Final Report

By: Yash Kasundra (a1838670)

University of Adelaide

Applied Machine Learning

ABSTRACT

This report will provide a brief introduction to the problems I overcame, and the methods and metrics used to develop and evaluate this project. The problem statement for the project is to develop a machine-learning model that can detect emotions from voice or audio signals. This report will have 10 parts: (i) Introduction, (ii) Related works, (iii) Novelty, (iv) Loading Data, (v) Data Visualization, (vi) Data Augmentation, (vii) Feature extraction, (viii) Data Preparation, (ix) Modelling and (x) Evaluation metric.

KEYWORDS

Machine Learning, Neural networks, Speech Emotion Recognition (SER), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Maxpooling, Data Augmentation

1. INTRODUCTION

For this project, my main goal is to create an emotion-detecting model that can perceive emotions from audio signals, whether recorded audio, movie dialogue, or a real-time voice. This is a classification problem where an input sample (Voice signal) needs to be classified into 8 predefined emotions (surprise, anger, calm, disgust, sad, fear, neutral, happy). And to achieve this, I will focus on creating my own model using deep neural networks with convolution layers and maxpooling layers. The basic process is to train the model on my dataset and then fine-tune it for better performance. This report will summarize the whole process that I undertook while creating this project and also the methodology & evaluation metric used in this project.

2. RELATED WORK

In the last decade, deep learning models are gradually replacing traditional machine learning methods and have become the mainstream algorithm in most ML fields. Therefore, several methods have been implemented and studied.

Since there has been an increase in the number of data points or features in audio signal processing domains, many traditional approaches for the classification and analysis of audio signals were developed. The main challenge in this project is Feature selection and classification that can accurately identify the emotional state of the speaker [2][3].

Zhang et al. described a method based on the Alex-Net model for emotion recognition in their work [6]. Liu and co. develop a strategy for spontaneously recalling emotions and used it on the RECOLA natural emotion dataset [5]. This research described an end-to-end LSTM-DNN-based model for statistical emotion evaluation that incorporates fully connected layers and the CNN-LSTM technique to extract significant features from raw data [4].

After features extraction from the voice signals, a few of the popular model architecture that has been used over time are:

- **RNN/LSTMs**
- **Attention-based models**
- **Listen-Attend-Spell (LAS)**

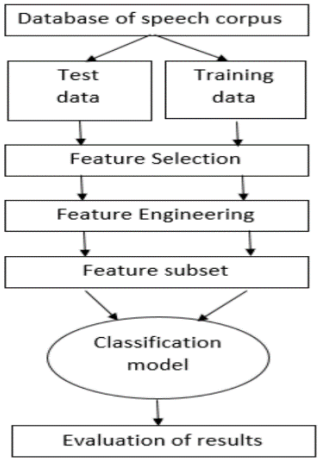


Figure 1: Traditional Machine Learning Approach

3. NOVELTY

There are a lot of prebuilt machine learning models for image classification and regression like VGG-16, ResNet50, Inceptionv3, EfficientNet, etc. And these models were created by different research teams from big tech giants like Google, Facebook, Amazon, etc. But when it comes to audio classification there is little to no work done on this topic. During my initial research I found out that there were no prebuilt models on audio classification and what little I could find was done on traditional neural networks or CNN. But the results were not as great as image classification since creating an optimal CNN takes time and research. You need to test out different architectures and then hyper-tune every parameter to get the best possible outcome. And this is what I undertook when I selected this project, my goal was to create the best CNN network to predict emotions from audio signals.

4. LOADING DATASETS

Like all the big data and ML project mine started out with me hunting for valid datasets to be used for my project. After some research, I decided to use these 4 datasets i.e., Surrey Audio-Visual Expressed Emotion (Savee), Crowd-sourced Emotional Multimodal Actors Dataset (Crema-D), Ryerson Audio-Visual Database of Emotional Speech and Song (Ravdess), Toronto emotional speech set (Tess). All these datasets were in a different format, so my first task was to load all these datasets into different data frames, refer to figure-2 to see the loading of Ravdess into a data frame. To achieve this I had to do some research on loading data from zip files [12] and then changing the labels of the audio signals since all files had different labels [13] for example, some datasets called the signal angry, some labeled it agitated, or irritated. So I had to change them into a single format.

