1.6936 - val_accuracy: 0.4500
Epoch 195/700
2880/2880 [==============================] - 26s 9ms/step - loss: 1.0110 -
accuracy: 0.7042 - val_loss: 1.5722 - val_accuracy: 0.4931
Epoch 196/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.9969 -
accuracy: 0.7160 - val_loss: 1.5411 - val_accuracy: 0.5097
Epoch 197/700
2880/2880 [==============================] - 27s 9ms/step - loss: 1.0089 -
accuracy: 0.7052 - val_loss: 1.6084 - val_accuracy: 0.4722
Epoch 198/700
2880/2880 [==============================] - 26s 9ms/step - loss: 1.0008 -
accuracy: 0.7080 - val_loss: 1.5737 - val_accuracy: 0.4722
Epoch 199/700
2880/2880 [==============================] - 27s 9ms/step - loss: 1.0021 -
accuracy: 0.7059 - val_loss: 1.5484 - val_accuracy: 0.5125
Epoch 200/700
```

```
2880/2880 [==============================] - 30s 11ms/step - loss: 0.9933 -
accuracy: 0.7115 - val_loss: 1.5452 - val_accuracy: 0.5028
Epoch 201/700
2880/2880 [==============================] - 36s 12ms/step - loss: 0.9854 -
accuracy: 0.7153 - val_loss: 1.5476 - val_accuracy: 0.5194
Epoch 202/700
2880/2880 [==============================] - 37s 13ms/step - loss: 0.9823 -
accuracy: 0.7146 - val_loss: 1.6190 - val_accuracy: 0.4861
Epoch 203/700
2880/2880 [==============================] - 33s 11ms/step - loss: 0.9873 -
accuracy: 0.7250 - val_loss: 1.4987 - val_accuracy: 0.5153
Epoch 204/700
2880/2880 [==============================] - 36s 12ms/step - loss: 0.9762 -
accuracy: 0.7087 - val_loss: 1.5450 - val_accuracy: 0.5125
Epoch 205/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.9531 -
accuracy: 0.7257 - val_loss: 1.5491 - val_accuracy: 0.4986
Epoch 206/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.9657 -
accuracy: 0.7153 - val_loss: 1.5088 - val_accuracy: 0.5181
Epoch 207/700
2880/2880 [==============================] - 36s 13ms/step - loss: 0.9428 -
accuracy: 0.7312 - val_loss: 1.4843 - val_accuracy: 0.5125
Epoch 208/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.9489 -
accuracy: 0.7319 - val_loss: 1.5198 - val_accuracy: 0.5111
Epoch 209/700
2880/2880 [==============================] - 31s 11ms/step - loss: 0.9413 -
accuracy: 0.7264 - val_loss: 1.6898 - val_accuracy: 0.4708
Epoch 210/700
2880/2880 [==============================] - 36s 12ms/step - loss: 0.9360 -
accuracy: 0.7312 - val_loss: 1.5075 - val_accuracy: 0.5125
Epoch 211/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.9389 -
accuracy: 0.7278 - val_loss: 1.5834 - val_accuracy: 0.4792
Epoch 212/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.9267 -
accuracy: 0.7417 - val_loss: 1.8820 - val_accuracy: 0.3958
Epoch 213/700
2880/2880 [==============================] - 36s 12ms/step - loss: 0.9308 -
accuracy: 0.7337 - val_loss: 1.7257 - val_accuracy: 0.4167
Epoch 214/700
2880/2880 [==============================] - 37s 13ms/step - loss: 0.9226 -
accuracy: 0.7372 - val_loss: 1.5298 - val_accuracy: 0.5222
Epoch 215/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.9096 -
accuracy: 0.7510 - val_loss: 1.4733 - val_accuracy: 0.5181
Epoch 216/700
```

```
2880/2880 [==============================] - 33s 11ms/step - loss: 0.9017 -
accuracy: 0.7410 - val_loss: 1.5116 - val_accuracy: 0.5125
Epoch 217/700
2880/2880 [==============================] - 34s 12ms/step - loss: 0.8991 -
accuracy: 0.7312 - val_loss: 1.5198 - val_accuracy: 0.5208
Epoch 218/700
2880/2880 [==============================] - 28s 10ms/step - loss: 0.8931 -
accuracy: 0.7552 - val_loss: 1.5316 - val_accuracy: 0.4972
Epoch 219/700
2880/2880 [==============================] - 30s 10ms/step - loss: 0.8886 -
accuracy: 0.7517 - val_loss: 1.5579 - val_accuracy: 0.5111
Epoch 220/700
2880/2880 [==============================] - 32s 11ms/step - loss: 0.8883 -
accuracy: 0.7469 - val_loss: 1.5118 - val_accuracy: 0.5083
Epoch 221/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.8910 -
accuracy: 0.7448 - val_loss: 1.4607 - val_accuracy: 0.5542
Epoch 222/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.8732 -
accuracy: 0.7545 - val_loss: 1.5382 - val_accuracy: 0.5083
Epoch 223/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.8769 -
accuracy: 0.7528 - val_loss: 1.4415 - val_accuracy: 0.5444
Epoch 224/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8667 -
accuracy: 0.7587 - val_loss: 1.5084 - val_accuracy: 0.5042
Epoch 225/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8583 -
accuracy: 0.7632 - val_loss: 1.4959 - val_accuracy: 0.5292
Epoch 226/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8571 -
accuracy: 0.7611 - val_loss: 1.4215 - val_accuracy: 0.5653
Epoch 227/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8589 -
accuracy: 0.7594 - val_loss: 1.6064 - val_accuracy: 0.4847
Epoch 228/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8454 -
accuracy: 0.7684 - val_loss: 1.4922 - val_accuracy: 0.5222
Epoch 229/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.8487 -
accuracy: 0.7632 - val_loss: 1.5149 - val_accuracy: 0.5042
Epoch 230/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8295 -
accuracy: 0.7670 - val_loss: 1.7186 - val_accuracy: 0.4222
Epoch 231/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8488 -
accuracy: 0.7580 - val_loss: 1.4542 - val_accuracy: 0.5097
Epoch 232/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8334 -
accuracy: 0.7656 - val_loss: 1.7771 - val_accuracy: 0.3903
Epoch 233/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8352 -
accuracy: 0.7618 - val_loss: 1.4504 - val_accuracy: 0.5375
Epoch 234/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.8106 -
accuracy: 0.7767 - val_loss: 1.4545 - val_accuracy: 0.5222
Epoch 235/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.8080 -
accuracy: 0.7788 - val_loss: 1.4666 - val_accuracy: 0.5403
Epoch 236/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.8030 -
accuracy: 0.7736 - val_loss: 1.4857 - val_accuracy: 0.5181
Epoch 237/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.8116 -
accuracy: 0.7708 - val_loss: 1.5954 - val_accuracy: 0.4806
Epoch 238/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.8096 -
accuracy: 0.7733 - val_loss: 1.4615 - val_accuracy: 0.5236
Epoch 239/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7987 -
accuracy: 0.7726 - val_loss: 1.3867 - val_accuracy: 0.5722
Epoch 240/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7988 -
accuracy: 0.7840 - val_loss: 1.4771 - val_accuracy: 0.5083
Epoch 241/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7952 -
accuracy: 0.7882 - val_loss: 1.4940 - val_accuracy: 0.5222
Epoch 242/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7728 -
accuracy: 0.7896 - val_loss: 1.4153 - val_accuracy: 0.5486
Epoch 243/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7796 -
accuracy: 0.7941 - val_loss: 1.4964 - val_accuracy: 0.5222
Epoch 244/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7674 -
accuracy: 0.7889 - val_loss: 1.4267 - val_accuracy: 0.5417
Epoch 245/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7844 -
accuracy: 0.7778 - val_loss: 1.4575 - val_accuracy: 0.5403
Epoch 246/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7661 -
accuracy: 0.7917 - val_loss: 1.4638 - val_accuracy: 0.5347
Epoch 247/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7664 -
accuracy: 0.7951 - val_loss: 1.5695 - val_accuracy: 0.4833
Epoch 248/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7509 -
accuracy: 0.7976 - val_loss: 1.4803 - val_accuracy: 0.5250
Epoch 249/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7691 -
accuracy: 0.7983 - val_loss: 1.4512 - val_accuracy: 0.5333
Epoch 250/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7457 -
accuracy: 0.7990 - val_loss: 1.4010 - val_accuracy: 0.5514
Epoch 251/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7524 -
accuracy: 0.7944 - val_loss: 1.3903 - val_accuracy: 0.5500
Epoch 252/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7431 -
accuracy: 0.8059 - val_loss: 1.3617 - val_accuracy: 0.5653
Epoch 253/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7454 -
accuracy: 0.7969 - val_loss: 1.4207 - val_accuracy: 0.5389
Epoch 254/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7288 -
accuracy: 0.8035 - val_loss: 1.5427 - val_accuracy: 0.5097
Epoch 255/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7249 -
accuracy: 0.8069 - val_loss: 1.4579 - val_accuracy: 0.5375
Epoch 256/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7145 -
accuracy: 0.8059 - val_loss: 1.4398 - val_accuracy: 0.5403
Epoch 257/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7267 -
accuracy: 0.8045 - val_loss: 1.4033 - val_accuracy: 0.5375
Epoch 258/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.7142 -
accuracy: 0.8094 - val_loss: 1.4537 - val_accuracy: 0.5347
Epoch 259/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7049 -
accuracy: 0.8184 - val_loss: 1.4333 - val_accuracy: 0.5458
Epoch 260/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7037 -
accuracy: 0.8188 - val_loss: 1.7141 - val_accuracy: 0.4306
Epoch 261/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7071 -
accuracy: 0.8045 - val_loss: 1.5073 - val_accuracy: 0.4931
Epoch 262/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7071 -
accuracy: 0.8094 - val_loss: 1.4504 - val_accuracy: 0.5347
Epoch 263/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6851 -
accuracy: 0.8208 - val_loss: 1.4847 - val_accuracy: 0.5042
Epoch 264/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6844 -
accuracy: 0.8174 - val_loss: 1.3263 - val_accuracy: 0.5708
Epoch 265/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.7089 -
accuracy: 0.8097 - val_loss: 1.3771 - val_accuracy: 0.5458
Epoch 266/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6763 -
accuracy: 0.8253 - val_loss: 1.3992 - val_accuracy: 0.5500
Epoch 267/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6904 -
accuracy: 0.8160 - val_loss: 1.4349 - val_accuracy: 0.5125
Epoch 268/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6722 -
accuracy: 0.8229 - val_loss: 1.4964 - val_accuracy: 0.5194
Epoch 269/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6819 -
accuracy: 0.8233 - val_loss: 1.3169 - val_accuracy: 0.5875
Epoch 270/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6611 -
accuracy: 0.8236 - val_loss: 1.3618 - val_accuracy: 0.5722
Epoch 271/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.6777 -
accuracy: 0.8170 - val_loss: 1.3956 - val_accuracy: 0.5417
Epoch 272/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6536 -
accuracy: 0.8306 - val_loss: 1.4834 - val_accuracy: 0.5208
Epoch 273/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6576 -
accuracy: 0.8247 - val_loss: 1.3323 - val_accuracy: 0.5833
Epoch 274/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6413 -
accuracy: 0.8344 - val_loss: 1.3252 - val_accuracy: 0.5889
Epoch 275/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6402 -
accuracy: 0.8413 - val_loss: 1.3312 - val_accuracy: 0.5667
Epoch 276/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6410 -
accuracy: 0.8333 - val_loss: 1.3147 - val_accuracy: 0.5847
Epoch 277/700
2880/2880 [==============================] - 28s 10ms/step - loss: 0.6258 -
accuracy: 0.8403 - val_loss: 1.4662 - val_accuracy: 0.5222
Epoch 278/700
2880/2880 [==============================] - 30s 10ms/step - loss: 0.6224 -
accuracy: 0.8431 - val_loss: 1.5451 - val_accuracy: 0.4931
Epoch 279/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6156 -
accuracy: 0.8444 - val_loss: 1.4119 - val_accuracy: 0.5417
Epoch 280/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6274 -
accuracy: 0.8351 - val_loss: 1.3614 - val_accuracy: 0.5569
Epoch 281/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6243 -
accuracy: 0.8330 - val_loss: 1.3014 - val_accuracy: 0.5750
Epoch 282/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6253 -
accuracy: 0.8382 - val_loss: 1.3957 - val_accuracy: 0.5681
Epoch 283/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6206 -
accuracy: 0.8375 - val_loss: 1.3030 - val_accuracy: 0.5708
Epoch 284/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6186 -
accuracy: 0.8420 - val_loss: 1.3077 - val_accuracy: 0.5833
Epoch 285/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6027 -
accuracy: 0.8410 - val_loss: 1.3276 - val_accuracy: 0.5556
Epoch 286/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5967 -
accuracy: 0.8566 - val_loss: 1.3747 - val_accuracy: 0.5528
Epoch 287/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.6069 -
accuracy: 0.8521 - val_loss: 1.3093 - val_accuracy: 0.5819
Epoch 288/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.6072 -
accuracy: 0.8375 - val_loss: 1.4449 - val_accuracy: 0.5347
Epoch 289/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5940 -
accuracy: 0.8497 - val_loss: 1.3740 - val_accuracy: 0.5486
Epoch 290/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5931 -
accuracy: 0.8462 - val_loss: 1.3512 - val_accuracy: 0.5472
Epoch 291/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5829 -
accuracy: 0.8590 - val_loss: 1.3525 - val_accuracy: 0.5417
Epoch 292/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5895 -
accuracy: 0.8583 - val_loss: 1.2774 - val_accuracy: 0.6111
Epoch 293/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5788 -
accuracy: 0.8590 - val_loss: 1.3130 - val_accuracy: 0.5764
Epoch 294/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.5888 -
accuracy: 0.8556 - val_loss: 1.3042 - val_accuracy: 0.5778
Epoch 295/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5633 -
accuracy: 0.8622 - val_loss: 1.3260 - val_accuracy: 0.5986
Epoch 296/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5745 -
accuracy: 0.8604 - val_loss: 1.3138 - val_accuracy: 0.5750
Epoch 297/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5738 -
accuracy: 0.8507 - val_loss: 1.2512 - val_accuracy: 0.6056
Epoch 298/700
2880/2880 [==============================] - 31s 11ms/step - loss: 0.5541 -
accuracy: 0.8628 - val_loss: 1.2963 - val_accuracy: 0.5750
Epoch 299/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5719 -
accuracy: 0.8604 - val_loss: 1.2637 - val_accuracy: 0.6125
Epoch 300/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5500 -
accuracy: 0.8632 - val_loss: 1.3352 - val_accuracy: 0.5556
Epoch 301/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5493 -
accuracy: 0.8587 - val_loss: 1.2568 - val_accuracy: 0.6056
Epoch 302/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5418 -
accuracy: 0.8656 - val_loss: 1.2746 - val_accuracy: 0.5889
Epoch 303/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5471 -
accuracy: 0.8601 - val_loss: 1.3419 - val_accuracy: 0.5528
Epoch 304/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5459 -
accuracy: 0.8653 - val_loss: 1.2361 - val_accuracy: 0.5972
Epoch 305/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5302 -
accuracy: 0.8733 - val_loss: 1.2725 - val_accuracy: 0.5778
Epoch 306/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5290 -
accuracy: 0.8771 - val_loss: 1.2728 - val_accuracy: 0.5861
Epoch 307/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.5495 -
accuracy: 0.8622 - val_loss: 1.2625 - val_accuracy: 0.6014
Epoch 308/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5285 -
accuracy: 0.8656 - val_loss: 1.3051 - val_accuracy: 0.5764
Epoch 309/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5252 -
accuracy: 0.8767 - val_loss: 1.2266 - val_accuracy: 0.6042
Epoch 310/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5114 -
accuracy: 0.8753 - val_loss: 1.2417 - val_accuracy: 0.6097
Epoch 311/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4952 -
accuracy: 0.8917 - val_loss: 1.2385 - val_accuracy: 0.6194
Epoch 312/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5188 -
accuracy: 0.8788 - val_loss: 1.2290 - val_accuracy: 0.5917
Epoch 313/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5184 -
accuracy: 0.8750 - val_loss: 1.2446 - val_accuracy: 0.6000
Epoch 314/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5210 -
accuracy: 0.8760 - val_loss: 1.2925 - val_accuracy: 0.5861
Epoch 315/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.5045 -
accuracy: 0.8722 - val_loss: 1.3422 - val_accuracy: 0.5833
Epoch 316/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.5147 -
accuracy: 0.8740 - val_loss: 1.3241 - val_accuracy: 0.5750
Epoch 317/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4944 -
accuracy: 0.8840 - val_loss: 1.2169 - val_accuracy: 0.6139
Epoch 318/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5067 -
accuracy: 0.8778 - val_loss: 1.2713 - val_accuracy: 0.5889
Epoch 319/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4924 -
accuracy: 0.8792 - val_loss: 1.3315 - val_accuracy: 0.5764
Epoch 320/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.5099 -
accuracy: 0.8736 - val_loss: 1.2547 - val_accuracy: 0.6014
Epoch 321/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4991 -
accuracy: 0.8861 - val_loss: 1.3096 - val_accuracy: 0.5736
Epoch 322/700
2880/2880 [==============================] - 27s 9ms/step - loss: 0.4769 -
accuracy: 0.8861 - val_loss: 1.2988 - val_accuracy: 0.5875
Epoch 323/700
2880/2880 [==============================] - 28s 10ms/step - loss: 0.4792 -
accuracy: 0.8896 - val_loss: 1.2269 - val_accuracy: 0.6181
Epoch 324/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.4896 -
accuracy: 0.8865 - val_loss: 1.3002 - val_accuracy: 0.5792
Epoch 325/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.4742 -
accuracy: 0.8802 - val_loss: 1.3080 - val_accuracy: 0.5708
Epoch 326/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.4719 -
accuracy: 0.8844 - val_loss: 1.2479 - val_accuracy: 0.5958
Epoch 327/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4663 -
accuracy: 0.8917 - val_loss: 1.2222 - val_accuracy: 0.6069
Epoch 328/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4683 -
accuracy: 0.8910 - val_loss: 1.3741 - val_accuracy: 0.5639
Epoch 329/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4655 -
accuracy: 0.8972 - val_loss: 1.2366 - val_accuracy: 0.6042
Epoch 330/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4685 -
accuracy: 0.8875 - val_loss: 1.2153 - val_accuracy: 0.5958
Epoch 331/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4615 -
accuracy: 0.8955 - val_loss: 1.2427 - val_accuracy: 0.6042
Epoch 332/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.4633 -
accuracy: 0.8906 - val_loss: 1.4339 - val_accuracy: 0.5111
Epoch 333/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4608 -
accuracy: 0.8906 - val_loss: 1.2562 - val_accuracy: 0.5958
Epoch 334/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4571 -
accuracy: 0.8892 - val_loss: 1.2341 - val_accuracy: 0.6278
Epoch 335/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4593 -
accuracy: 0.8962 - val_loss: 1.2774 - val_accuracy: 0.5917
Epoch 336/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4462 -
accuracy: 0.8969 - val_loss: 1.2262 - val_accuracy: 0.6042
Epoch 337/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4428 -
accuracy: 0.8976 - val_loss: 1.1785 - val_accuracy: 0.6236
Epoch 338/700
2880/2880 [==============================] - 30s 10ms/step - loss: 0.4485 -
accuracy: 0.9007 - val_loss: 1.2284 - val_accuracy: 0.6097
Epoch 339/700
2880/2880 [==============================] - 28s 10ms/step - loss: 0.4366 -
accuracy: 0.9003 - val_loss: 1.1745 - val_accuracy: 0.6153
Epoch 340/700
2880/2880 [==============================] - 29s 10ms/step - loss: 0.4380 -
accuracy: 0.8948 - val_loss: 1.1915 - val_accuracy: 0.6167
Epoch 341/700
2880/2880 [==============================] - 29s 10ms/step - loss: 0.4319 -
accuracy: 0.9021 - val_loss: 1.2258 - val_accuracy: 0.6153
Epoch 342/700
2880/2880 [==============================] - 31s 11ms/step - loss: 0.4132 -
accuracy: 0.9125 - val_loss: 1.2332 - val_accuracy: 0.6028
Epoch 343/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.4276 -
accuracy: 0.8924 - val_loss: 1.1843 - val_accuracy: 0.6306
Epoch 344/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4228 -
accuracy: 0.9028 - val_loss: 1.3803 - val_accuracy: 0.5681
Epoch 345/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4186 -
accuracy: 0.9104 - val_loss: 1.2469 - val_accuracy: 0.5889
Epoch 346/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4144 -
accuracy: 0.9083 - val_loss: 1.2111 - val_accuracy: 0.6042
Epoch 347/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4190 -
accuracy: 0.9090 - val_loss: 1.3142 - val_accuracy: 0.5653
Epoch 348/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4280 -
accuracy: 0.9024 - val_loss: 1.2459 - val_accuracy: 0.5833
Epoch 349/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4236 -
accuracy: 0.9028 - val_loss: 1.1676 - val_accuracy: 0.6222
Epoch 350/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4117 -
accuracy: 0.9031 - val_loss: 1.1842 - val_accuracy: 0.6333
Epoch 351/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.4071 -
accuracy: 0.9073 - val_loss: 1.2126 - val_accuracy: 0.6042
Epoch 352/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3999 -
accuracy: 0.9094 - val_loss: 1.1415 - val_accuracy: 0.6389
Epoch 353/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3933 -
accuracy: 0.9146 - val_loss: 1.1566 - val_accuracy: 0.6389
Epoch 354/700
2880/2880 [==============================] - 27s 9ms/step - loss: 0.4043 -
accuracy: 0.9115 - val_loss: 1.1776 - val_accuracy: 0.6028
Epoch 355/700
2880/2880 [==============================] - 29s 10ms/step - loss: 0.3987 -
accuracy: 0.9149 - val_loss: 1.1692 - val_accuracy: 0.6292
Epoch 356/700
2880/2880 [==============================] - 30s 10ms/step - loss: 0.3965 -
accuracy: 0.9122 - val_loss: 1.1738 - val_accuracy: 0.6236
Epoch 357/700
2880/2880 [==============================] - 29s 10ms/step - loss: 0.3916 -
accuracy: 0.9149 - val_loss: 1.1676 - val_accuracy: 0.6278
Epoch 358/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3843 -
accuracy: 0.9146 - val_loss: 1.2475 - val_accuracy: 0.6000
Epoch 359/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3934 -
accuracy: 0.9156 - val_loss: 1.2179 - val_accuracy: 0.6125
Epoch 360/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3885 -
accuracy: 0.9177 - val_loss: 1.1605 - val_accuracy: 0.6361
Epoch 361/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3725 -
accuracy: 0.9198 - val_loss: 1.2468 - val_accuracy: 0.6111
Epoch 362/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.3737 -
accuracy: 0.9233 - val_loss: 1.1186 - val_accuracy: 0.6444
Epoch 363/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3825 -
accuracy: 0.9219 - val_loss: 1.1914 - val_accuracy: 0.6056
Epoch 364/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3755 -
accuracy: 0.9264 - val_loss: 1.2218 - val_accuracy: 0.5903
Epoch 365/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3604 -
accuracy: 0.9253 - val_loss: 1.2524 - val_accuracy: 0.5972
Epoch 366/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3745 -
accuracy: 0.9142 - val_loss: 1.2329 - val_accuracy: 0.6264
Epoch 367/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.3700 -
accuracy: 0.9194 - val_loss: 1.2891 - val_accuracy: 0.5778
Epoch 368/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.3630 -
accuracy: 0.9233 - val_loss: 1.2350 - val_accuracy: 0.6069
Epoch 369/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3778 -
accuracy: 0.9187 - val_loss: 1.1772 - val_accuracy: 0.6250
Epoch 370/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3655 -
accuracy: 0.9156 - val_loss: 1.1745 - val_accuracy: 0.6111
Epoch 371/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.3674 -
accuracy: 0.9142 - val_loss: 1.1468 - val_accuracy: 0.6389
Epoch 372/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.3612 -
accuracy: 0.9212 - val_loss: 1.1443 - val_accuracy: 0.6333
Epoch 373/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.3662 -
accuracy: 0.9194 - val_loss: 1.2101 - val_accuracy: 0.6125
Epoch 374/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.3582 -
accuracy: 0.9240 - val_loss: 1.3157 - val_accuracy: 0.5736
Epoch 375/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3524 -
accuracy: 0.9278 - val_loss: 1.1797 - val_accuracy: 0.6347
Epoch 376/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3416 -
accuracy: 0.9302 - val_loss: 1.1584 - val_accuracy: 0.6264
Epoch 377/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3532 -
accuracy: 0.9253 - val_loss: 1.2172 - val_accuracy: 0.6278
Epoch 378/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3679 -
accuracy: 0.9187 - val_loss: 1.1074 - val_accuracy: 0.6556
Epoch 379/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3351 -
accuracy: 0.9330 - val_loss: 1.1300 - val_accuracy: 0.6319
Epoch 380/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3348 -
accuracy: 0.9299 - val_loss: 1.1928 - val_accuracy: 0.6181
Epoch 381/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.3388 -
accuracy: 0.9302 - val_loss: 1.2452 - val_accuracy: 0.5944
Epoch 382/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3355 -
accuracy: 0.9281 - val_loss: 1.1303 - val_accuracy: 0.6514
Epoch 383/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3335 -
accuracy: 0.9295 - val_loss: 1.1496 - val_accuracy: 0.6347
Epoch 384/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3277 -
accuracy: 0.9323 - val_loss: 1.1584 - val_accuracy: 0.6139
Epoch 385/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3346 -
accuracy: 0.9340 - val_loss: 1.1058 - val_accuracy: 0.6431
Epoch 386/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3371 -
accuracy: 0.9292 - val_loss: 1.2188 - val_accuracy: 0.6125
Epoch 387/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.3290 -
accuracy: 0.9403 - val_loss: 1.2034 - val_accuracy: 0.6222
Epoch 388/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3216 -
accuracy: 0.9354 - val_loss: 1.1382 - val_accuracy: 0.6431
Epoch 389/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3282 -
accuracy: 0.9306 - val_loss: 1.2698 - val_accuracy: 0.5931
Epoch 390/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3260 -
accuracy: 0.9361 - val_loss: 1.1631 - val_accuracy: 0.6347
Epoch 391/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3209 -
accuracy: 0.9330 - val_loss: 1.2122 - val_accuracy: 0.6125
Epoch 392/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3147 -
accuracy: 0.9340 - val_loss: 1.1250 - val_accuracy: 0.6250
Epoch 393/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3151 -
accuracy: 0.9347 - val_loss: 1.1172 - val_accuracy: 0.6347
Epoch 394/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3188 -
accuracy: 0.9361 - val_loss: 1.2239 - val_accuracy: 0.6028
Epoch 395/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3126 -
accuracy: 0.9389 - val_loss: 1.1247 - val_accuracy: 0.6375
Epoch 396/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3241 -
accuracy: 0.9295 - val_loss: 1.1933 - val_accuracy: 0.6208
Epoch 397/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.2924 -
accuracy: 0.9451 - val_loss: 1.1206 - val_accuracy: 0.6472
Epoch 398/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.3127 -
accuracy: 0.9410 - val_loss: 1.1156 - val_accuracy: 0.6292
Epoch 399/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.2961 -
accuracy: 0.9448 - val_loss: 1.0836 - val_accuracy: 0.6556
Epoch 400/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.1227 -
accuracy: 0.9823 - val_loss: 0.9550 - val_accuracy: 0.6944
Epoch 581/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1185 -
accuracy: 0.9861 - val_loss: 0.9674 - val_accuracy: 0.6931
Epoch 582/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1164 -
accuracy: 0.9844 - val_loss: 0.9811 - val_accuracy: 0.6833
Epoch 583/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1228 -
accuracy: 0.9865 - val_loss: 0.9443 - val_accuracy: 0.6931
Epoch 584/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1139 -
accuracy: 0.9875 - val_loss: 0.9984 - val_accuracy: 0.6792
Epoch 585/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.1117 -
accuracy: 0.9872 - val_loss: 0.9816 - val_accuracy: 0.6847
Epoch 586/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1132 -
accuracy: 0.9903 - val_loss: 0.9880 - val_accuracy: 0.6792
Epoch 587/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1232 -
accuracy: 0.9847 - val_loss: 0.9481 - val_accuracy: 0.6972
Epoch 588/700
```

