Voice Recognition

Team Members

Garvit Gupta	Paridhi Jain
Geet Gopal Asati	Khushi Tomar

Problem Statement

This project is based on a voice recognition model. Our primary goal would be to recognize the person speaking the words rather than the words spoken. Speaker identification would determine if a registered speaker provides a given utterance and Speaker verification would declare the identity claim of the speaker.

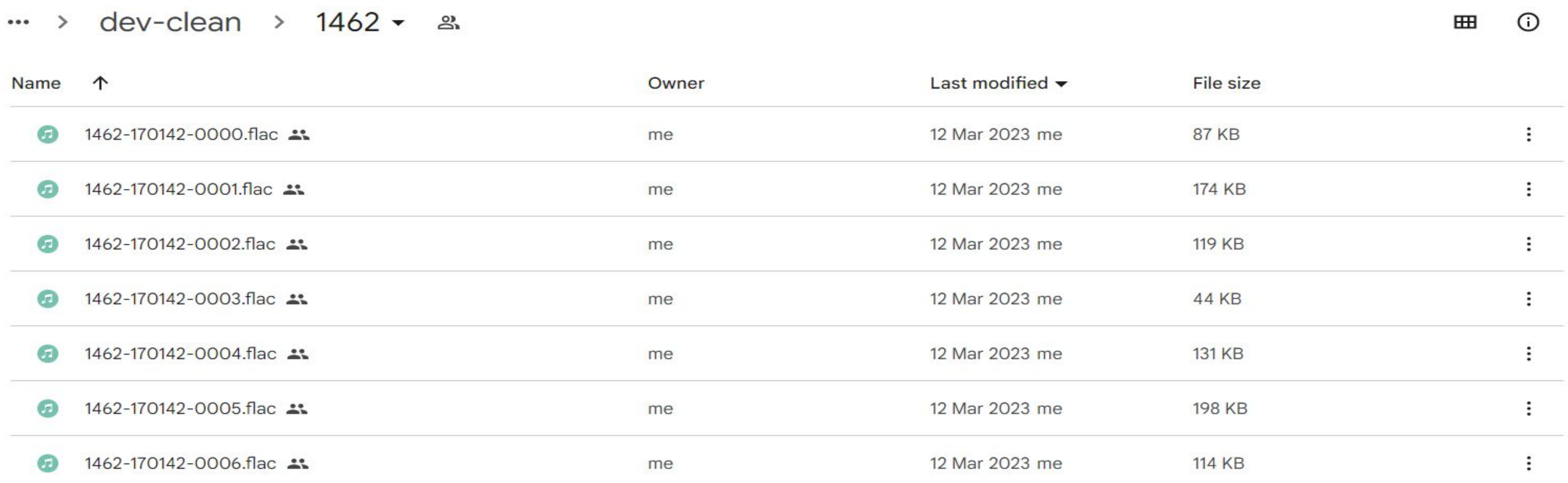
Introduction

- Our goal is to develop a model that can effectively and accurately identify a registered speaker.
- There are two primary methods which we will employ in our model development process and then we will use a voting classifier to combine the predictions of these two methods.
- The first method involves using the python library librosa to extract numerical features from the audio clips, which will be utilized to train a neural network (NN) model.
- The second approach involves converting the audio clips into images, which will then be used to train a convolutional neural network (CNN) model.

Neural Network Model (NN) Method 1

DATASET:-

- https://drive.google.com/drive/folders/1yqE_-f1p93o0sVntdXpE3iw0BPHLkty6?usp=share_link
- **Above is the link of the dataset folder we used. It consists of voice folders of 33 speakers i.e. 33 classes. Each speaker folder has audio samples of the speakers.**



... > dev-clean > 1462

Name	Owner	Last modified	File size
1462-170142-0000.flac	me	12 Mar 2023 me	87 KB
1462-170142-0001.flac	me	12 Mar 2023 me	174 KB
1462-170142-0002.flac	me	12 Mar 2023 me	119 KB
1462-170142-0003.flac	me	12 Mar 2023 me	44 KB
1462-170142-0004.flac	me	12 Mar 2023 me	131 KB
1462-170142-0005.flac	me	12 Mar 2023 me	198 KB
1462-170142-0006.flac	me	12 Mar 2023 me	114 KB

Procedure:

Firstly, we have included and installed the required libraries which are:-

1. **Tensorflow**
2. **Keras**
3. **Pandas**
4. **Numpy**
5. **Librosa**
6. **Matplotlib**
7. **Scikit-Learn**

Procedure:

- Then we mounted the Google Drive to Collab for including our dataset .

```
from google.colab import drive
drive.mount('/content/drive')
```

- Further We installed resampy as a requirement for Efficient sample rate conversion in Python.

```
!pip install llvmlite==0.31.0
!pip install resampy
```

Various Voice Features

- **Mel-frequency cepstral coefficients (MFCCs)**

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

A/D Conversion: Analog to digital conversion.