After loading them into a data frame, I changed their formats and merge all these different data frames into a single data frame and classified all these data into 8 different emotions which are Surprise, Angry, Calm, Disgust, Sad, Fear, Neutral, and Happy. You can check figure-3 for the code.

```
In [4]:
ravdess_directory_list = os.listdir(Ravdess)

file_emotion = []
file_path = []
for dir in ravdess_directory_list:
    # as their are 20 different actors in our previous directory we need to extract files for each
actor.
    actor = os.listdir(Ravdess + dir)
    for file in actor:
        part = file.split('.')[0]
        part = part.split('-')
        # third part in each file represents the emotion associated to that file.
        file_emotion.append(int(part[2]))
        file_path.append(Ravdess + dir + '/' + file)

# dataframe for emotion of files
emotion_df = pd.DataFrame(file_emotion, columns=['Emotions'])

# dataframe for path of files.
path_df = pd.DataFrame(file_path, columns=['Path'])
Ravdess_df = pd.concat([emotion_df, path_df], axis=1)

# changing integers to actual emotions.
Ravdess_df.Emotions.replace({1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fear',
7:'disgust', 8:'surprise'}, inplace=True)
Ravdess_df.head()
```

Figure 2: Loading of Ravdess into a data frame

```
In [8]:
# creating Dataframe using all the 4 dataframes we created so far.
data_path = pd.concat([Ravdess_df, Crema_df, Tess_df, Savee_df], axis = 0)
data_path.to_csv("data_path.csv",index=False)
data_path.head()
```

Figure 3: Concating all 4 data frames

5. DATA VISUALIZATION

After getting my data ready, I created few plots to check the number of samples in each emotion, to see if all the emotion had equal number of samples for better performance of our model.

After looking at the graph in figure 4, I found out there were not enough samples for surprise and calm emotions. If we pass this data as it is then it would create an imbalance in the classes which will in turn affect the performance of our model.

```
plt.title('Count of Emotions', size=16)
sns.countplot(data_path.Emotions)
plt.ylabel('Count', size=12)
plt.xlabel('Emotions', size=12)
sns.despine(top=True, right=True, left=False, bottom=False)
plt.show()
```

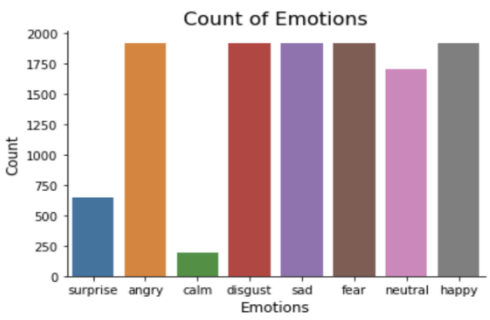


Figure 4: Emotion Bar-Graph

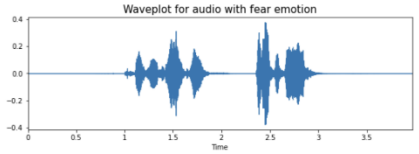
Now I had 2 options either look for another dataset that has these emotions or upsample the ones I got. I'll discuss this issue in detail in the next part.

I also plotted a few wavelength plots and spectrogram plots for all the emotions to familiarize myself with the difference in frequency, pitch, and noise for all different emotions. For example, anger and fear have a high pitch while calm and happiness have a low pitch. Figure 5 shows an example of the plot for fear.

Definition of wave plot and spectrogram:

- Wave plots - Wave plots let us know the loudness of the audio at a given time.
- Spectrograms - A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to time for given audio/music signals.

```
In [11]:
emotion='fear'
path = np.array(data_path.Path[data_path.Emotions==emotion])[1]
data, sampling_rate = librosa.load(path)
create_waveplot(data, sampling_rate, emotion)
create_spectrogram(data, sampling_rate, emotion)
Audio(path)
```



Out[11]:

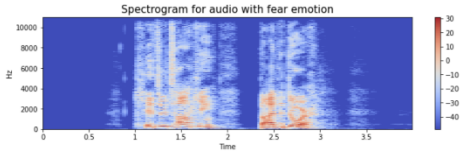


Figure 5: Wave plot and Spectrogram Plot for Fear

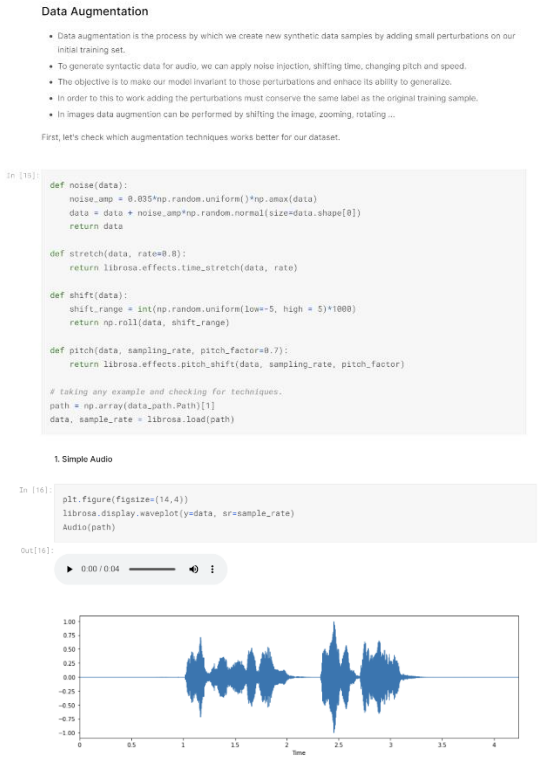
6. DATA AUGMENTATION

Coming back to the issue of having low samples for surprise and calm emotion, I looked for different dataset but there were none with these emotions. Since I was already using the best 4 dataset for audio signals with emotions. So, my only option was to up sample them using data augmentation methods. Refer these links for more details [14][15][16][17][18].

This was my first roadblock since I had never worked with audio signals, and I didn't have any clue about how to upsample audio signals. Up-sampling photos were pretty easy considering we can crop them, change the hue, rotate the image, and many other methods. But for the audio signal, it is very different and there are very few resources out there that are really helpful. After an intense week of research and gathering information from all the links, I mentioned above. I found out that data augmentation is:

- By adding small perturbations to our training set, we can create new artificial data points and this process is called data augmentation or up-sampling.
- We can use changing speed and pitch, noise injection, and shifting time to generate artificial data for audio.
- In order for this to work adding the noise or changing pitch must preserve the label of the original training sample.

But I still had to choose which augmentation techniques to work best for my dataset. So created a function from scratch and then plotted wave plot and spectrogram plot to find a difference in the techniques used. The below Images shows different augmentation example.



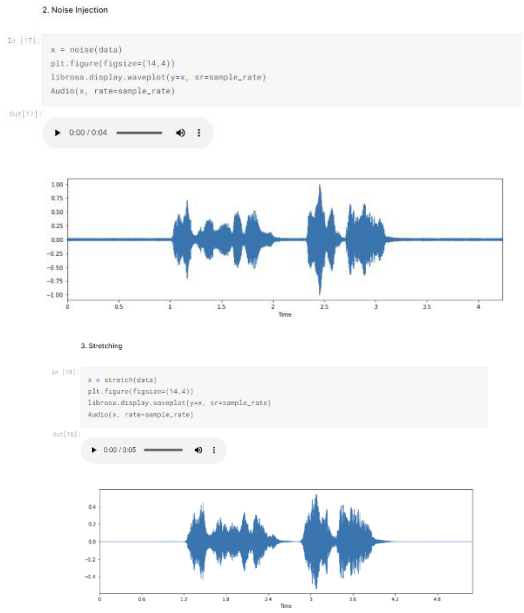


Figure 6: Code and wave plot for different data augmentation techniques

After checking the impact of all the different augmentation techniques, I decided to use noise, stretching (i.e. changing speed), and some pitching.

7. FEATURE EXTRACTION

Now, I had enough samples for all emotions after augmentation. The next step was Feature extraction, this is a very important part of analyzing and finding relations between different things. As we already know that the data provided by audio cannot be understood by the models directly so we need to convert them into an understandable format for which feature extraction is used. But again since I never worked with audio signals I didn't know how to tackle this problem of feature extraction.

Just like data augmentation, I had to go through old papers and do some digging to know different feature extraction techniques. It took me an entire week to gather these details from all these different articles and blogs [19][20][21][22] [23], as shown here in figure 7.