```
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1143 -
accuracy: 0.9851 - val_loss: 0.9706 - val_accuracy: 0.6764
Epoch 589/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1106 -
accuracy: 0.9878 - val_loss: 0.9517 - val_accuracy: 0.6917
Epoch 593/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1134 -
accuracy: 0.9854 - val_loss: 0.9433 - val_accuracy: 0.7014
Epoch 594/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.1065 -
accuracy: 0.9899 - val_loss: 0.9522 - val_accuracy: 0.6903
Epoch 595/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1134 -
accuracy: 0.9882 - val_loss: 1.0557 - val_accuracy: 0.6542
Epoch 596/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1071 -
accuracy: 0.9868 - val_loss: 0.9572 - val_accuracy: 0.6917
Epoch 597/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1159 -
accuracy: 0.9847 - val_loss: 0.9464 - val_accuracy: 0.7111
Epoch 598/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1134 -
accuracy: 0.9854 - val_loss: 0.9417 - val_accuracy: 0.7069
Epoch 599/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1109 -
accuracy: 0.9882 - val_loss: 0.9623 - val_accuracy: 0.6944
Epoch 600/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1089 -
accuracy: 0.9878 - val_loss: 0.9422 - val_accuracy: 0.6986
Epoch 601/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1101 -
accuracy: 0.9913 - val_loss: 0.9563 - val_accuracy: 0.6889
Epoch 602/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1134 -
accuracy: 0.9851 - val_loss: 0.9103 - val_accuracy: 0.7069
Epoch 603/700
2880/2880 [==============================] - 24s 9ms/step - loss: 0.1064 -
accuracy: 0.9878 - val_loss: 0.9286 - val_accuracy: 0.7028
Epoch 604/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1088 -
accuracy: 0.9868 - val_loss: 0.9545 - val_accuracy: 0.6903
Epoch 605/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1105 -
accuracy: 0.9875 - val_loss: 0.9437 - val_accuracy: 0.7097
Epoch 606/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1052 -
accuracy: 0.9868 - val_loss: 0.9489 - val_accuracy: 0.6903
Epoch 607/700
```

```
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1069 -
accuracy: 0.9892 - val_loss: 0.9483 - val_accuracy: 0.7000
Epoch 608/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1041 -
accuracy: 0.9927 - val_loss: 0.9577 - val_accuracy: 0.6903
Epoch 609/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1057 -
accuracy: 0.9872 - val_loss: 0.9469 - val_accuracy: 0.6972
Epoch 610/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1075 -
accuracy: 0.9885 - val_loss: 0.9352 - val_accuracy: 0.7083
Epoch 611/700
 736/2880 [======>…] - ETA: 16s - loss: 0.1017 - accuracy:
0.9851

IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)


2880/2880 [==============================] - 24s 8ms/step - loss: 0.1053 -
accuracy: 0.9882 - val_loss: 0.9364 - val_accuracy: 0.6972
Epoch 612/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1061 -
accuracy: 0.9889 - val_loss: 0.9552 - val_accuracy: 0.6847
Epoch 616/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0991 -
accuracy: 0.9931 - val_loss: 0.9227 - val_accuracy: 0.7042
Epoch 617/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1079 -
accuracy: 0.9875 - val_loss: 0.9406 - val_accuracy: 0.6972
Epoch 618/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1055 -
accuracy: 0.9885 - val_loss: 0.9424 - val_accuracy: 0.6931
Epoch 619/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1018 -
accuracy: 0.9899 - val_loss: 0.9136 - val_accuracy: 0.7153
Epoch 620/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1026 -
accuracy: 0.9882 - val_loss: 0.9724 - val_accuracy: 0.6917
Epoch 621/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1113 -
accuracy: 0.9851 - val_loss: 0.9559 - val_accuracy: 0.6806
```

```
Epoch 622/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1114 -
accuracy: 0.9865 - val_loss: 0.9260 - val_accuracy: 0.7083
Epoch 623/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1080 -
accuracy: 0.9847 - val_loss: 0.9430 - val_accuracy: 0.7056
Epoch 624/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1056 -
accuracy: 0.9875 - val_loss: 0.9760 - val_accuracy: 0.6833
Epoch 625/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0953 -
accuracy: 0.9924 - val_loss: 0.9740 - val_accuracy: 0.6903
Epoch 626/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1015 -
accuracy: 0.9889 - val_loss: 0.9736 - val_accuracy: 0.6806
Epoch 627/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0961 -
accuracy: 0.9899 - val_loss: 0.9369 - val_accuracy: 0.6931
Epoch 628/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0991 -
accuracy: 0.9899 - val_loss: 0.9315 - val_accuracy: 0.7042
Epoch 629/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0983 -
accuracy: 0.9896 - val_loss: 0.9350 - val_accuracy: 0.6986
Epoch 630/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1005 -
accuracy: 0.9892 - val_loss: 0.9131 - val_accuracy: 0.7111
Epoch 631/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1011 -
accuracy: 0.9889 - val_loss: 1.0025 - val_accuracy: 0.6750
Epoch 632/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0977 -
accuracy: 0.9913 - val_loss: 0.9755 - val_accuracy: 0.6847
Epoch 633/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1031 -
accuracy: 0.9865 - val_loss: 0.9069 - val_accuracy: 0.7194
Epoch 634/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1009 -
accuracy: 0.9878 - val_loss: 0.9392 - val_accuracy: 0.7014
Epoch 635/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.1034 -
accuracy: 0.9861 - val_loss: 0.9398 - val_accuracy: 0.7014
Epoch 636/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0950 -
accuracy: 0.9872 - val_loss: 0.9270 - val_accuracy: 0.7056
Epoch 637/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.1033 -
accuracy: 0.9882 - val_loss: 0.9675 - val_accuracy: 0.6972
```

```
Epoch 638/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0964 -
accuracy: 0.9896 - val_loss: 0.9577 - val_accuracy: 0.6972
Epoch 639/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0975 -
accuracy: 0.9889 - val_loss: 0.9243 - val_accuracy: 0.7014
Epoch 640/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0995 -
accuracy: 0.9885 - val_loss: 0.9331 - val_accuracy: 0.7097
Epoch 641/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0990 -
accuracy: 0.9885 - val_loss: 0.9186 - val_accuracy: 0.7097
Epoch 642/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0929 -
accuracy: 0.9917 - val_loss: 0.9068 - val_accuracy: 0.7167
Epoch 643/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0946 -
accuracy: 0.9920 - val_loss: 1.0043 - val_accuracy: 0.6833
Epoch 644/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0954 -
accuracy: 0.9896 - val_loss: 0.9378 - val_accuracy: 0.7028
Epoch 645/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0935 -
accuracy: 0.9917 - val_loss: 0.9367 - val_accuracy: 0.7056
Epoch 646/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0914 -
accuracy: 0.9903 - val_loss: 0.9264 - val_accuracy: 0.7069
Epoch 647/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0870 -
accuracy: 0.9931 - val_loss: 0.9244 - val_accuracy: 0.7125
Epoch 648/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0925 -
accuracy: 0.9899 - val_loss: 0.9171 - val_accuracy: 0.7097
Epoch 649/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0962 -
accuracy: 0.9892 - val_loss: 0.9075 - val_accuracy: 0.7139
Epoch 650/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0893 -
accuracy: 0.9948 - val_loss: 0.9267 - val_accuracy: 0.7069
Epoch 651/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0962 -
accuracy: 0.9910 - val_loss: 0.8929 - val_accuracy: 0.7097
Epoch 652/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0960 -
accuracy: 0.9903 - val_loss: 0.9321 - val_accuracy: 0.7028
Epoch 653/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0869 -
accuracy: 0.9917 - val_loss: 0.9305 - val_accuracy: 0.7139
```

```
Epoch 654/700
1488/2880 [==============>…] - ETA: 11s - loss: 0.0984 - accuracy:
0.9866

IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)


2880/2880 [==============================] - 28s 10ms/step - loss: 0.0976 -
accuracy: 0.9889 - val_loss: 0.9248 - val_accuracy: 0.7083
Epoch 661/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0922 -
accuracy: 0.9906 - val_loss: 0.9352 - val_accuracy: 0.7069
Epoch 662/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0915 -
accuracy: 0.9896 - val_loss: 0.9048 - val_accuracy: 0.7153
Epoch 663/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0961 -
accuracy: 0.9906 - val_loss: 0.8993 - val_accuracy: 0.7083
Epoch 664/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0919 -
accuracy: 0.9892 - val_loss: 0.9323 - val_accuracy: 0.7069
Epoch 665/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0942 -
accuracy: 0.9878 - val_loss: 0.9368 - val_accuracy: 0.6986
Epoch 666/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0921 -
accuracy: 0.9896 - val_loss: 0.9848 - val_accuracy: 0.6903
Epoch 667/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0870 -
accuracy: 0.9910 - val_loss: 0.9225 - val_accuracy: 0.7028
Epoch 668/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0951 -
accuracy: 0.9882 - val_loss: 0.9248 - val_accuracy: 0.7111
Epoch 669/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0882 -
accuracy: 0.9924 - val_loss: 0.9121 - val_accuracy: 0.7056
Epoch 670/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0882 -
accuracy: 0.9906 - val_loss: 0.9196 - val_accuracy: 0.7097
Epoch 671/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0916 -
```

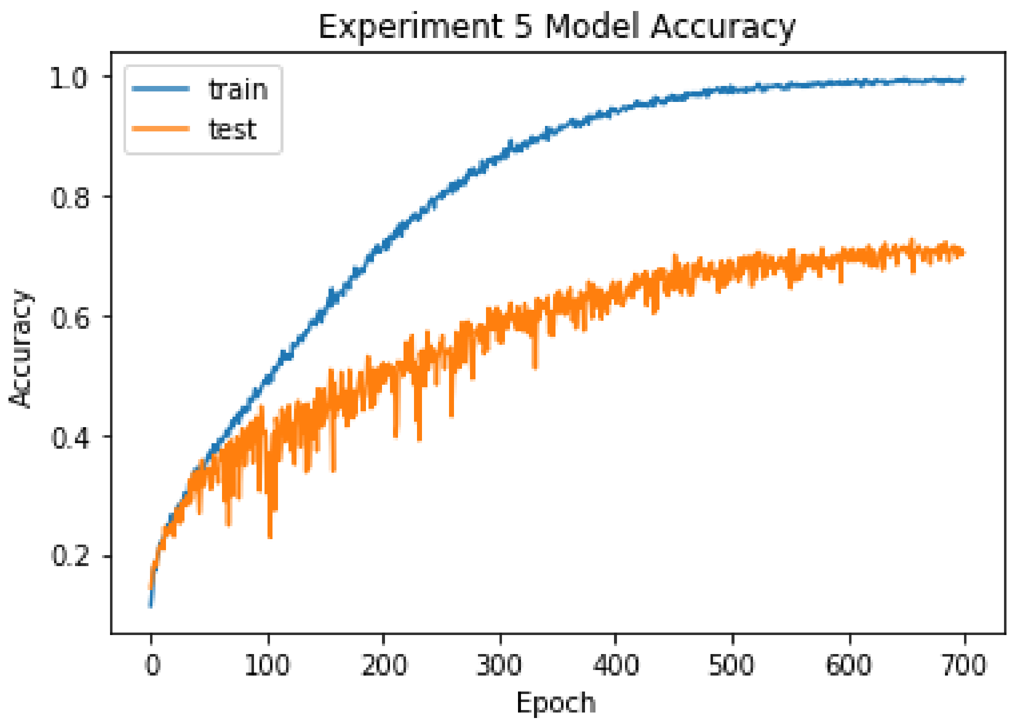
```
accuracy: 0.9906 - val_loss: 0.9333 - val_accuracy: 0.6986
Epoch 672/700
2880/2880 [==============================] - 22s 8ms/step - loss: 0.0863 -
accuracy: 0.9913 - val_loss: 0.9723 - val_accuracy: 0.6847
Epoch 673/700
2880/2880 [==============================] - 27s 9ms/step - loss: 0.0895 -
accuracy: 0.9910 - val_loss: 0.9210 - val_accuracy: 0.7083
Epoch 674/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0844 -
accuracy: 0.9937 - val_loss: 0.9210 - val_accuracy: 0.7056
Epoch 675/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0855 -
accuracy: 0.9944 - val_loss: 0.9301 - val_accuracy: 0.7056
Epoch 676/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0831 -
accuracy: 0.9931 - val_loss: 0.9188 - val_accuracy: 0.7069
Epoch 677/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.0823 -
accuracy: 0.9931 - val_loss: 0.9269 - val_accuracy: 0.7014
Epoch 678/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0916 -
accuracy: 0.9896 - val_loss: 0.9302 - val_accuracy: 0.7083
Epoch 679/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0899 -
accuracy: 0.9906 - val_loss: 0.9262 - val_accuracy: 0.7139
Epoch 680/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0888 -
accuracy: 0.9878 - val_loss: 0.9585 - val_accuracy: 0.6917
Epoch 681/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0853 -
accuracy: 0.9920 - val_loss: 0.9039 - val_accuracy: 0.7167
Epoch 682/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0816 -
accuracy: 0.9931 - val_loss: 0.9275 - val_accuracy: 0.7028
Epoch 683/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0961 -
accuracy: 0.9872 - val_loss: 0.9046 - val_accuracy: 0.7167
Epoch 684/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0854 -
accuracy: 0.9927 - val_loss: 0.9167 - val_accuracy: 0.7153
Epoch 685/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0859 -
accuracy: 0.9903 - val_loss: 0.8988 - val_accuracy: 0.7208
Epoch 686/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0844 -
accuracy: 0.9931 - val_loss: 0.9258 - val_accuracy: 0.7042
Epoch 687/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0818 -
```

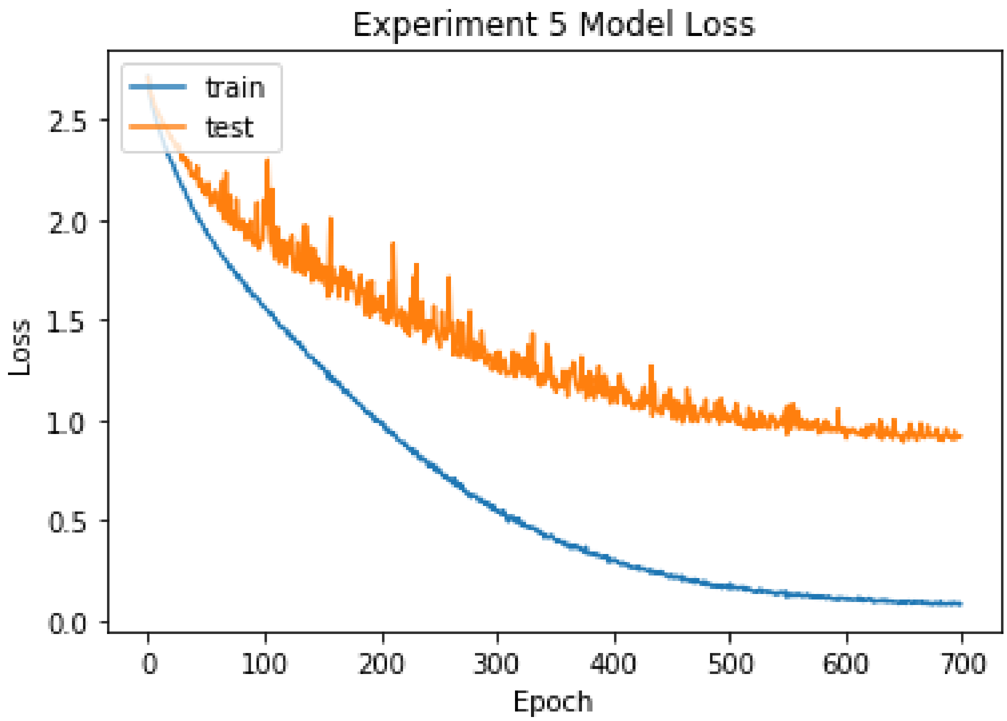
```
accuracy: 0.9913 - val_loss: 0.9232 - val_accuracy: 0.7083
Epoch 688/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0823 -
accuracy: 0.9941 - val_loss: 0.9577 - val_accuracy: 0.6889
Epoch 689/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0872 -
accuracy: 0.9913 - val_loss: 0.9404 - val_accuracy: 0.7069
Epoch 690/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0880 -
accuracy: 0.9892 - val_loss: 0.9151 - val_accuracy: 0.7111
Epoch 691/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.0863 -
accuracy: 0.9917 - val_loss: 0.9153 - val_accuracy: 0.7139
Epoch 692/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0863 -
accuracy: 0.9924 - val_loss: 0.9226 - val_accuracy: 0.7069
Epoch 693/700
2880/2880 [==============================] - 25s 9ms/step - loss: 0.0808 -
accuracy: 0.9903 - val_loss: 0.9038 - val_accuracy: 0.7139
Epoch 694/700
2880/2880 [==============================] - 27s 9ms/step - loss: 0.0919 -
accuracy: 0.9896 - val_loss: 0.9107 - val_accuracy: 0.7069
Epoch 695/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.0879 -
accuracy: 0.9896 - val_loss: 0.9467 - val_accuracy: 0.6958
Epoch 696/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0876 -
accuracy: 0.9920 - val_loss: 0.9058 - val_accuracy: 0.7111
Epoch 697/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0855 -
accuracy: 0.9892 - val_loss: 0.9182 - val_accuracy: 0.7083
Epoch 698/700
2880/2880 [==============================] - 24s 8ms/step - loss: 0.0839 -
accuracy: 0.9882 - val_loss: 0.9137 - val_accuracy: 0.7097
Epoch 699/700
2880/2880 [==============================] - 23s 8ms/step - loss: 0.0886 -
accuracy: 0.9910 - val_loss: 0.9244 - val_accuracy: 0.7000
Epoch 700/700
2880/2880 [==============================] - 26s 9ms/step - loss: 0.0830 -
accuracy: 0.9941 - val_loss: 0.9213 - val_accuracy: 0.7042
```

[0]:

```
plt.plot(cnnhistory.history['accuracy'])
plt.plot(cnnhistory.history['val_accuracy'])
plt.title('Experiment 5 Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()

plt.plot(cnnhistory.history['loss'])
plt.plot(cnnhistory.history['val_loss'])
plt.title('Experiment 5 Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





[0]:
```
# Saving the model.json

import json
model_json = model.to_json()
with open("experiment_5_model_json.json", "w") as json_file:
    json_file.write(model_json)
```

[0]:
```
# loading json and creating model
from keras.models import model_from_json
json_file = open('experiment_5_model_json.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
loaded_model.load_weights("./saved_models/Experiment_5_model.h5")
print("Loaded model from disk")

# evaluate loaded model on test data
loaded_model.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics=['accuracy'])
score = loaded_model.evaluate(x_testcnn, y_test, verbose=0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))
```

```
Loaded model from disk
accuracy: 70.97%
```

[0]:
```
len(data3_df)
```

[0]:
```
240
```

[0]:
```
data3_df.head()
```

[0]:
```
                                        path  source  actor  gender  intensity  \
0  ./RAVDESS/Actor_21/03-01-04-01-01-01-21.wav       1     21    male          0
1  ./RAVDESS/Actor_21/03-01-05-02-02-01-21.wav       1     21    male          1
2  ./RAVDESS/Actor_21/03-01-06-02-01-02-21.wav       1     21    male          1
3  ./RAVDESS/Actor_21/03-01-05-02-01-02-21.wav       1     21    male          1
4  ./RAVDESS/Actor_21/03-01-05-01-02-01-21.wav       1     21    male          0

   statement  repetition  emotion         label
0          0           0        4      male_sad
1          1           0        5    male_angry
2          0           1        6  male_fearful
3          0           1        5    male_angry
4          1           0        5    male_angry
```

[0]:
```
data_test = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data3_df))):
    X, sample_rate = librosa.load(data3_df.path[i],␣
↪res_type='kaiser_fast',duration=input_duration,sr=22050*2,offset=0.5)
#     X = X[10000:90000]
    sample_rate = np.array(sample_rate)
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=13),␣
↪axis=0)
    feature = mfccs
    data_test.loc[i] = [feature]

test_valid = pd.DataFrame(data_test['feature'].values.tolist())
test_valid = np.array(test_valid)
test_valid_lb = np.array(data3_df.label)
lb = LabelEncoder()
test_valid_lb = np_utils.to_categorical(lb.fit_transform(test_valid_lb))
test_valid = np.expand_dims(test_valid, axis=2)
```

```
100%|          | 240/240 [00:14<00:00, 16.30it/s]
```

[0]:
```
preds = loaded_model.predict(test_valid,
                         batch_size=16,
                         verbose=1)
```

```
240/240 [==============================] - 0s 2ms/step
```

[0]:
```
preds
```

[0]:
```
array([[3.0842977e-05, 3.8080507e-05, 2.9988369e-04, …, 1.6876167e-02,
         5.4351506e-03, 2.4606060e-02],
        [4.0956683e-02, 8.9627457e-07, 3.2169750e-04, …, 1.3101639e-04,
         3.2533935e-04, 3.8803718e-03],
        [9.1405241e-03, 4.0867736e-04, 8.4304746e-04, …, 2.9067406e-02,
         9.3415216e-02, 1.5357861e-01],
        …,
        [2.5686793e-04, 6.2882978e-01, 5.6472700e-02, …, 5.2473741e-03,
         3.8200136e-02, 6.2902283e-04],
        [2.4085045e-01, 9.2818225e-03, 9.2755541e-02, …, 1.0005037e-04,
         1.2099562e-03, 1.1845501e-04],
        [9.7002443e-03, 4.7807696e-01, 3.5796919e-01, …, 4.4292139e-04,
         3.0974627e-03, 5.6699291e-04]], dtype=float32)
```

[0]:
```
preds1=preds.argmax(axis=1)
```

[0]:
```
abc = preds1.astype(int).flatten()
predictions = (lb.inverse_transform(abc))
preddf = pd.DataFrame({'predictedvalues': predictions})
print(preddf[:10])

actual=test_valid_lb.argmax(axis=1)
abc123 = actual.astype(int).flatten()
actualvalues = (lb.inverse_transform((abc123)))
actualdf = pd.DataFrame({'actualvalues': actualvalues})
print(actualdf[:10])
finaldf = actualdf.join(preddf)
finaldf[20:40]
```

```
  predictedvalues
0    male_disgust
1      male_angry
2    male_fearful
3      male_angry
4      male_angry
5    male_disgust
6    female_angry
7      male_angry
8      male_angry
9      male_happy
     actualvalues
0        male_sad
1      male_angry
2    male_fearful
3      male_angry
```

```
4        male_angry
5      male_fearful
6    male_surprised
7        male_happy
8      male_disgust
9    male_surprised
```

[0]:

```
     actualvalues  predictedvalues
20   male_neutral     male_disgust
21     male_happy       male_angry
22      male_calm     male_disgust
23     male_happy     male_disgust
24     male_happy       male_angry
25      male_calm     male_disgust
26     male_angry       male_angry
27       male_sad       male_happy
28     male_happy     male_disgust
29   male_disgust     male_disgust
30   male_disgust     male_disgust
31   male_fearful   male_surprised
32     male_happy       male_angry
33   male_disgust     male_disgust
34       male_sad     male_disgust
35   male_disgust     male_disgust
36   male_neutral     male_disgust
37     male_angry       male_angry
38 male_surprised     female_angry
39   male_neutral        male_calm
```

[0]:

```
finaldf.groupby('actualvalues').count()
```

[0]:

```
                  predictedvalues
actualvalues
female_angry                   16
female_calm                    16
female_disgust                 16
female_fearful                 16
female_happy                   16
female_neutral                  8
female_sad                     16
female_surprised               16
male_angry                     16
male_calm                      16
male_disgust                   16
male_fearful                   16
male_happy                     16
male_neutral                    8
```

```
male_sad                    16
male_surprised              16
```

[0]:
```
finaldf.groupby('predictedvalues').count()
```

[0]:
```
                  actualvalues
predictedvalues
female_angry                48
female_calm                 21
female_disgust              22
female_fearful               5
female_happy                 9
female_neutral              13
female_sad                  24
female_surprised            10
male_angry                  17
male_calm                    7
male_disgust                38
male_fearful                 2
male_happy                  10
male_neutral                 2
male_sad                     5
male_surprised               7
```

[0]:
```
finaldf.to_csv('Predictions_Experiment_5.csv', index=False)
```

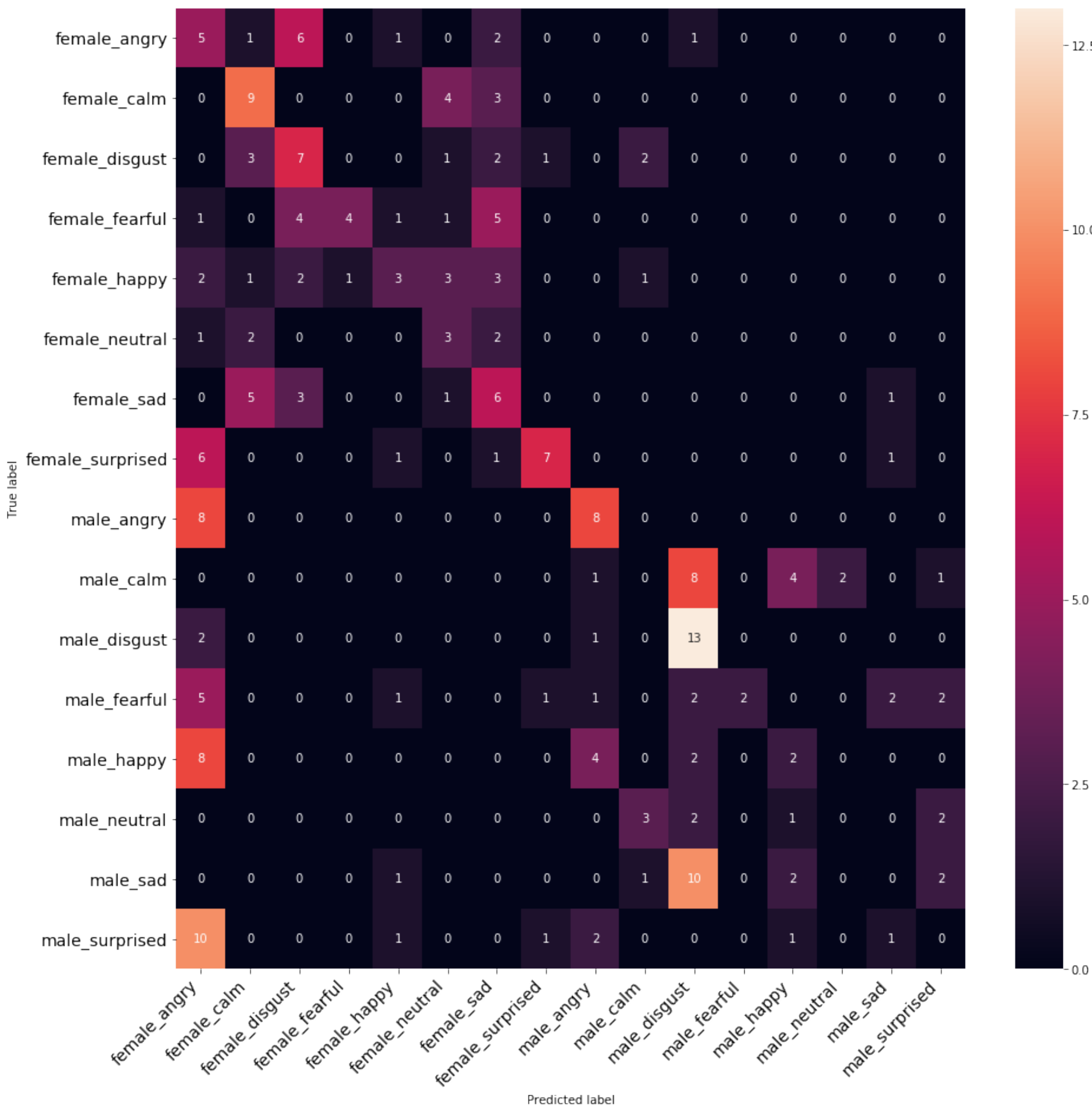
[0]:
```
from sklearn.metrics import accuracy_score
y_true = finaldf.actualvalues
y_pred = finaldf.predictedvalues
print(accuracy_score(y_true, y_pred)*100)
```

```
28.749999999999996
```