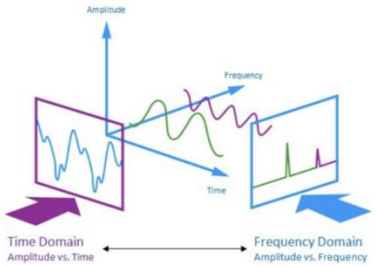
After this I created different functions to implement these features and a basic neural network, tested the impact of different features, and then decided to use these 5 features for my final model:

- MelSpectogram
- MFCC
- RMS (root mean square) value
- Chroma_stft
- Zero Crossing Rate

Feature Extraction

- Extraction of features is a very important part in analyzing and finding relations between different things. As we already know that the data provided of audio cannot be understood by the models directly so we need to convert them into an understandable format for which feature extraction is used.

The audio signal is a three-dimensional signal in which three axes represent time, amplitude and frequency.



I am no expert on audio signals and feature extraction on audio files so i need to search and found a very good blog written by Askash Mallik on feature extraction.

As stated there with the help of the sample rate and the sample data, one can perform several transformations on it to extract valuable features out of it.

1. Zero Crossing Rate : The rate of sign-changes of the signal during the duration of a particular frame.
2. Energy : The sum of squares of the signal values, normalized by the respective frame length.
3. Entropy of Energy : The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
4. Spectral Centroid : The center of gravity of the spectrum.
5. Spectral Spread : The second central moment of the spectrum.
6. Spectral Entropy : Entropy of the normalized spectral energies for a set of sub-frames.
7. Spectral Flux : The squared difference between the normalized magnitudes of the spectra of the two successive frames.
8. Spectral Rolloff : The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
9. MFCCs Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.
10. Chroma Vector : A 12-element representation of the spectral energy where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).
11. Chroma Deviation : The standard deviation of the 12 chroma coefficients.

In this project i am not going deep in feature selection process to check which features are good for our dataset rather i am only extracting 5 features:

- Zero Crossing Rate
- Chroma_stft
- MFCC
- RMS(root mean square) value
- MelSpectogram to train our model.

Figure 7: Different Feature for voice signals

Implementing all these features from scratch as well as testing them on basic neural network took me almost a week as well. This was probably the hardest part in my project along with data augmentation. Since model creation and evaluation are pretty easy, since I've done those a lot of times.

If you passed the audio signal directly into the CNN model it would not be able to comprehend the meaning and relations with other audio signals and thus would fail to make predictions. But if you pass numbers as shown in figure-8 below, the audio signals have been transformed into numbers that models can understand and find relationships among the audio signal. Thus, we achieved what we wanted at this stage.

```
In [24]:
Features = pd.DataFrame(X)
Features['labels'] = Y
Features.to_csv('features.csv', index=False)
Features.head()
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...
0	0.185239	0.585543	0.541992	0.555859	0.615102	0.599604	0.652054	0.691854	0.766230	0.791168	...
1	0.302097	0.748427	0.716290	0.740596	0.802801	0.760048	0.693101	0.699719	0.734826	0.753985	...
2	0.147298	0.646143	0.595935	0.561826	0.547853	0.612391	0.561209	0.622703	0.689758	0.756473	...
3	0.199350	0.517106	0.521565	0.508298	0.564973	0.626469	0.698655	0.668579	0.603630	0.621905	...
4	0.296762	0.653405	0.640598	0.633179	0.681640	0.741104	0.730208	0.660096	0.651581	0.663689	...

5 rows × 163 columns

Figure 8: Output of Feature extraction

```
In [21]:
def extract_features(data):
    # ZCR
    result = np.array([])
    zcr = np.mean(librosa.feature.zero_crossing_rate(y=data).T, axis=0)
    result=np.hstack((result, zcr)) # stacking horizontally

    # Chroma_stft
    stft = np.abs(librosa.stft(data))
    chroma_stft = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
    result = np.hstack((result, chroma_stft)) # stacking horizontally

    # MFCC
    mfcc = np.mean(librosa.feature.mfcc(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mfcc)) # stacking horizontally

    # Root Mean Square Value
    rms = np.mean(librosa.feature.rms(y=data).T, axis=0)
    result = np.hstack((result, rms)) # stacking horizontally

    # MelSpectogram
    mel = np.mean(librosa.feature.melspectrogram(y=data, sr=sample_rate).T, axis=0)
    result = np.hstack((result, mel)) # stacking horizontally

    return result

def get_features(path):
    # duration and offset are used to take care of the no audio in start and the ending of each au
dio files as seen above.
    data, sample_rate = librosa.load(path, duration=2.5, offset=0.6)