[0]:
```
classes = finaldf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(finaldf.actualvalues, finaldf.predictedvalues)
print(accuracy_score(finaldf.actualvalues, finaldf.predictedvalues))
print_confusion_matrix(c, class_names = classes)

# Classification report
classes = finaldf.actualvalues.unique()
classes.sort()
print(classification_report(finaldf.actualvalues, finaldf.predictedvalues,␣
 ↪target_names=classes))
```

```
0.2875
```

	precision	recall	f1-score	support
female_angry	0.10	0.31	0.16	16
female_calm	0.43	0.56	0.49	16
female_disgust	0.32	0.44	0.37	16
female_fearful	0.80	0.25	0.38	16
female_happy	0.33	0.19	0.24	16
female_neutral	0.23	0.38	0.29	8
female_sad	0.25	0.38	0.30	16
female_surprised	0.70	0.44	0.54	16
male_angry	0.47	0.50	0.48	16
male_calm	0.00	0.00	0.00	16
male_disgust	0.34	0.81	0.48	16
male_fearful	1.00	0.12	0.22	16
male_happy	0.20	0.12	0.15	16
male_neutral	0.00	0.00	0.00	8
male_sad	0.00	0.00	0.00	16
male_surprised	0.00	0.00	0.00	16
accuracy			0.29	240
macro avg	0.32	0.28	0.26	240
weighted avg	0.34	0.29	0.26	240



6.0.4 We had not expected such poor results

Anyway, let's check for gender and emotions

[0]:
```
classes
```

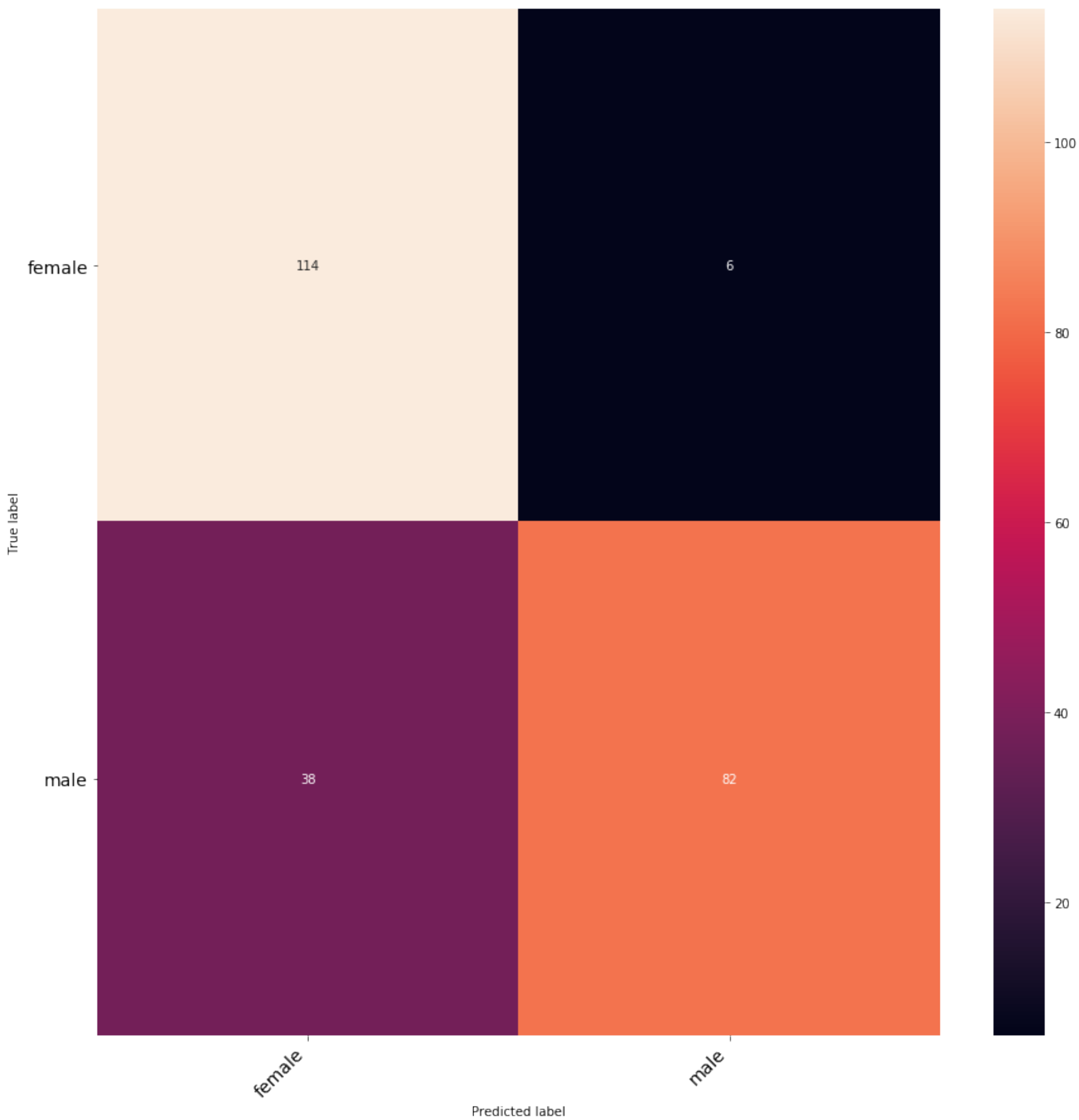
[0]:
```
array(['female_angry', 'female_calm', 'female_disgust', 'female_fearful',
       'female_happy', 'female_neutral', 'female_sad', 'female_surprised',
       'male_angry', 'male_calm', 'male_disgust', 'male_fearful',
       'male_happy', 'male_neutral', 'male_sad', 'male_surprised'],
      dtype=object)
```

Let's group the gender and check for the results for the combined data

[0]:
```
modidf = finaldf
modidf['actualvalues'] = finaldf.actualvalues.replace({'female_angry':'female'
                                       , 'female_disgust':'female'
                                       , 'female_fearful':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprised':'female'
                                       , 'female_calm':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_fearful':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprised':'male'
                                       , 'male_calm':'male'
                                       , 'male_neutral':'male'
                                       , 'male_disgust':'male'
                                      })

modidf['predictedvalues'] = finaldf.predictedvalues.replace({'female_angry':
 ↪'female'
                                       , 'female_disgust':'female'
                                       , 'female_fearful':'female'
                                       , 'female_happy':'female'
                                       , 'female_sad':'female'
                                       , 'female_surprised':'female'
                                       , 'female_calm':'female'
                                       , 'female_neutral':'female'
                                       , 'male_angry':'male'
                                       , 'male_neutral':'male'
                                       , 'male_fearful':'male'
                                       , 'male_happy':'male'
                                       , 'male_sad':'male'
                                       , 'male_surprised':'male'
                                       , 'male_calm':'male'
                                       , 'male_disgust':'male'
                                      })

classes = modidf.actualvalues.unique()
classes.sort()

# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print("Accuracy is: ",accuracy_score(modidf.actualvalues, modidf.
 ↪predictedvalues))
print_confusion_matrix(c, class_names = classes)
classes = modidf.actualvalues.unique()
```

```
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,
 ↪target_names=classes))
```

```
Accuracy is:  0.8166666666666667
              precision    recall  f1-score   support

      female       0.75      0.95      0.84       120
        male       0.93      0.68      0.79       120

    accuracy                           0.82       240
   macro avg       0.84      0.82      0.81       240
weighted avg       0.84      0.82      0.81       240
```

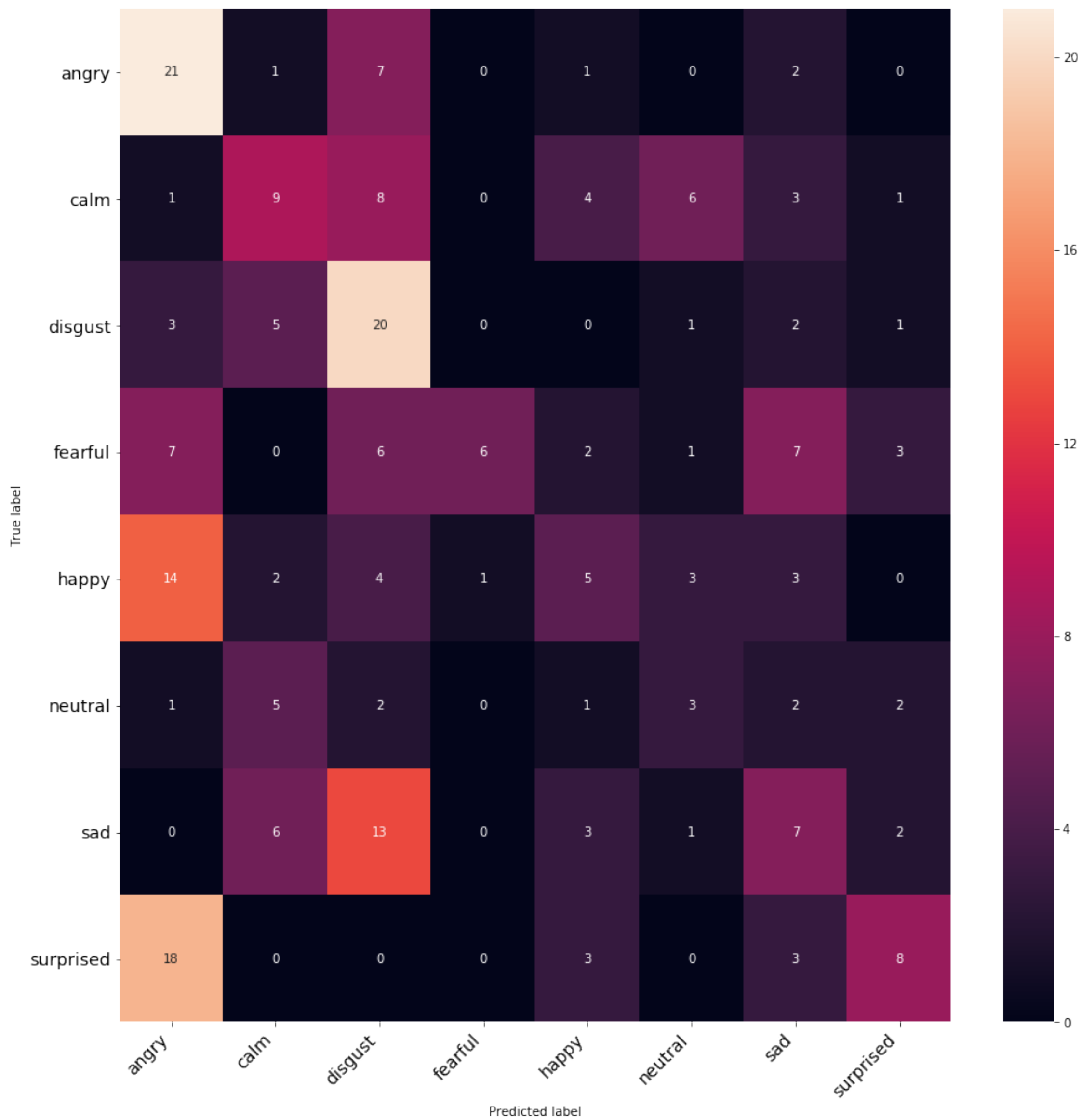


6.1 Let's group together the emotions and look for the performance for the combined data

[0]:
```
modidf = pd.read_csv("Predictions_Experiment_5.csv")
modidf['actualvalues'] = finaldf.actualvalues.replace({'female_angry':'angry'
                                       , 'female_disgust':'disgust'
                                       , 'female_fearful':'fearful'
                                       , 'female_happy':'happy'
                                       , 'female_sad':'sad'
                                       , 'female_surprised':'surprised'
                                       , 'female_calm':'calm'
                                       , 'female_neutral':'neutral'
                                       , 'male_angry':'angry'
                                       , 'male_neutral':'neutral'
                                       , 'male_fearful':'fearful'
                                       , 'male_happy':'happy'
                                       , 'male_sad':'sad'
                                       , 'male_surprised':'surprised'
                                       , 'male_calm':'calm'
                                       , 'male_disgust':'disgust'
                                      })

modidf['predictedvalues'] = finaldf.predictedvalues.replace({'female_angry':
 ↪'angry'
                                       , 'female_disgust':'disgust'
                                       , 'female_fearful':'fearful'
                                       , 'female_happy':'happy'
                                       , 'female_sad':'sad'
                                       , 'female_surprised':'surprised'
                                       , 'female_calm':'calm'
                                       , 'female_neutral':'neutral'
                                       , 'male_angry':'angry'
                                       , 'male_neutral':'neutral'
                                       , 'male_fearful':'fearful'
                                       , 'male_happy':'happy'
                                       , 'male_sad':'sad'
                                       , 'male_surprised':'surprised'
                                       , 'male_calm':'calm'
                                       , 'male_disgust':'disgust'
                                      })

classes = modidf.actualvalues.unique()
classes.sort()
```

```python
# Confusion matrix
c = confusion_matrix(modidf.actualvalues, modidf.predictedvalues)
print("Accuracy is: ",accuracy_score(modidf.actualvalues, modidf.
 ↪predictedvalues))
print_confusion_matrix(c, class_names = classes)
classes = modidf.actualvalues.unique()
classes.sort()
print(classification_report(modidf.actualvalues, modidf.predictedvalues,␣
 ↪target_names=classes))
```

```
Accuracy is:  0.32916666666666666
              precision    recall  f1-score   support

       angry       0.32      0.66      0.43        32
        calm       0.32      0.28      0.30        32
     disgust       0.33      0.62      0.43        32
     fearful       0.86      0.19      0.31        32
       happy       0.26      0.16      0.20        32
     neutral       0.20      0.19      0.19        16
         sad       0.24      0.22      0.23        32
   surprised       0.47      0.25      0.33        32

    accuracy                           0.33       240
   macro avg       0.38      0.32      0.30       240
weighted avg       0.39      0.33      0.31       240
```



7 Experiment 6: Experiments with 2D CNN Architecture

Back to Experiments and Results

[0]:
```
i = 0
```

[0]:
```
def speedNpitch(data):
    length_change = np.random.uniform(low=0.8, high = 1)
    speed_fac = 1.2  / length_change # try changing 1.0 to 2.0 ... =D
    tmp = np.interp(np.arange(0,len(data),speed_fac),np.
↪arange(0,len(data)),data)
```

```
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data

def prepare_data(df, n, aug, mfcc):
    X = np.empty(shape=(df.shape[0], n, 216, 1))
    input_length = sampling_rate * audio_duration

    cnt = 0
    for fname in tqdm(df.path):
        file_path = fname
        data, _ = librosa.load(file_path, sr=sampling_rate
                               ,res_type="kaiser_fast"
                               ,duration=2.5
                               ,offset=0.5
                              )

        if len(data) > input_length:
            max_offset = len(data) - input_length
            offset = np.random.randint(max_offset)
            data = data[offset:(input_length+offset)]
        else:
            if input_length > len(data):
                max_offset = input_length - len(data)
                offset = np.random.randint(max_offset)
            else:
                offset = 0
            data = np.pad(data, (offset, int(input_length) - len(data) -␣
↪offset), "constant")

        if aug == 1:
            data = speedNpitch(data)

        if mfcc == 1:
            MFCC = librosa.feature.mfcc(data, sr=sampling_rate, n_mfcc=n_mfcc)
            MFCC = np.expand_dims(MFCC, axis=-1)
            X[cnt,] = MFCC

        else:
            melspec = librosa.feature.melspectrogram(data, n_mels = n_melspec)
            logspec = librosa.amplitude_to_db(melspec)
            logspec = np.expand_dims(logspec, axis=-1)
            X[cnt,] = logspec

        cnt += 1
```

```python
    return X


def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7),
 ↪fontsize=14):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
        bottom, top = heatmap.get_ylim()
        heatmap.set_ylim(bottom + 0.5, top - 0.5)
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")

    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,
 ↪ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,
 ↪ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')



def get_2d_conv_model(n):
    global i
    i = i + 1
    nclass = 14
    inp = Input(shape=(n,216,1))
    x = Convolution2D(32, (4,10), padding="same")(inp)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)
```

```
    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)
    x = Dropout(rate=0.2)(x)

    x = Flatten()(x)
    x = Dense(64)(x)
    x = Dropout(rate=0.2)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Dropout(rate=0.2)(x)

    out = Dense(nclass, activation=softmax)(x)
    model = models.Model(inputs=inp, outputs=out)

    opt = optimizers.Adam(0.001)
    model.compile(optimizer=opt, loss=losses.categorical_crossentropy,␣
↪metrics=['acc'])
    model.summary()
    tf.keras.utils.plot_model(model, to_file='model_'+str(i)+'.png',␣
↪show_shapes=True, show_layer_names=True)
    return model

class get_results:
    def __init__(self, model_history, model ,X_test, y_test, labels):
        self.model_history = model_history
        self.model = model
        self.X_test = X_test
        self.y_test = y_test
        self.labels = labels

    def create_plot(self, model_history):
        plt.plot(model_history.history['acc'])
        plt.plot(model_history.history['val_acc'])
        plt.title('Model Accuracy')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='best')
        plt.show()
        plt.plot(model_history.history['loss'])
        plt.plot(model_history.history['val_loss'])
        plt.title('Model loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='best')
```

```
        plt.show()

    def create_results(self, model):
        opt = optimizers.Adam(0.001)
        model.compile(loss='categorical_crossentropy', optimizer=opt,␣
↪metrics=['accuracy'])
        score = model.evaluate(X_test, y_test, verbose=0)
        print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

    def confusion_results(self, X_test, y_test, labels, model):
        preds = model.predict(X_test,
                              batch_size=16,
                              verbose=2)
        preds=preds.argmax(axis=1)
        preds = preds.astype(int).flatten()
        preds = (lb.inverse_transform((preds)))

        actual = y_test.argmax(axis=1)
        actual = actual.astype(int).flatten()
        actual = (lb.inverse_transform((actual)))

        classes = labels
        classes.sort()

        c = confusion_matrix(actual, preds)
        print_confusion_matrix(c, class_names = classes)

    def print_classification_report(self, X_test, y_test, labels, model):
        preds = model.predict(X_test,
                              batch_size=16,
                              verbose=2)
        preds=preds.argmax(axis=1)
        preds = preds.astype(int).flatten()
        preds = (lb.inverse_transform((preds)))

        actual = y_test.argmax(axis=1)
        actual = actual.astype(int).flatten()
        actual = (lb.inverse_transform((actual)))

        classes = labels
        classes.sort()
        print(classification_report(actual, preds, target_names=classes))

    def accuracy_results_gender(self, X_test, y_test, labels, model):
        preds = model.predict(X_test,
                     batch_size=16,
                     verbose=2)
```

```
preds = preds.argmax(axis=1)
preds = preds.astype(int).flatten()
preds = (lb.inverse_transform((preds)))

actual = y_test.argmax(axis=1)
actual = actual.astype(int).flatten()
actual = (lb.inverse_transform((actual)))

actual = pd.DataFrame(actual).replace({'female_angry':'female'
           , 'female_disgust':'female'
           , 'female_fear':'female'
           , 'female_happy':'female'
           , 'female_sad':'female'
           , 'female_surprise':'female'
           , 'female_neutral':'female'
           , 'male_angry':'male'
           , 'male_fear':'male'
           , 'male_happy':'male'
           , 'male_sad':'male'
           , 'male_surprise':'male'
           , 'male_neutral':'male'
           , 'male_disgust':'male'
          })
preds = pd.DataFrame(preds).replace({'female_angry':'female'
       , 'female_disgust':'female'
       , 'female_fear':'female'
       , 'female_happy':'female'
       , 'female_sad':'female'
       , 'female_surprise':'female'
       , 'female_neutral':'female'
       , 'male_angry':'male'
       , 'male_fear':'male'
       , 'male_happy':'male'
       , 'male_sad':'male'
       , 'male_surprise':'male'
       , 'male_neutral':'male'
       , 'male_disgust':'male'
      })

classes = actual.loc[:,0].unique()
classes.sort()

c = confusion_matrix(actual, preds)
print(accuracy_score(actual, preds))
print_confusion_matrix(c, class_names = classes)
```

7.1 We will use the Data_combined.csv file that we have created earlier

[0]:
```
ref = pd.read_csv("./Data_combined.csv")
ref.head()
```

[0]:
```
        labels source                       path
0    male_fear  SAVEE   ./SAVEE_used/KL_f04.wav
1   male_angry  SAVEE   ./SAVEE_used/KL_a11.wav
2    male_fear  SAVEE   ./SAVEE_used/JE_f11.wav
3     male_sad  SAVEE  ./SAVEE_used/DC_sa11.wav
4    male_fear  SAVEE   ./SAVEE_used/KL_f08.wav
```

[0]:
```
sampling_rate=44100
audio_duration=2.5
n_mfcc = 30
mfcc = prepare_data(ref, n = n_mfcc, aug = 0, mfcc = 1)
```

```
100%|     | 4720/4720 [03:29<00:00, 22.58it/s]
```

[0]:
```
# Split between train and test
X_train, X_test, y_train, y_test = train_test_split(mfcc
                                                    , ref.labels
                                                    , test_size=0.25
                                                    , shuffle=True
                                                    , random_state=42
                                                   )



# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

# Normalization as per the standard NN process
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std

# Build CNN model
model = get_2d_conv_model(n=n_mfcc)
model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    batch_size=16, epochs=20)
```

```
WARNING:tensorflow:From /home/subodh/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
```

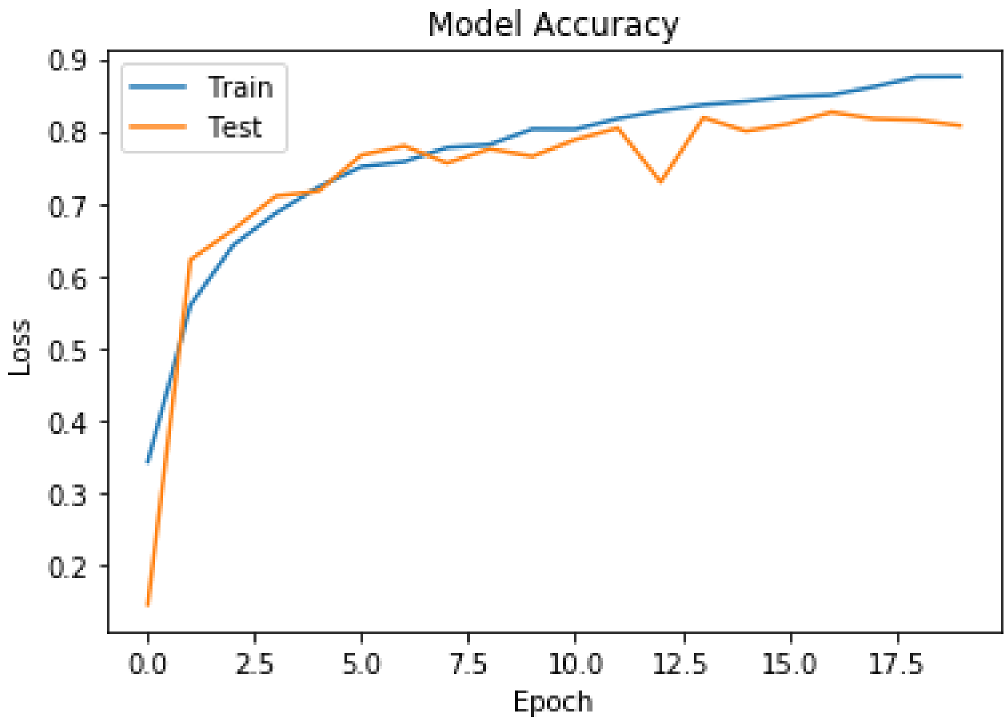
```
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /home/subodh/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:4070: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 30, 216, 1)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 216, 32)       1312
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 216, 32)       128
_________________________________________________________________
activation_1 (Activation)    (None, 30, 216, 32)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 108, 32)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 108, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 108, 32)       40992
_________________________________________________________________
batch_normalization_2 (Batch (None, 15, 108, 32)       128
_________________________________________________________________
activation_2 (Activation)    (None, 15, 108, 32)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 54, 32)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 54, 32)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 54, 32)         40992
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 54, 32)         128
_________________________________________________________________
activation_3 (Activation)    (None, 7, 54, 32)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 27, 32)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 27, 32)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 27, 32)         40992
_________________________________________________________________
batch_normalization_4 (Batch (None, 3, 27, 32)         128
_________________________________________________________________
activation_4 (Activation)    (None, 3, 27, 32)         0
```

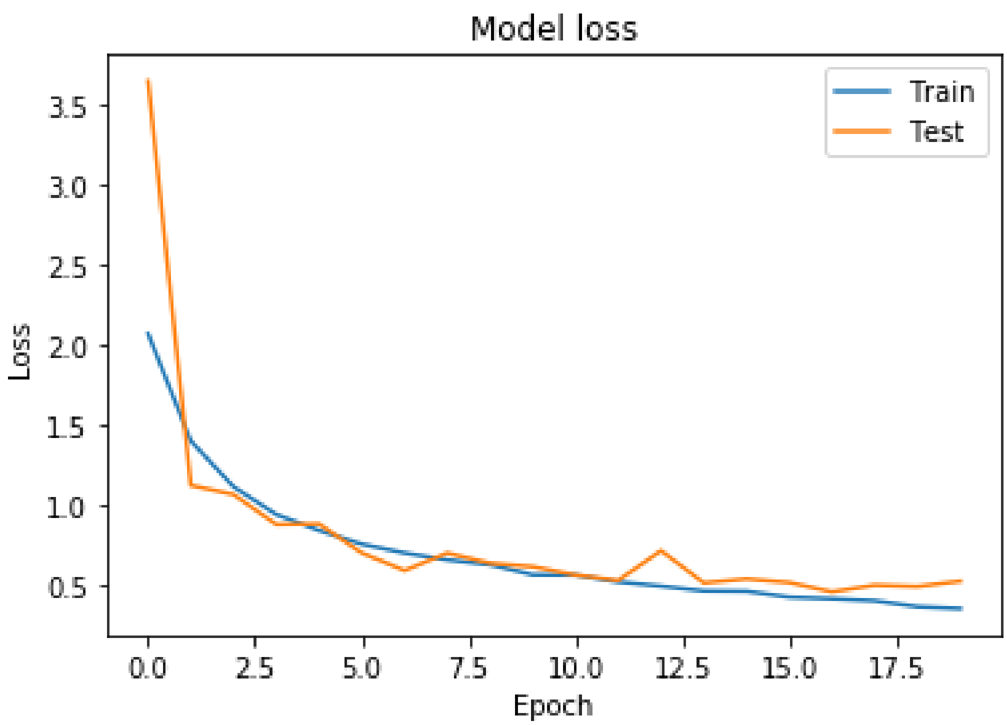
```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 13, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 13, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 416)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)               26688
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)               256
_________________________________________________________________
activation_5 (Activation)    (None, 64)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)               0
_________________________________________________________________
dense_2 (Dense)              (None, 14)               910
=================================================================
Total params: 152,654
Trainable params: 152,270
Non-trainable params: 384
_________________________________________________________________
WARNING:tensorflow:From /home/subodh/anaconda3/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 3540 samples, validate on 1180 samples
Epoch 1/20
3540/3540 [==============================] - 49s 14ms/step - loss: 2.0691 - acc:
0.3424 - val_loss: 3.6425 - val_acc: 0.1449
Epoch 2/20
3540/3540 [==============================] - 49s 14ms/step - loss: 1.3999 - acc:
0.5593 - val_loss: 1.1242 - val_acc: 0.6220
Epoch 3/20
3540/3540 [==============================] - 48s 14ms/step - loss: 1.1158 - acc:
0.6421 - val_loss: 1.0675 - val_acc: 0.6636
Epoch 4/20
3540/3540 [==============================] - 50s 14ms/step - loss: 0.9428 - acc:
0.6867 - val_loss: 0.8804 - val_acc: 0.7102
Epoch 5/20
3540/3540 [==============================] - 48s 13ms/step - loss: 0.8433 - acc:
0.7229 - val_loss: 0.8839 - val_acc: 0.7169
Epoch 6/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.7579 - acc:
0.7508 - val_loss: 0.7032 - val_acc: 0.7669
Epoch 7/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.7024 - acc:
```

```
0.7579 - val_loss: 0.5933 - val_acc: 0.7797
Epoch 8/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.6597 - acc:
0.7774 - val_loss: 0.7023 - val_acc: 0.7559
Epoch 9/20
3540/3540 [==============================] - 50s 14ms/step - loss: 0.6295 - acc:
0.7814 - val_loss: 0.6389 - val_acc: 0.7754
Epoch 10/20
3540/3540 [==============================] - 49s 14ms/step - loss: 0.5669 - acc:
0.8028 - val_loss: 0.6171 - val_acc: 0.7653
Epoch 11/20
3540/3540 [==============================] - 49s 14ms/step - loss: 0.5651 - acc:
0.8025 - val_loss: 0.5656 - val_acc: 0.7881
Epoch 12/20
3540/3540 [==============================] - 49s 14ms/step - loss: 0.5224 - acc:
0.8172 - val_loss: 0.5315 - val_acc: 0.8042
Epoch 13/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.4956 - acc:
0.8282 - val_loss: 0.7172 - val_acc: 0.7297
Epoch 14/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.4659 - acc:
0.8364 - val_loss: 0.5163 - val_acc: 0.8186
Epoch 15/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.4637 - acc:
0.8412 - val_loss: 0.5410 - val_acc: 0.8000
Epoch 16/20
3540/3540 [==============================] - 48s 13ms/step - loss: 0.4273 - acc:
0.8475 - val_loss: 0.5184 - val_acc: 0.8102
Epoch 17/20
3540/3540 [==============================] - 51s 14ms/step - loss: 0.4173 - acc:
0.8500 - val_loss: 0.4605 - val_acc: 0.8263
Epoch 18/20
3540/3540 [==============================] - 48s 14ms/step - loss: 0.4044 - acc:
0.8616 - val_loss: 0.5017 - val_acc: 0.8169
Epoch 19/20
3540/3540 [==============================] - 49s 14ms/step - loss: 0.3677 - acc:
0.8751 - val_loss: 0.4936 - val_acc: 0.8153
Epoch 20/20
3540/3540 [==============================] - 49s 14ms/step - loss: 0.3557 - acc:
0.8754 - val_loss: 0.5267 - val_acc: 0.8076
```

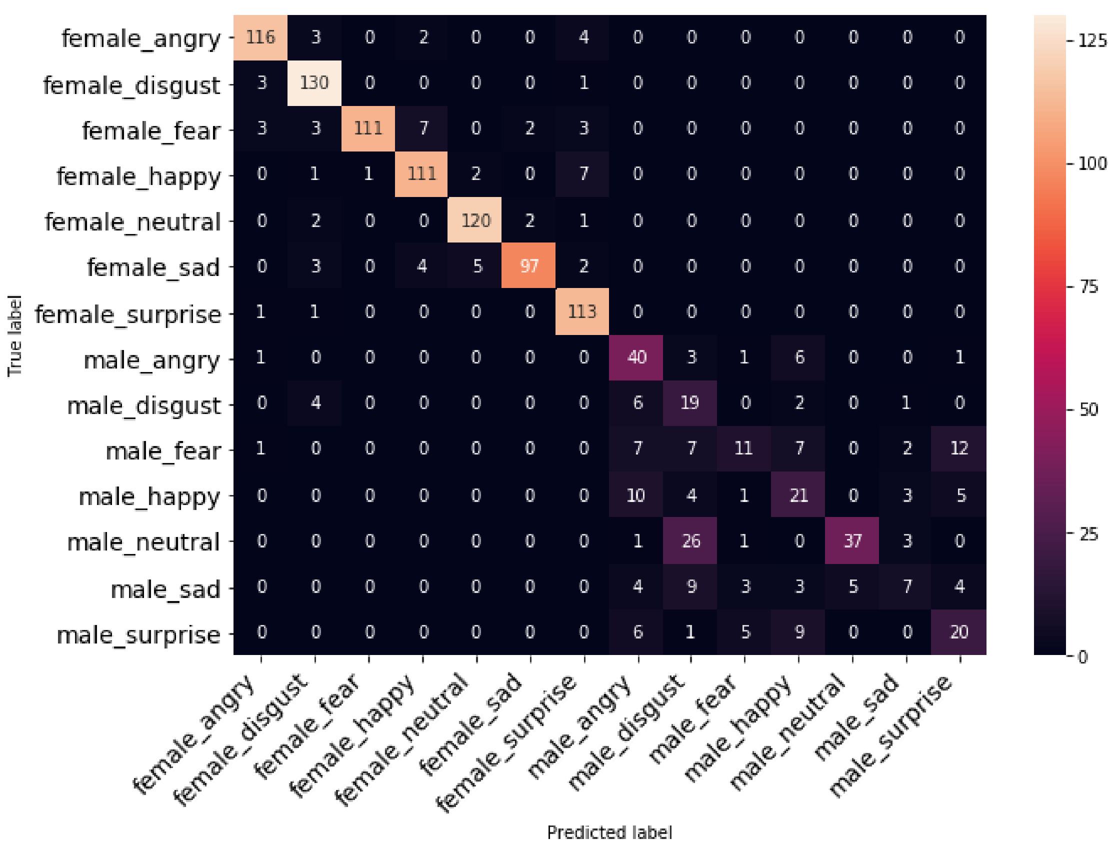
[0]:

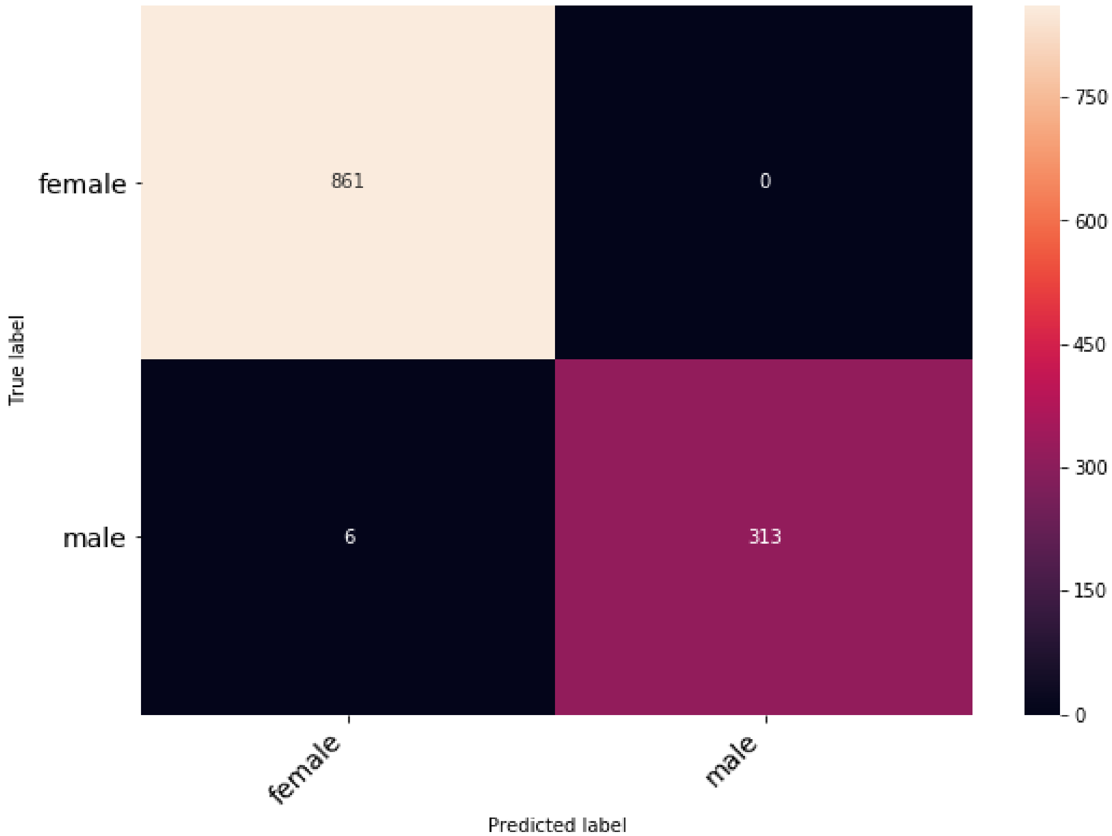
```
results = get_results(model_history,model,X_test,y_test, ref.labels.unique())
results.create_plot(model_history)
results.create_results(model)
results.confusion_results(X_test, y_test, ref.labels.unique(), model)
results.print_classification_report(X_test, y_test, ref.labels.unique(), model)
results.accuracy_results_gender(X_test, y_test, ref.labels.unique(), model)
```