    # without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    # data with noise
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    # data with stretching and pitching
    new_data = stretch(data)
    data_stretch_pitch = pitch(new_data, sample_rate)
    res3 = extract_features(data_stretch_pitch)
    result = np.vstack((result, res3)) # stacking vertically

    return result
```

Figure 9: Code for selecting chosen 5 Features

8. DATA PREPARATION

Now, I had some raw data that needed to be normalized and thus I used oneHotEncoder.
We use oneHotEncoder on data that has no relation with each other and also because ML algo will give more weightage to larger numbers than lower numbers and this might not be true in every sense. Simply it means that if a feature is represented by that column, it receives a 1. Otherwise, it receives a 0 [7]. Just as shown below:

Type
AA
AB
CD
AA

Onehot encoding →

Type	AA_Onehot	AB_Onehot	CD_Onehot
AA	1	0	0
AB	0	1	0
CD	0	0	1
AA	0	0	0

Figure 10: One-Hot-Encoding example [7]

After normalizing my data with the help of one_hot_encoding, it was time to create train and test data frames. So I split my current data into a 75-25% ratio where 75% was my training set and 25% was my testing data. Then I applied the StandardScaler method to remove the mean and scale each feature to unit variance. And the main reason for using this method is since all Features are different their worth to the model is also different. Some Features might contribute more towards the success of the model, while some might contribute less. Thus each feature is measured at different scales which might end up creating a bias. To deal with such kind of problem, feature-wide standardization (μ=0, σ=1) is used prior to model fitting [8].

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Figure 11: Mathematical Formula for standardization [8]

After expanding the data to match the input size of our model, it was time to create the deep neural network. Figure-12 shows the implementation of all the steps mentioned in data preparation.

Figure 12: Implementation of Data Preparation

9. MODELING

We have our data ready to be passed into our model, so it was time to start creating different architecture of CNN and other models to compare their performance. Here is a little detail about CNN's few layers:

Convolution Layer: The convolution layer handles the main computation of the network and thus it is the core building block of CNN. The basic working of the Convolutional layer is that it performs a dot product between 2 matrices, where one matrix is a restricted portion of the data and the other is the set of learnable parameters sometimes known as kernel [9].

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Figure 13: Convolution layer formula

Here is an example of how the convolution layer works, it takes a subset of the feature transforms it based on the kernel and the convolutional formula, and creates a new output as shown below:

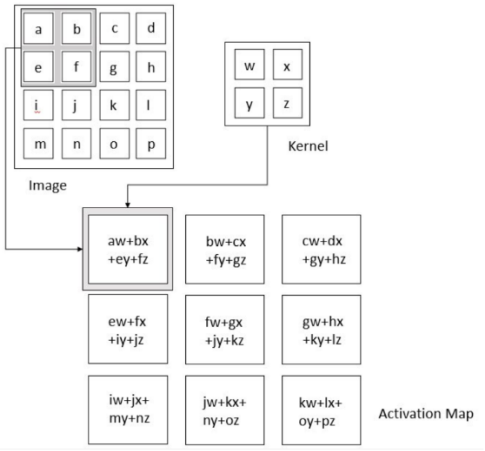


Figure 14: Basic Working of CNN

Pooling Layer: Based on the nearby outputs, the pooling layer changes the output of the network at a certain location based on the pooling layer used. This helps in reducing the required number of computations and weights by reducing the spatial size of the representation. The pooling operation is processed on every piece of the representation separately.

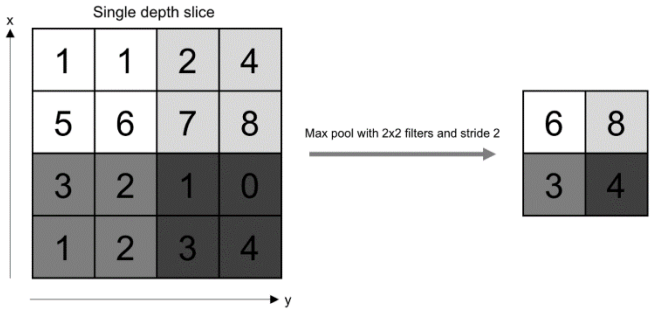


Figure 15: Working of Pooling layer

$$W_{out} = \frac{W - F}{S} + 1$$

Figure 16: Formula for Pooling layer

There are many pooling functions to choose from such as weighted average based on distance from the center, mean of the neighborhood, and L2 norm of the rectangular neighborhood. However, the most popular process is taking

the maximum output from the neighborhood which is also called max pooling [10].

The below Image shows the implementation of the basic LSTM model. After implementing that model, it was tested on a subset of the testing set and I found that it was underfitting the data so I discarded this model.