Model Accuracy
Train
Test
Loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model loss
Train
Test
Loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

```
accuracy: 80.76%
                  precision    recall  f1-score   support

   female_angry       0.93      0.93      0.93       125
 female_disgust       0.88      0.97      0.93       134
    female_fear       0.99      0.86      0.92       129
   female_happy       0.90      0.91      0.90       122
 female_neutral       0.94      0.96      0.95       125
     female_sad       0.96      0.87      0.92       111
female_surprise       0.86      0.98      0.92       115
     male_angry       0.54      0.77      0.63        52
   male_disgust       0.28      0.59      0.38        32
      male_fear       0.50      0.23      0.32        47
     male_happy       0.44      0.48      0.46        44
   male_neutral       0.88      0.54      0.67        68
       male_sad       0.44      0.20      0.27        35
  male_surprise       0.48      0.49      0.48        41

       accuracy                           0.81      1180
      macro avg       0.72      0.70      0.69      1180
   weighted avg       0.82      0.81      0.81      1180

0.9949152542372881
```





[0]:
```
sampling_rate=44100
audio_duration=2.5
n_mfcc = 30
mfcc_aug = prepare_data(ref, n = n_mfcc, aug = 1, mfcc = 1)
```

```
100%|          | 4720/4720 [02:17<00:00, 34.24it/s]
```

[0]:
```
# Split between train and test
X_train, X_test, y_train, y_test = train_test_split(mfcc_aug
                                                    , ref.labels
                                                    , test_size=0.25
                                                    , shuffle=True
                                                    , random_state=42
                                                   )


# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
```

```python
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

# Normalization as per the standard NN process
# mean = np.mean(X_train, axis=0)
# std = np.std(X_train, axis=0)

# X_train = (X_train - mean)/std
# X_test = (X_test - mean)/std

# Build CNN model
model = get_2d_conv_model(n=n_mfcc)
model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    batch_size=16, epochs=20)
```

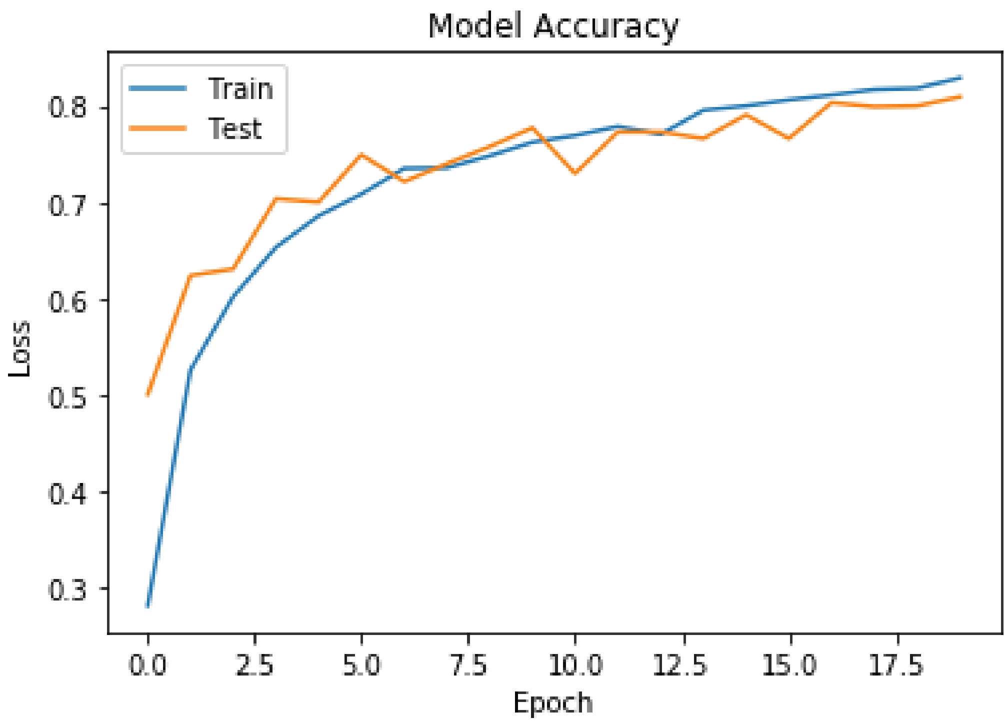
```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 30, 216, 1)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 216, 32)       1312
_________________________________________________________________
batch_normalization_6 (Batch (None, 30, 216, 32)       128
_________________________________________________________________
activation_6 (Activation)    (None, 30, 216, 32)       0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 15, 108, 32)       0
_________________________________________________________________
dropout_7 (Dropout)          (None, 15, 108, 32)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 15, 108, 32)       40992
_________________________________________________________________
batch_normalization_7 (Batch (None, 15, 108, 32)       128
_________________________________________________________________
activation_7 (Activation)    (None, 15, 108, 32)       0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 54, 32)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 54, 32)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 54, 32)         40992
_________________________________________________________________
batch_normalization_8 (Batch (None, 7, 54, 32)         128
_________________________________________________________________
activation_8 (Activation)    (None, 7, 54, 32)         0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 3, 27, 32)         0
```

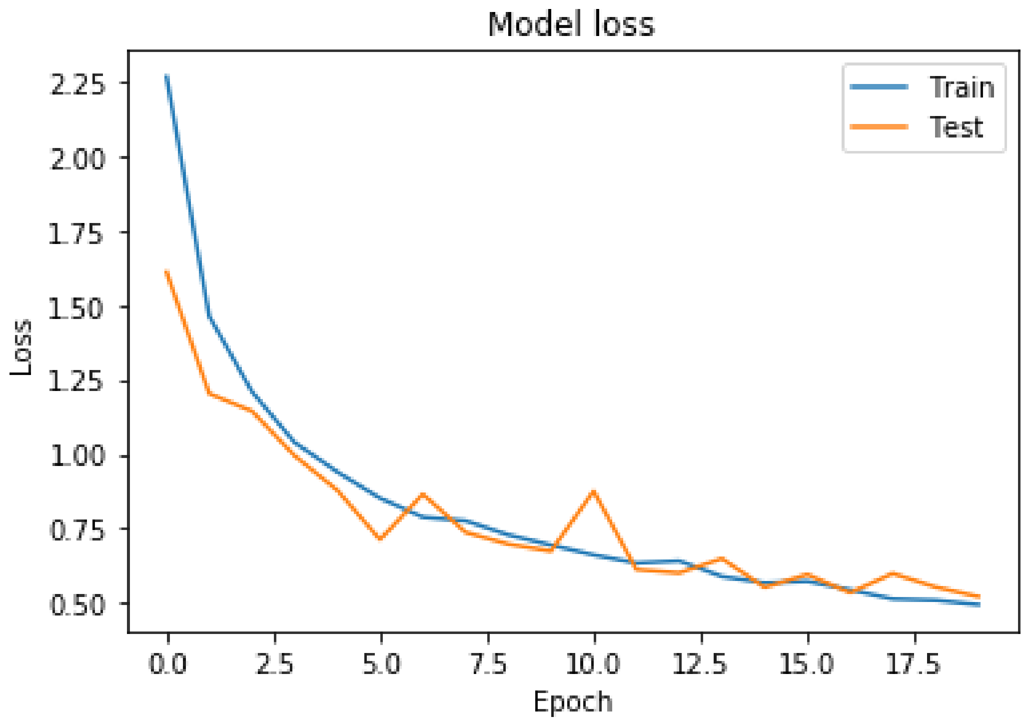
```
_________________________________________________________________
dropout_9 (Dropout)          (None, 3, 27, 32)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 27, 32)         40992
_________________________________________________________________
batch_normalization_9 (Batch (None, 3, 27, 32)         128
_________________________________________________________________
activation_9 (Activation)    (None, 3, 27, 32)         0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 13, 32)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 1, 13, 32)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 416)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                26688
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_10 (Batc (None, 64)                256
_________________________________________________________________
activation_10 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 14)                910
=================================================================
Total params: 152,654
Trainable params: 152,270
Non-trainable params: 384
_________________________________________________________________
Train on 3540 samples, validate on 1180 samples
Epoch 1/20
3540/3540 [==============================] - 50s 14ms/step - loss: 2.2666 - acc:
0.2805 - val_loss: 1.6107 - val_acc: 0.5000
Epoch 2/20
3540/3540 [==============================] - 49s 14ms/step - loss: 1.4623 - acc:
0.5254 - val_loss: 1.2027 - val_acc: 0.6237
Epoch 3/20
3540/3540 [==============================] - 50s 14ms/step - loss: 1.2095 - acc:
0.6017 - val_loss: 1.1439 - val_acc: 0.6305
Epoch 4/20
3540/3540 [==============================] - 53s 15ms/step - loss: 1.0383 - acc:
0.6528 - val_loss: 0.9938 - val_acc: 0.7034
Epoch 5/20
3540/3540 [==============================] - 51s 14ms/step - loss: 0.9398 - acc:
0.6856 - val_loss: 0.8785 - val_acc: 0.7000
Epoch 6/20
```

```
3540/3540 [==============================] - 50s 14ms/step - loss: 0.8517 - acc:
0.7082 - val_loss: 0.7124 - val_acc: 0.7492
Epoch 7/20
3540/3540 [==============================] - 50s 14ms/step - loss: 0.7878 - acc:
0.7350 - val_loss: 0.8651 - val_acc: 0.7212
Epoch 8/20
3540/3540 [==============================] - 50s 14ms/step - loss: 0.7757 - acc:
0.7356 - val_loss: 0.7369 - val_acc: 0.7398
Epoch 9/20
3540/3540 [==============================] - 51s 14ms/step - loss: 0.7284 - acc:
0.7480 - val_loss: 0.6972 - val_acc: 0.7576
Epoch 10/20
3540/3540 [==============================] - 54s 15ms/step - loss: 0.6941 - acc:
0.7621 - val_loss: 0.6737 - val_acc: 0.7771
Epoch 11/20
3540/3540 [==============================] - 55s 16ms/step - loss: 0.6608 - acc:
0.7692 - val_loss: 0.8756 - val_acc: 0.7297
Epoch 12/20
3540/3540 [==============================] - 52s 15ms/step - loss: 0.6335 - acc:
0.7785 - val_loss: 0.6118 - val_acc: 0.7729
Epoch 13/20
3540/3540 [==============================] - 54s 15ms/step - loss: 0.6408 - acc:
0.7703 - val_loss: 0.6004 - val_acc: 0.7729
Epoch 14/20
3540/3540 [==============================] - 55s 15ms/step - loss: 0.5886 - acc:
0.7955 - val_loss: 0.6485 - val_acc: 0.7661
Epoch 15/20
3540/3540 [==============================] - 61s 17ms/step - loss: 0.5674 - acc:
0.8000 - val_loss: 0.5525 - val_acc: 0.7907
Epoch 16/20
3540/3540 [==============================] - 56s 16ms/step - loss: 0.5727 - acc:
0.8062 - val_loss: 0.5953 - val_acc: 0.7661
Epoch 17/20
3540/3540 [==============================] - 54s 15ms/step - loss: 0.5444 - acc:
0.8113 - val_loss: 0.5344 - val_acc: 0.8034
Epoch 18/20
3540/3540 [==============================] - 56s 16ms/step - loss: 0.5135 - acc:
0.8169 - val_loss: 0.5985 - val_acc: 0.7992
Epoch 19/20
3540/3540 [==============================] - 55s 16ms/step - loss: 0.5092 - acc:
0.8184 - val_loss: 0.5535 - val_acc: 0.8000
Epoch 20/20
3540/3540 [==============================] - 55s 16ms/step - loss: 0.4943 - acc:
0.8288 - val_loss: 0.5215 - val_acc: 0.8093
```

[0]:
```
results = get_results(model_history,model,X_test,y_test, ref.labels.unique())
results.create_plot(model_history)
```

```
results.create_results(model)
results.confusion_results(X_test, y_test, ref.labels.unique(), model)
results.print_classification_report(X_test, y_test, ref.labels.unique(), model)
results.accuracy_results_gender(X_test, y_test, ref.labels.unique(), model)
```

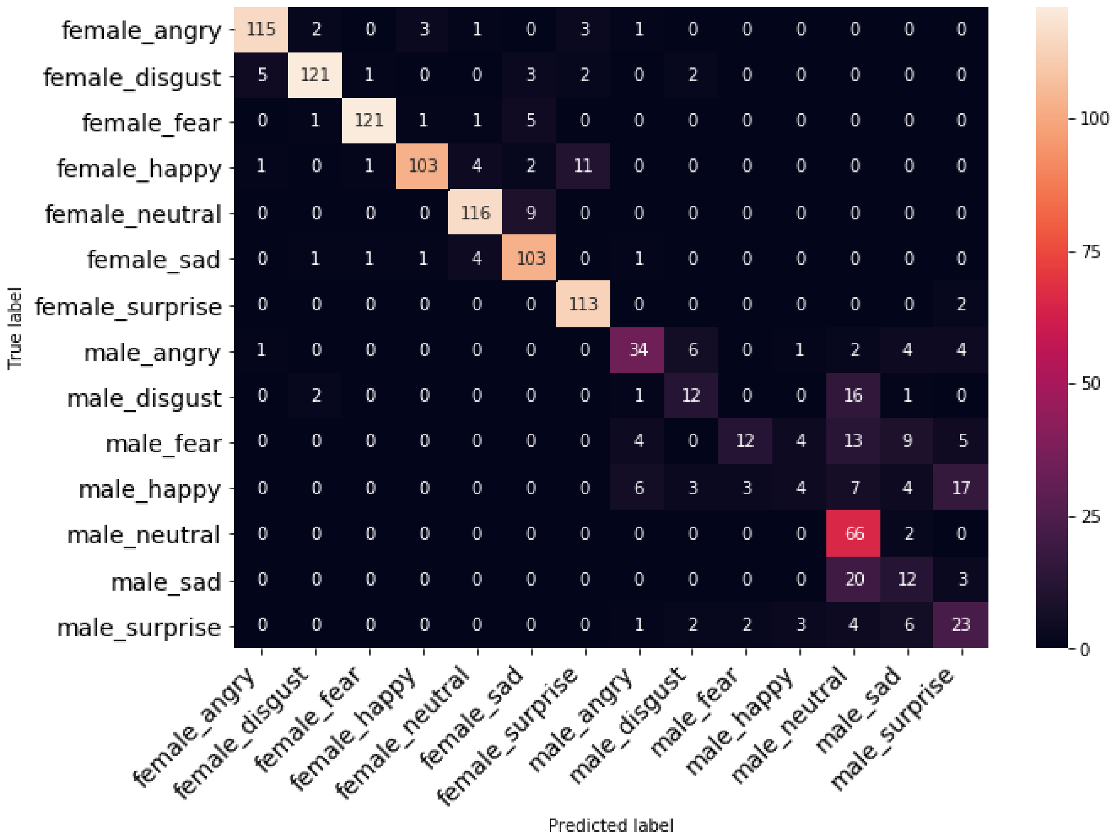




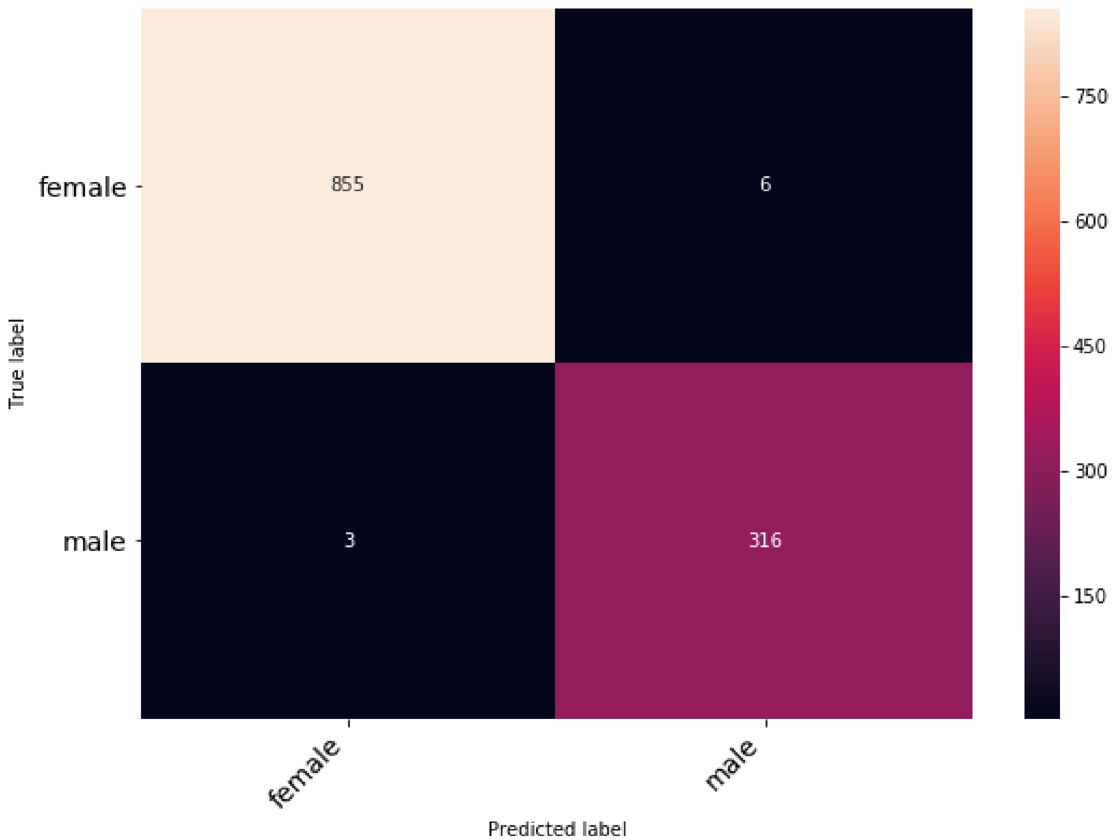
accuracy: 80.93%

	precision	recall	f1-score	support
female_angry	0.94	0.92	0.93	125
female_disgust	0.95	0.90	0.93	134
female_fear	0.98	0.94	0.96	129
female_happy	0.95	0.84	0.90	122
female_neutral	0.92	0.93	0.92	125
female_sad	0.84	0.93	0.88	111
female_surprise	0.88	0.98	0.93	115
male_angry	0.71	0.65	0.68	52
male_disgust	0.48	0.38	0.42	32
male_fear	0.71	0.26	0.37	47
male_happy	0.33	0.09	0.14	44
male_neutral	0.52	0.97	0.67	68
male_sad	0.32	0.34	0.33	35
male_surprise	0.43	0.56	0.48	41
accuracy			0.81	1180
macro avg	0.71	0.69	0.68	1180
weighted avg	0.81	0.81	0.80	1180

0.9923728813559322



True label
Predicted label
female_angry
female_disgust
female_fear
female_happy
female_neutral
female_sad
female_surprise
male_angry
male_disgust
male_fear
male_happy
male_neutral
male_sad
male_surprise
115 2 0 3 1 0 3 1 0 0 0 0 0 0
5 121 1 0 0 3 2 0 2 0 0 0 0 0
0 1 121 1 1 5 0 0 0 0 0 0 0 0
1 0 1 103 4 2 11 0 0 0 0 0 0 0
0 0 0 0 116 9 0 0 0 0 0 0 0 0
0 1 1 1 4 103 0 1 0 0 0 0 0 0
0 0 0 0 0 0 113 0 0 0 0 0 0 2
1 0 0 0 0 0 0 34 6 0 1 2 4 4
0 2 0 0 0 0 0 1 12 0 0 16 1 0
0 0 0 0 0 0 0 4 0 12 4 13 9 5
0 0 0 0 0 0 0 6 3 3 4 7 4 17
0 0 0 0 0 0 0 0 0 0 0 66 2 0
0 0 0 0 0 0 0 0 0 0 0 20 12 3
0 0 0 0 0 0 0 1 2 2 3 4 6 23
100
75
50
25
0



[0]:
```
sampling_rate=44100
audio_duration=2.5
n_melspec = 60
specgram = prepare_data(ref, n = n_melspec, aug = 0, mfcc = 0)
```

```
100%|      | 4720/4720 [02:52<00:00, 27.38it/s]
```

[0]:
```
# Split between train and test
X_train, X_test, y_train, y_test = train_test_split(specgram
                                                    , ref.labels
                                                    , test_size=0.25
                                                    , shuffle=True
                                                    , random_state=42
                                                   )




# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
```

```python
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

# Normalization as per the standard NN process
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std

# Build CNN model
model = get_2d_conv_model(n=n_melspec)
model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    batch_size=16, epochs=20)
```

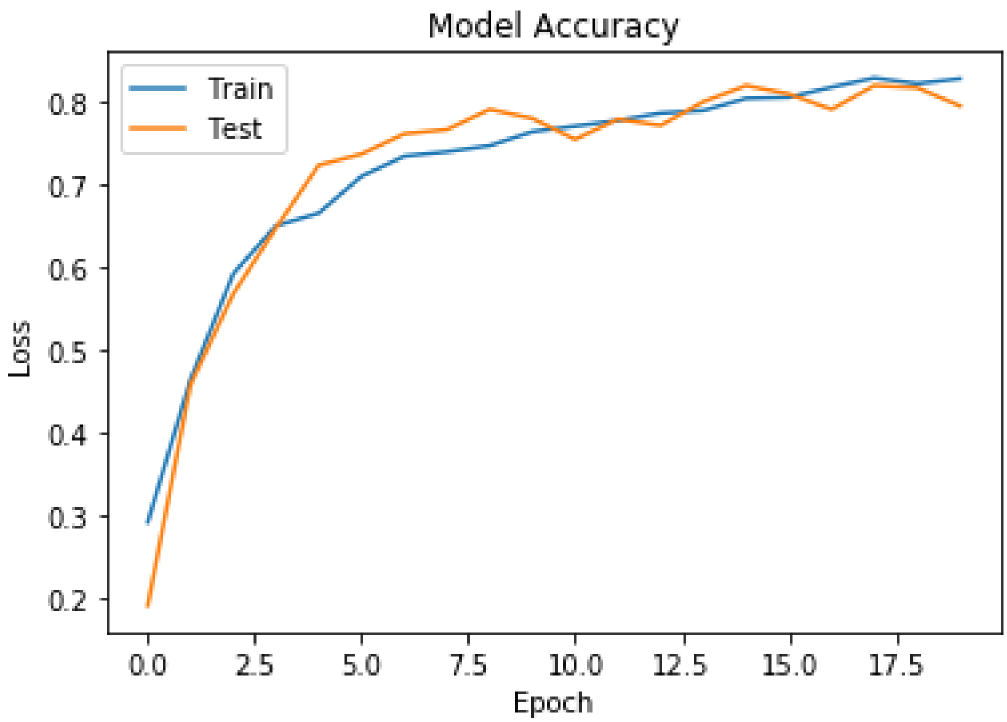
```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 60, 216, 1)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 60, 216, 32)       1312
_________________________________________________________________
batch_normalization_11 (Batc (None, 60, 216, 32)       128
_________________________________________________________________
activation_11 (Activation)   (None, 60, 216, 32)       0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 30, 108, 32)       0
_________________________________________________________________
dropout_13 (Dropout)         (None, 30, 108, 32)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 30, 108, 32)       40992
_________________________________________________________________
batch_normalization_12 (Batc (None, 30, 108, 32)       128
_________________________________________________________________
activation_12 (Activation)   (None, 30, 108, 32)       0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 15, 54, 32)        0
_________________________________________________________________
dropout_14 (Dropout)         (None, 15, 54, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 15, 54, 32)        40992
_________________________________________________________________
batch_normalization_13 (Batc (None, 15, 54, 32)        128
_________________________________________________________________
activation_13 (Activation)   (None, 15, 54, 32)        0
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 27, 32)         0
```

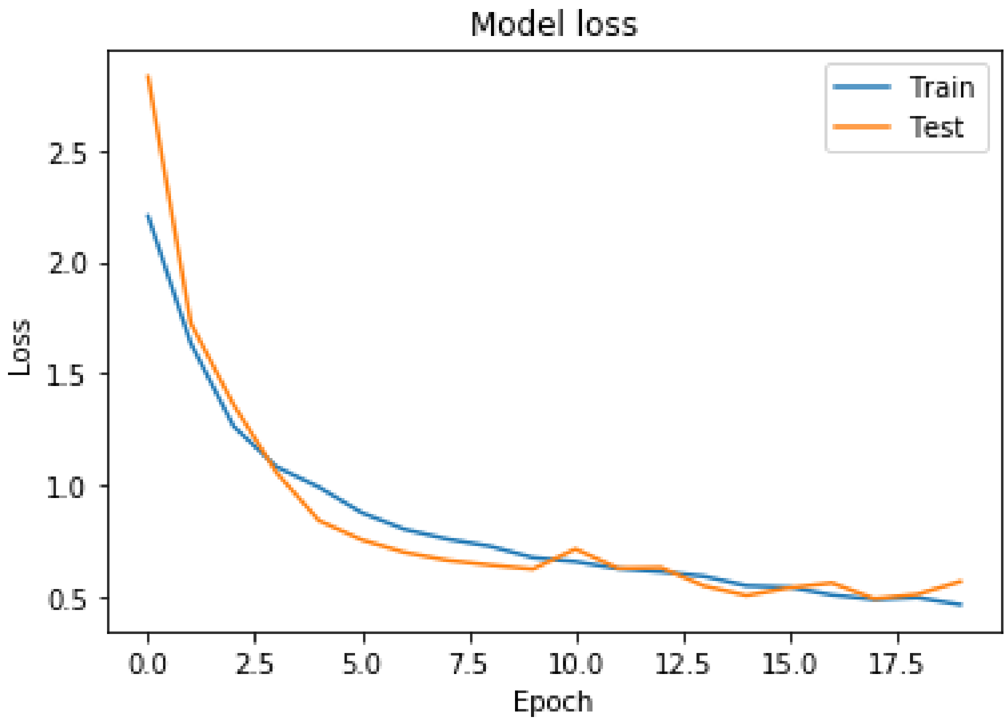
```
_________________________________________________________________
dropout_15 (Dropout)         (None, 7, 27, 32)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 7, 27, 32)         40992
_________________________________________________________________
batch_normalization_14 (Batc (None, 7, 27, 32)         128
_________________________________________________________________
activation_14 (Activation)   (None, 7, 27, 32)         0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 3, 13, 32)         0
_________________________________________________________________
dropout_16 (Dropout)         (None, 3, 13, 32)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1248)              0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                79936
_________________________________________________________________
dropout_17 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_15 (Batc (None, 64)                256
_________________________________________________________________
activation_15 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_18 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 14)                910
=================================================================
Total params: 205,902
Trainable params: 205,518
Non-trainable params: 384
_________________________________________________________________
Train on 3540 samples, validate on 1180 samples
Epoch 1/20
3540/3540 [==============================] - 109s 31ms/step - loss: 2.2047 -
acc: 0.2910 - val_loss: 2.8286 - val_acc: 0.1898
Epoch 2/20
3540/3540 [==============================] - 115s 32ms/step - loss: 1.6345 -
acc: 0.4638 - val_loss: 1.7248 - val_acc: 0.4576
Epoch 3/20
3540/3540 [==============================] - 105s 30ms/step - loss: 1.2641 -
acc: 0.5921 - val_loss: 1.3608 - val_acc: 0.5678
Epoch 4/20
3540/3540 [==============================] - 107s 30ms/step - loss: 1.0831 -
acc: 0.6503 - val_loss: 1.0582 - val_acc: 0.6475
Epoch 5/20
3540/3540 [==============================] - 106s 30ms/step - loss: 0.9918 -
acc: 0.6655 - val_loss: 0.8417 - val_acc: 0.7237
Epoch 6/20
```

```
3540/3540 [==============================] - 105s 30ms/step - loss: 0.8772 -
acc: 0.7102 - val_loss: 0.7557 - val_acc: 0.7373
Epoch 7/20
3540/3540 [==============================] - 105s 30ms/step - loss: 0.8025 -
acc: 0.7347 - val_loss: 0.6992 - val_acc: 0.7619
Epoch 8/20
3540/3540 [==============================] - 105s 30ms/step - loss: 0.7590 -
acc: 0.7401 - val_loss: 0.6640 - val_acc: 0.7669
Epoch 9/20
3540/3540 [==============================] - 108s 31ms/step - loss: 0.7270 -
acc: 0.7475 - val_loss: 0.6427 - val_acc: 0.7915
Epoch 10/20
3540/3540 [==============================] - 106s 30ms/step - loss: 0.6760 -
acc: 0.7644 - val_loss: 0.6247 - val_acc: 0.7805
Epoch 11/20
3540/3540 [==============================] - 104s 30ms/step - loss: 0.6582 -
acc: 0.7712 - val_loss: 0.7143 - val_acc: 0.7551
Epoch 12/20
3540/3540 [==============================] - 109s 31ms/step - loss: 0.6262 -
acc: 0.7780 - val_loss: 0.6282 - val_acc: 0.7797
Epoch 13/20
3540/3540 [==============================] - 104s 29ms/step - loss: 0.6125 -
acc: 0.7870 - val_loss: 0.6314 - val_acc: 0.7720
Epoch 14/20
3540/3540 [==============================] - 106s 30ms/step - loss: 0.5931 -
acc: 0.7901 - val_loss: 0.5500 - val_acc: 0.8008
Epoch 15/20
3540/3540 [==============================] - 108s 31ms/step - loss: 0.5505 -
acc: 0.8048 - val_loss: 0.5070 - val_acc: 0.8203
Epoch 16/20
3540/3540 [==============================] - 105s 30ms/step - loss: 0.5442 -
acc: 0.8062 - val_loss: 0.5414 - val_acc: 0.8102
Epoch 17/20
3540/3540 [==============================] - 105s 30ms/step - loss: 0.5092 -
acc: 0.8184 - val_loss: 0.5630 - val_acc: 0.7915
Epoch 18/20
3540/3540 [==============================] - 108s 30ms/step - loss: 0.4888 -
acc: 0.8294 - val_loss: 0.4923 - val_acc: 0.8203
Epoch 19/20
3540/3540 [==============================] - 106s 30ms/step - loss: 0.4976 -
acc: 0.8226 - val_loss: 0.5126 - val_acc: 0.8178
Epoch 20/20
3540/3540 [==============================] - 108s 30ms/step - loss: 0.4660 -
acc: 0.8285 - val_loss: 0.5693 - val_acc: 0.7958
```

[0]:
```
results = get_results(model_history,model,X_test,y_test, ref.labels.unique())
results.create_plot(model_history)
```

```
results.create_results(model)
results.confusion_results(X_test, y_test, ref.labels.unique(), model)
results.print_classification_report(X_test, y_test, ref.labels.unique(), model)
results.accuracy_results_gender(X_test, y_test, ref.labels.unique(), model)
```

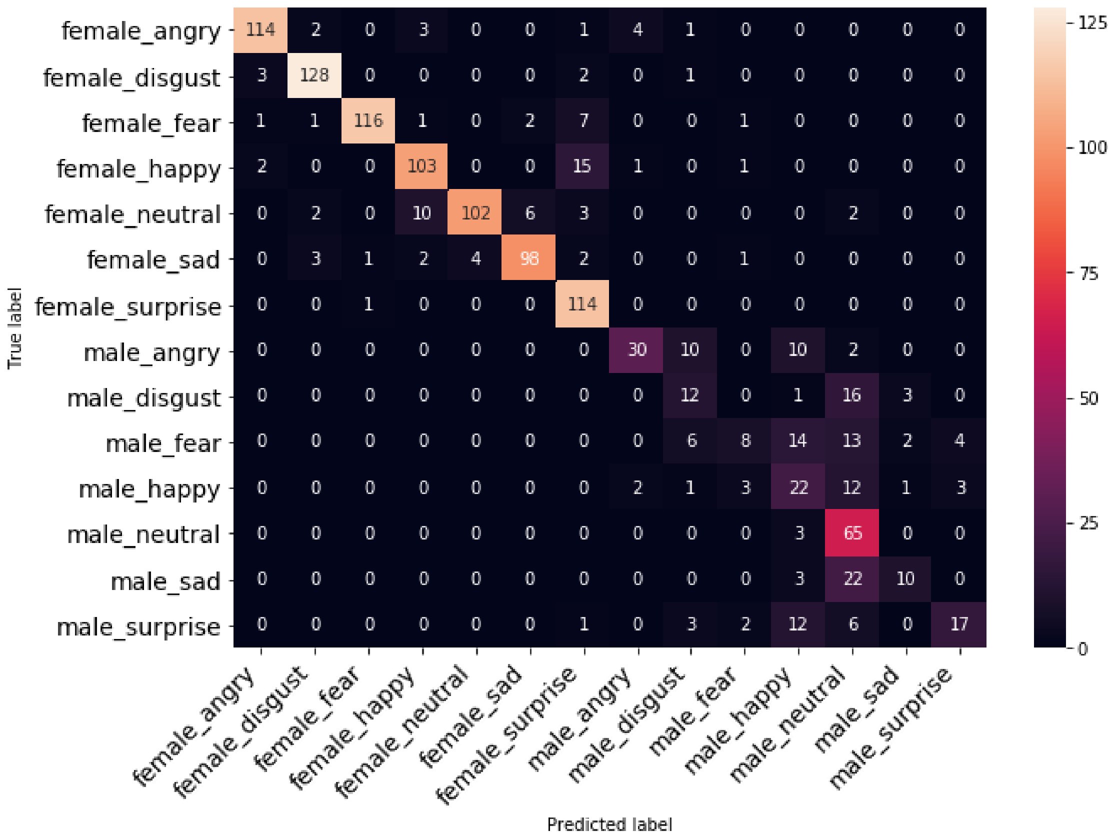


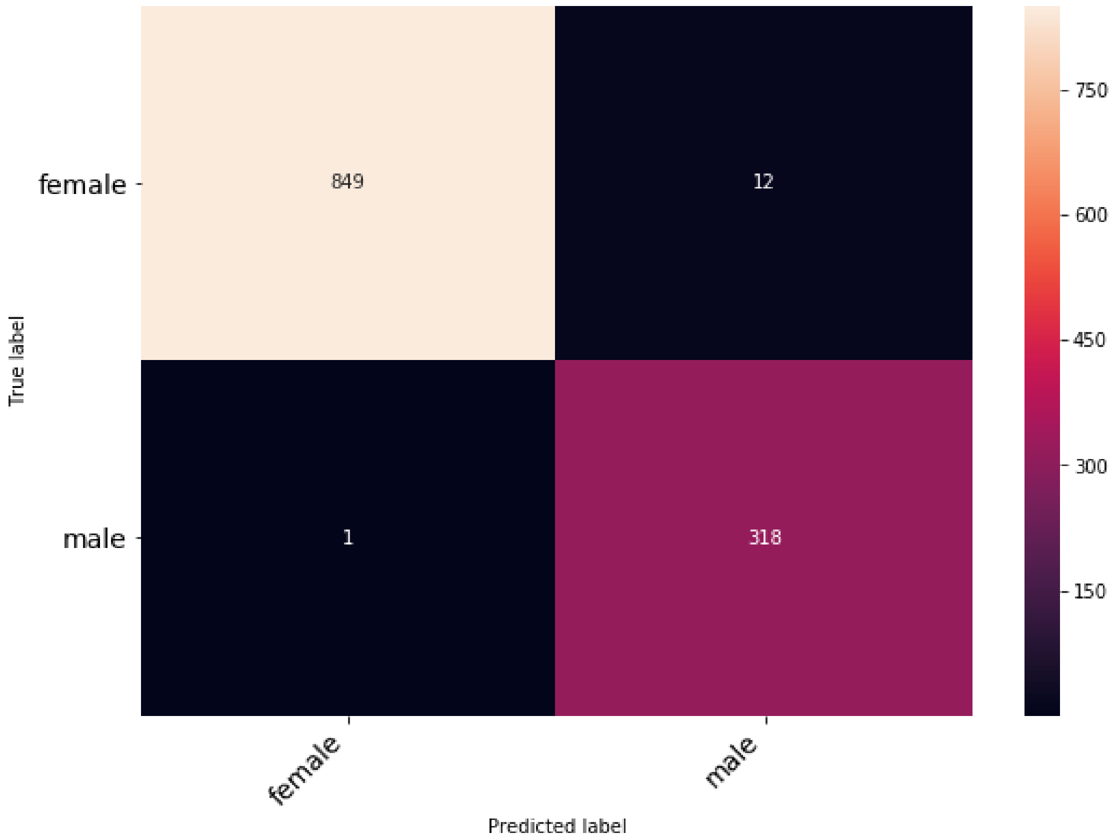


accuracy: 79.58%

	precision	recall	f1-score	support
female_angry	0.95	0.91	0.93	125
female_disgust	0.94	0.96	0.95	134
female_fear	0.98	0.90	0.94	129
female_happy	0.87	0.84	0.85	122
female_neutral	0.96	0.82	0.88	125
female_sad	0.92	0.88	0.90	111
female_surprise	0.79	0.99	0.88	115
male_angry	0.81	0.58	0.67	52
male_disgust	0.35	0.38	0.36	32
male_fear	0.50	0.17	0.25	47
male_happy	0.34	0.50	0.40	44
male_neutral	0.47	0.96	0.63	68
male_sad	0.62	0.29	0.39	35
male_surprise	0.71	0.41	0.52	41
accuracy			0.80	1180
macro avg	0.73	0.68	0.68	1180
weighted avg	0.82	0.80	0.79	1180