```
In [13]:
def build_model(input_shape):
    model = tf.keras.Sequential()

    model.add(LSTM(128, input_shape=input_shape, return_sequences=True))
    model.add(LSTM(64))

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))

    model.add(Dense(6, activation='softmax'))

    return model
```

```
In [14]:
# create network
input_shape = (None,13)
model = build_model(input_shape)

# compile model
optimiser = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimiser,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Figure 17: LSTM model implementation

```
# New model
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.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)
```

Figure 18: CNN architecture 1

Both Figure-18,19 shows different CNN architectures I implemented, again it was tested on a subset of the testing dataset and results were a disappointment. Since both these models were overfitting on the training data. Now I had to think about the usage of CNN and the reason behind these models' overfitting. After testing it out a bit, I found out that I made this architecture a little too complex, and thus I decreased the number of layers in my final model.

```
In [46]:
model = models.Sequential()
model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu",
                        input_shape=(X_train.shape[1], 1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(512, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=5, strides=1,
                        padding="same", activation="relu"))
model.add(layers.BatchNormalization())
model.add(layers.MaxPool1D(pool_size=5, strides=2, padding="same"))

model.add(layers.Conv1D(256, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(layers.Conv1D(128, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling1D(pool_size=3, strides = 2, padding = 'same'))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(7, activation="softmax"))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["acc", f1_m])
```

Figure 19: CNN architecture 2

```
In [59]:
model=Sequential()
model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu', input_shape=
(x_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(256, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Conv1D(128, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))
model.add(Dropout(0.2))

model.add(Conv1D(64, kernel_size=5, strides=1, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5, strides = 2, padding = 'same'))

model.add(Flatten())
model.add(Dense(units=32, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(units=8, activation='softmax'))
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

model.summary()
```

Figure 20: Final model implementation

Along with the model, I also used ReduceLROnPlateau from the sklearn library. The main reason is that if we have a high learning rate then we would miss the global minima and in the worst-case scenario, our model might diverge from the optimal solution. While if the learning rate is too small, then we will be stuck on a local minima [24].

The function of ReduceLROnPlateau is to track the performance of the model and then reduce the learning rate when the model stops improving for a given number of epochs. The logic is that model has approached a sub-optimal solution with the current learning rate and is oscillating around the global minima. To make the model take smaller steps toward the optimal solution of the cost function we need to reduce its learning rate [24].

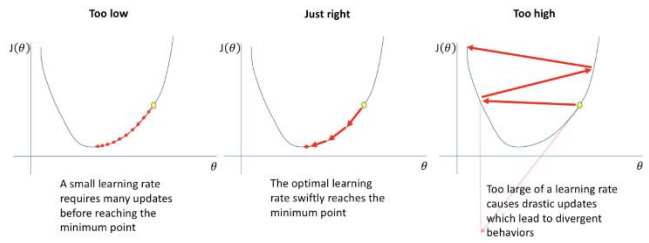


Figure 21: Different Learning rates impact

10. Evaluation Metric

If we talk about classification problem, the most common metrics used are:

I. **Accuracy:** It is the most straight forward metric used in Machine Learning. It basically defines how accurate the model is. For example, if the model classifies 90 of the samples from 100 total accurately then it has 0.9 or 90% accuracy.

II. **Precision:** Precision is the ratio of true positives to the total predicted positive observations [11]. Thus, having a high precision relates to low false positive rates. Check figure 10 for more info. There are 2 ways to compute precision for the multiclass problem:

- Macro averaged precision: calculate precision for all classes separately and then take their average [11].
- Micro averaged precision: calculate class-wise true positive and false positive and then use that to calculate overall precision [11].

Some terminology for evaluation metrics are:

True Positive (TP) – These are correctly predicted values which means the actual class was yes and the predicted class was also yes [11].

False Positive (FP) – These are predicted incorrectly i.e. the actual class is no but it is predicted as yes [11].

True Negative (TN) – These are correctly predicted negatives which means the actual class was no and the predicted class was also no [11].

False Negative (FN) – The actual class was yes and the predicted class was no [11].