0.9889830508474576


True label
Predicted label
female_angry
female_disgust
female_fear
female_happy
female_neutral
female_sad
female_surprise
male_angry
male_disgust
male_fear
male_happy
male_neutral
male_sad
male_surprise
114 2 0 3 0 0 1 4 1 0 0 0 0 0
3 128 0 0 0 0 2 0 1 0 0 0 0 0
1 1 116 1 0 2 7 0 0 1 0 0 0 0
2 0 0 103 0 0 15 1 0 1 0 0 0 0
0 2 0 10 102 6 3 0 0 0 0 2 0 0
0 3 1 2 4 98 2 0 0 1 0 0 0 0
0 0 1 0 0 0 114 0 0 0 0 0 0 0
0 0 0 0 0 0 0 30 10 0 10 2 0 0
0 0 0 0 0 0 0 0 12 0 1 16 3 0
0 0 0 0 0 0 0 0 6 8 14 13 2 4
0 0 0 0 0 0 0 2 1 3 22 12 1 3
0 0 0 0 0 0 0 0 0 0 3 65 0 0
0 0 0 0 0 0 0 0 0 0 3 22 10 0
0 0 0 0 0 0 1 0 3 2 12 6 0 17
125
100
75
50
25
0



[0]:
```
sampling_rate=44100
audio_duration=2.5
n_melspec = 60
aug_specgram = prepare_data(ref,  n = n_melspec, aug = 1, mfcc = 0)
```

```
100%|      | 4720/4720 [03:20<00:00, 23.57it/s]
```

[0]:
```
# Split between train and test
X_train, X_test, y_train, y_test = train_test_split(aug_specgram
                                                     , ref.labels
                                                     , test_size=0.25
                                                     , shuffle=True
                                                     , random_state=42
                                                    )




# one hot encode the target
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
```

```python
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

# Normalization as per the standard NN process
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std

# Build CNN model
model = get_2d_conv_model(n=n_melspec)
model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    batch_size=16, epochs=20)
```

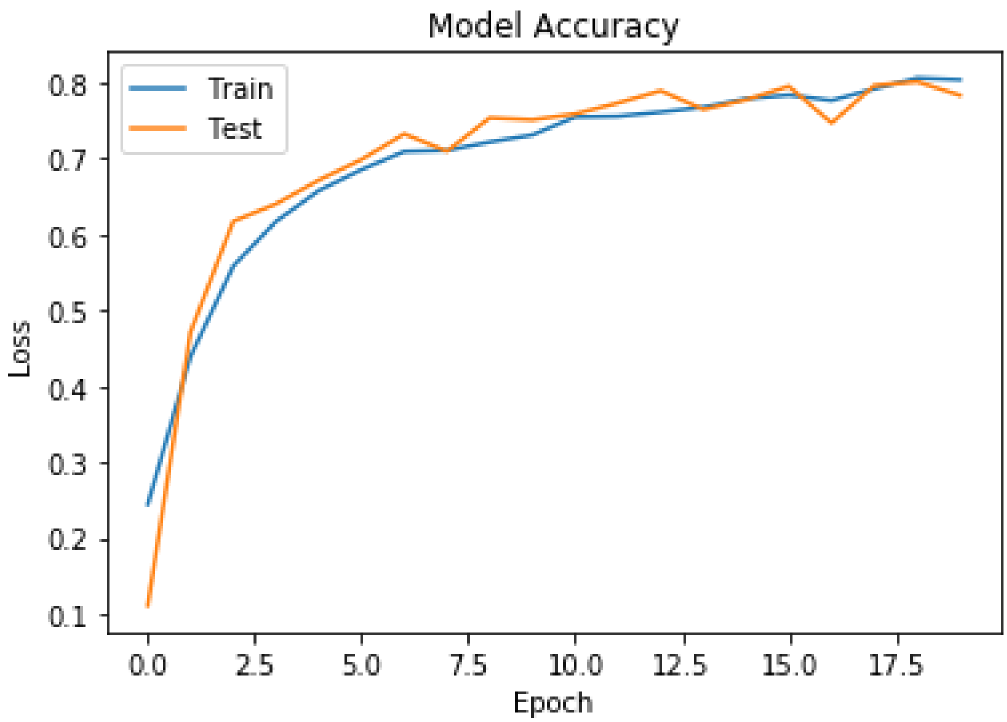
```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 60, 216, 1)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 60, 216, 32)       1312
_________________________________________________________________
batch_normalization_16 (Batc (None, 60, 216, 32)       128
_________________________________________________________________
activation_16 (Activation)   (None, 60, 216, 32)       0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 30, 108, 32)       0
_________________________________________________________________
dropout_19 (Dropout)         (None, 30, 108, 32)       0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 30, 108, 32)       40992
_________________________________________________________________
batch_normalization_17 (Batc (None, 30, 108, 32)       128
_________________________________________________________________
activation_17 (Activation)   (None, 30, 108, 32)       0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 15, 54, 32)        0
_________________________________________________________________
dropout_20 (Dropout)         (None, 15, 54, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 15, 54, 32)        40992
_________________________________________________________________
batch_normalization_18 (Batc (None, 15, 54, 32)        128
_________________________________________________________________
activation_18 (Activation)   (None, 15, 54, 32)        0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 7, 27, 32)         0
```

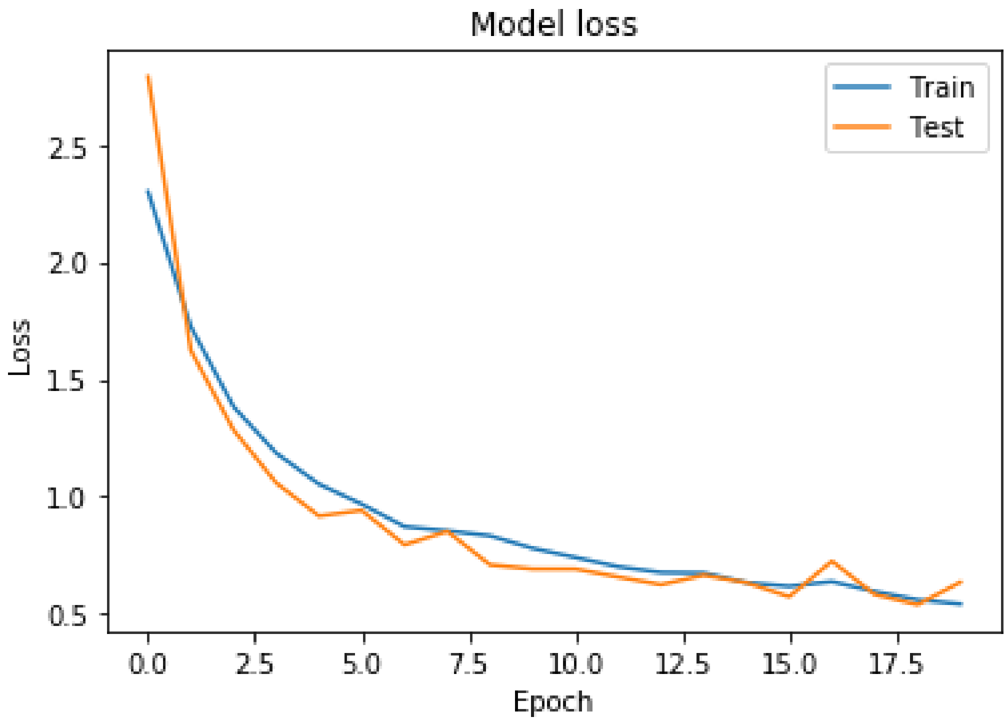
```
_________________________________________________________________
dropout_21 (Dropout)         (None, 7, 27, 32)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 7, 27, 32)         40992
_________________________________________________________________
batch_normalization_19 (Batc (None, 7, 27, 32)         128
_________________________________________________________________
activation_19 (Activation)   (None, 7, 27, 32)         0
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 3, 13, 32)         0
_________________________________________________________________
dropout_22 (Dropout)         (None, 3, 13, 32)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1248)              0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                79936
_________________________________________________________________
dropout_23 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_20 (Batc (None, 64)                256
_________________________________________________________________
activation_20 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_24 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 14)                910
=================================================================
Total params: 205,902
Trainable params: 205,518
Non-trainable params: 384
_________________________________________________________________
Train on 3540 samples, validate on 1180 samples
Epoch 1/20
3540/3540 [==============================] - 111s 31ms/step - loss: 2.2990 -
acc: 0.2444 - val_loss: 2.7894 - val_acc: 0.1110
Epoch 2/20
3540/3540 [==============================] - 102s 29ms/step - loss: 1.7233 -
acc: 0.4390 - val_loss: 1.6219 - val_acc: 0.4712
Epoch 3/20
3540/3540 [==============================] - 110s 31ms/step - loss: 1.3826 -
acc: 0.5579 - val_loss: 1.2831 - val_acc: 0.6169
Epoch 4/20
3540/3540 [==============================] - 113s 32ms/step - loss: 1.1844 -
acc: 0.6167 - val_loss: 1.0577 - val_acc: 0.6398
Epoch 5/20
3540/3540 [==============================] - 114s 32ms/step - loss: 1.0527 -
acc: 0.6573 - val_loss: 0.9167 - val_acc: 0.6712
Epoch 6/20
```

```
3540/3540 [==============================] - 111s 31ms/step - loss: 0.9674 -
acc: 0.6850 - val_loss: 0.9407 - val_acc: 0.6983
Epoch 7/20
3540/3540 [==============================] - 99s 28ms/step - loss: 0.8710 - acc:
0.7090 - val_loss: 0.7938 - val_acc: 0.7322
Epoch 8/20
3540/3540 [==============================] - 102s 29ms/step - loss: 0.8537 -
acc: 0.7105 - val_loss: 0.8513 - val_acc: 0.7093
Epoch 9/20
3540/3540 [==============================] - 116s 33ms/step - loss: 0.8323 -
acc: 0.7215 - val_loss: 0.7081 - val_acc: 0.7534
Epoch 10/20
3540/3540 [==============================] - 110s 31ms/step - loss: 0.7787 -
acc: 0.7308 - val_loss: 0.6906 - val_acc: 0.7508
Epoch 11/20
3540/3540 [==============================] - 107s 30ms/step - loss: 0.7397 -
acc: 0.7545 - val_loss: 0.6903 - val_acc: 0.7585
Epoch 12/20
3540/3540 [==============================] - 117s 33ms/step - loss: 0.7001 -
acc: 0.7554 - val_loss: 0.6568 - val_acc: 0.7729
Epoch 13/20
3540/3540 [==============================] - 109s 31ms/step - loss: 0.6756 -
acc: 0.7610 - val_loss: 0.6243 - val_acc: 0.7890
Epoch 14/20
3540/3540 [==============================] - 103s 29ms/step - loss: 0.6735 -
acc: 0.7672 - val_loss: 0.6645 - val_acc: 0.7644
Epoch 15/20
3540/3540 [==============================] - 105s 30ms/step - loss: 0.6329 -
acc: 0.7785 - val_loss: 0.6326 - val_acc: 0.7771
Epoch 16/20
3540/3540 [==============================] - 104s 29ms/step - loss: 0.6174 -
acc: 0.7833 - val_loss: 0.5731 - val_acc: 0.7949
Epoch 17/20
3540/3540 [==============================] - 110s 31ms/step - loss: 0.6372 -
acc: 0.7763 - val_loss: 0.7237 - val_acc: 0.7466
Epoch 18/20
3540/3540 [==============================] - 107s 30ms/step - loss: 0.5954 -
acc: 0.7915 - val_loss: 0.5828 - val_acc: 0.7958
Epoch 19/20
3540/3540 [==============================] - 106s 30ms/step - loss: 0.5599 -
acc: 0.8056 - val_loss: 0.5390 - val_acc: 0.8008
Epoch 20/20
3540/3540 [==============================] - 104s 29ms/step - loss: 0.5419 -
acc: 0.8034 - val_loss: 0.6344 - val_acc: 0.7831
```

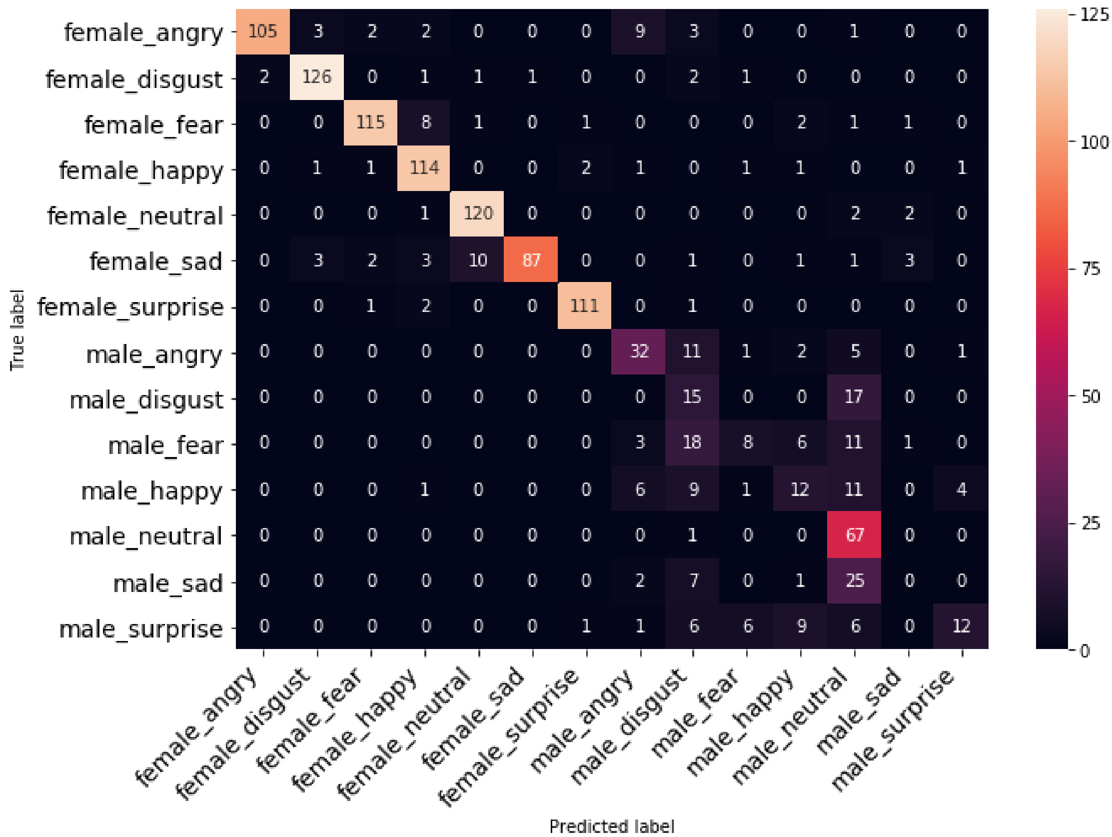
[0]:
```
results = get_results(model_history,model,X_test,y_test, ref.labels.unique())
results.create_plot(model_history)
```

```
results.create_results(model)
results.confusion_results(X_test, y_test, ref.labels.unique(), model)
results.print_classification_report(X_test, y_test, ref.labels.unique(), model)
results.accuracy_results_gender(X_test, y_test, ref.labels.unique(), model)
```

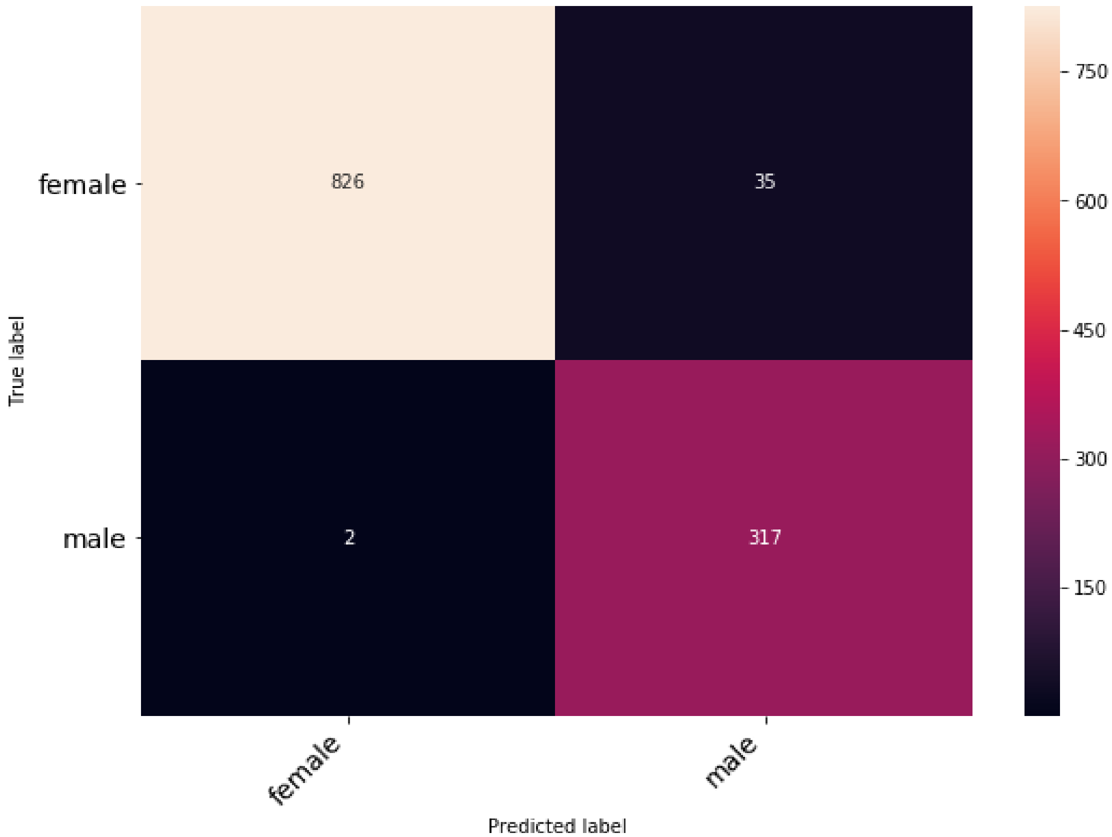





```
accuracy: 78.31%
                 precision    recall  f1-score   support

   female_angry       0.98      0.84      0.91       125
  female_disgust       0.95      0.94      0.94       134
    female_fear       0.95      0.89      0.92       129
   female_happy       0.86      0.93      0.90       122
  female_neutral       0.91      0.96      0.93       125
     female_sad       0.99      0.78      0.87       111
 female_surprise       0.97      0.97      0.97       115
     male_angry       0.59      0.62      0.60        52
   male_disgust       0.20      0.47      0.28        32
      male_fear       0.44      0.17      0.25        47
     male_happy       0.35      0.27      0.31        44
   male_neutral       0.46      0.99      0.62        68
       male_sad       0.00      0.00      0.00        35
  male_surprise       0.67      0.29      0.41        41

       accuracy                           0.78      1180
      macro avg       0.67      0.65      0.64      1180
   weighted avg       0.80      0.78      0.78      1180

0.9686440677966102
```



True label
Predicted label
female_angry
female_disgust
female_fear
female_happy
female_neutral
female_sad
female_surprise
male_angry
male_disgust
male_fear
male_happy
male_neutral
male_sad
male_surprise



Back to Top

Back to Table of Contents

Back to Experiments and Results

```
[0]:
```