III. **Recall:** It is also known as sensitivity because it is the ratio of correctly predicted positive observations to all the observations in the actual class [11]. Check figure 10. Similar to precision, we can calculate recall in 2 ways Macro and Micro for multi-class problems.

IV. **F1 Score:** The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1 score is a metric that combines both precision and recall [11]. It is defined as a simple weighted average (harmonic mean) of precision and recall. Check out figure 10 for the mathematical representation of the F1-score. And there are 2 ways to calculate the F1 score for multiclass problems i.e.

- **Macro averaged F1 Score**: calculate the f1 score of every class and then average them [11].
- **Micro averaged F1 Score**: calculate the macro-averaged precision score and macro-averaged recall score and then take their harmonic mean [11].

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{F1 score} = \frac{2 * (\text{precision} * \text{recall})}{\text{precision} + \text{recall}}$$

Figure 22: Confusion Matrix and Mathematical representation of Precision, Recall, and F1-Score

The below Images show the predictions made by the final model as well as the confusion matrix and classification report.

```
In [53]:
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
y_pred

2022-07-01 15:53:31.286309: W tensorflow/core/framework/cpu_allocator_impl.cc:80] Allocation
of 92473920 exceeds 10% of free system memory.

Out[53]:
array([3, 3, 0, ..., 4, 4, 3])

In [54]:
y_check=np.argmax(y_test,axis=1)
y_check

Out[54]:
array([3, 3, 0, ..., 4, 4, 3])
```

Figure 23: Prediction Done by the final model

```
In [64]:
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (12, 10))
cm = pd.DataFrame(cm , index = [i for i in encoder.categories_] , columns = [i for i in encoder.
categories_])
sns.heatmap(cm, linecolor='white', cmap='Blues', linewidth=1, annot=True, fmt='')
plt.title('Confusion Matrix', size=20)
plt.xlabel('Predicted Labels', size=14)
plt.ylabel('Actual Labels', size=14)
plt.show()
```

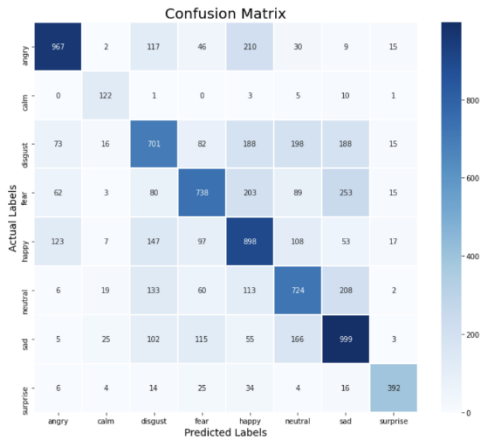


Figure 24: Confusion matrix

```
In [57]:
print(f'Model Confusion Matrix\n',classification_report(y_check,y_pred,target_names=emotion_name
s))

Model Confusion Matrix
              precision    recall  f1-score   support

     disgust       0.92      0.95      0.94      1549
       happy       0.96      0.88      0.92      1537
         sad       0.92      0.91      0.92      1548
     neutral       0.85      0.94      0.89      1511
        fear       0.91      0.93      0.92      1485
       angry       0.96      0.90      0.93      1551
    surprise       0.98      0.95      0.97       549

    accuracy                           0.92      9730
   macro avg       0.93      0.93      0.93      9730
weighted avg       0.92      0.92      0.92      9730
```

Figure-25: Classification report

As you can see from above figure-25, I was able to achieve around 92% F1-score. And this result is way better then traditional approaches which was only able to obtain 60-70% F1 score. Thus, in the end, I was able to achieve my desired output and was able to create an optimal CNN network that can be used to get better results with audio signals. During this project I learned many important skills related to machine learning and life in general, some of them were time management, and the impact of good planning and research, now I have a solid knowledge about data augmentation of the audio signal as well as extraction of features from the audio signal. This project helped me revise many important topics like plotting graphs & charts, data loading complex files, CNN, micro-managing hyperparameters, and last but not least evaluating models based on loss curves, F1-scores, and confusion matrix.

REFERENCES

[1] Middleton, M., 2022. Deep Learning vs. Machine Learning — What's the Difference? | Flatiron School. Flatiron School. Available at: https://flatironschool.com/blog/deep-learning-vs-machine-learning/

[2] S.R. Ashokkumar, G. MohanBabu
Extreme learning adaptive neuro-fuzzy inference system model for classifying the epilepsy using Q-Tuned wavelet transform
J. Intell. Fuzzy Syst., 39 (1) (2020), pp. 233-248

[3] M. Premkumar, T.V.P. Sundararajan
Defense countermeasures for DoS attacks in WSNs using deep radial basis networks
Wireless Pers. Commun., 120 (4) (2021), pp. 2545-2560

[4] A. Ganapathy
Speech emotion recognition using deep learning techniques
ABC J. Adv. Res., 5 (2) (2016), pp. 113-122

[5] M. Premkumar, T.V.P. Sundararajan, K.V. Kumar
Various defense countermeasures against DoS attacks in wireless sensor networks
Int. J. Sci. Technol. Res., 8 (10) (2019), pp. 2926-2935

[6] M. Premkumar, M. Kathiravan, R. Thirukkumaran
Efficient broadcast authentication using TSG algorithm for WSN
Int. J. Comput. Appl., 58 (4) (2012), pp. 34-39

[7] scikit-learn.2022. sklearn.preprocessing.OneHotEncoder https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[8] Loukas, S., 2022. How Scikit-Learn's StandardScaler works. Medium. https://towardsdatascience.com/how-and-why-to-standardize-your-data-996926c2c832#:~:text=StandardScaler%20removes%20the%20mean%20and,standard%20deviation%20of%20each%20feature

[9] Brownlee, J.,2022. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[10] Le, J., 2022. Convolutional Neural Networks: The Biologically-Inspired Model | Codementor. Codementor.io. https://www.codementor.io/@james_aka_yale/convolutional-neural-networks-the-biologically-inspired-model-iq6s48zms

[11] T, B., 2022. Comprehensive Guide on Multiclass Classification Metrics Medium https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd

[12] Working with ZIP files in python (2021) GeeksforGeeks. https://www.geeksforgeeks.org/working-zip-files-python/

[13] How to change column labels in pandas dataframe? (no date) How to Change Column Labels in Pandas DataFrame? - Python Examples. https://pythonexamples.org/pandas-dataframe-change-rename-column-labels/

[14] Han, S & Lee, J 2022, 'NU-Wave 2: A General Neural Audio Upsampling Model for Various Sampling Rates', arXiv.org.

[15] Lu, R, Duan, Z & Zhang, C 2017, 'Metric learning based data augmentation for environmental sound classification', in 2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), vol. 2017-, IEEE.

[16] Gemmeke, JF, Vuegen, L, Karsmakers, P, Vanrumste, B & Van hamme, H 2013, 'An exemplar-based NMF approach to audio event detection', in 2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, IEEE.

[17] Ma, E. (2019) Data Augmentation for Audio, Medium. Medium. https://medium.com/@makcedward/data-augmentation-for-audio-76912b01fdf6

[18] Augmentationmode (no date) Augment audio data - MATLAB. https://www.mathworks.com/help/audio/ref/audiodataaugmenter.html

[19] Mallik, A. (2020) Audio signal feature extraction and clustering, Medium. Project Heuristics. https://medium.com/heuristics/audio-signal-feature-extraction-and-clustering-935319d2225
[20] Sharma, G, Umapathy, K & Krishnan, S 2020, 'Trends in audio signal feature extraction methods', Applied Acoustics, vol. 158, p. 107020–.
[21] Ghoraani, B & Krishnan, S 2011, 'Time-Frequency Matrix Feature Extraction and Classification of Environmental Audio Signals', IEEE Transactions on Audio, Speech, and Language Processing, vol. 19, no. 7, pp. 2197–2209.
[22] Kobayashi, T, Kubota, A & Suzuki, Y 2018, 'Audio Feature Extraction Based on Sub-Band Signal Correlations for Music Genre Classification', in 2018 IEEE International Symposium on Multimedia (ISM), IEEE
[23] Ludeña-Choez, J & Gallardo-Antolín, A 2015, 'Feature extraction based on the high-pass filtering of audio signals for Acoustic Event Classification', Computer Speech & Language, vol. 30, no. 1
[24] Why models often benefit from reducing the learning rate during training, Stack Overflow. https://stackoverflow.com/questions/65869114/why-models-often-benefit-from-reducing-the-learning-rate-during-training#:~:text=ReduceLROnPlateau%20purpose%20is%20to%20track,oscillate%20around%20the%20global%20minimum.