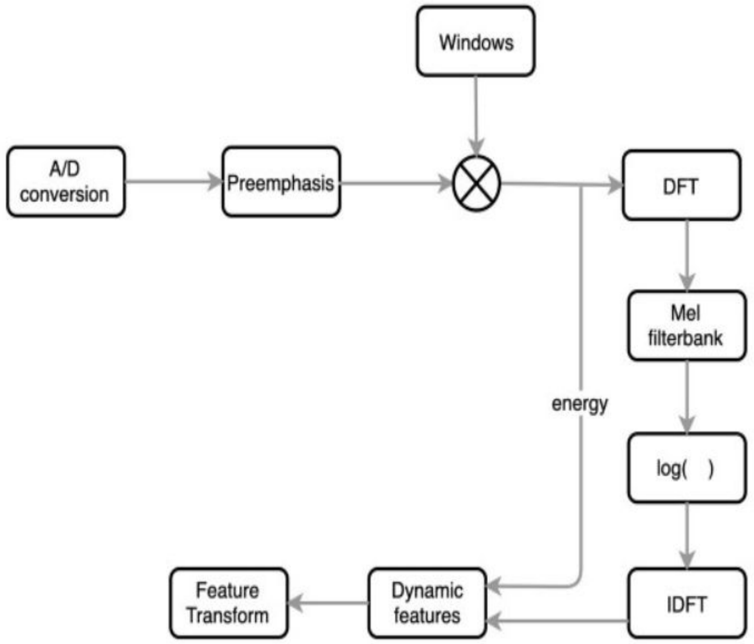
Preemphasis: Preemphasis increases the magnitude of energy in the higher frequency

Windowing:breaking the audio signal into different segments to detect phones in the voice.

DFT(Discrete Fourier Transform): Converting the signal from the time domain to the frequency domain

Mel-Filter Bank: We use the mel scale to map the actual frequency to the frequency that human beings will perceive.

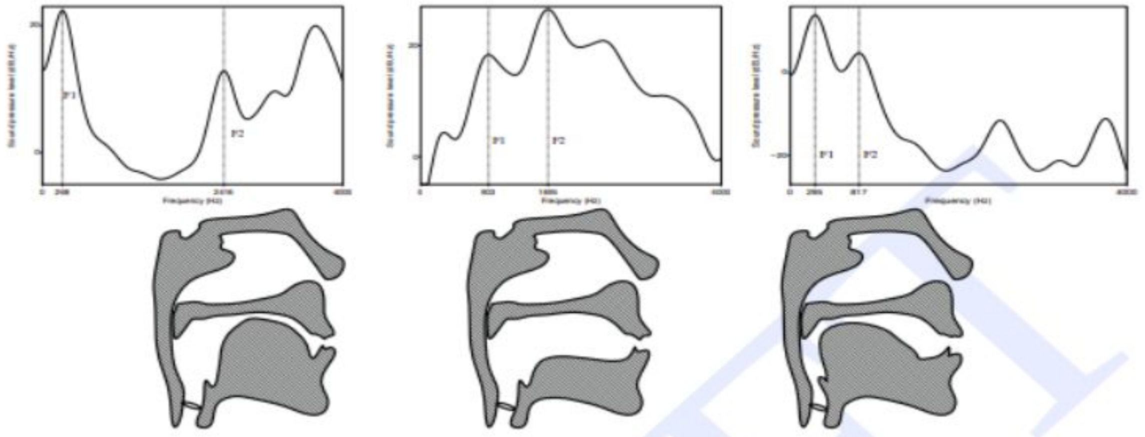
Applying Log and IDFT: doing the inverse transform of the output from the previous step



$$mel(f) = 1127\ln(1 + \frac{f}{700})$$

MFCC

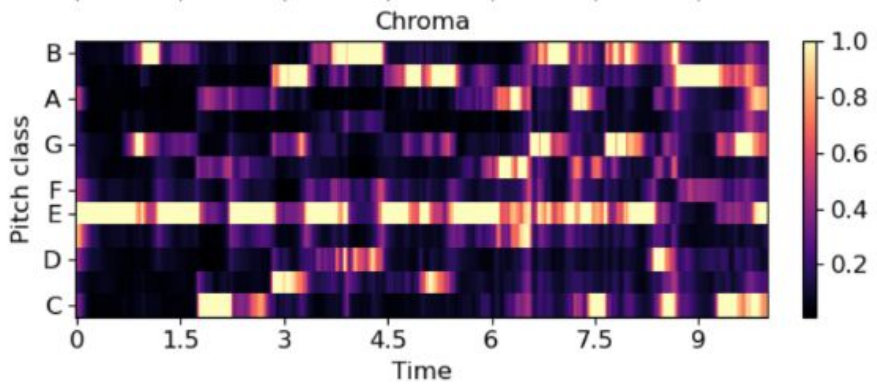
The periods in the time domain and frequency domain are inverted after IDFT transformation.Hence, a inverse relation between time and frequency domains w.r.t frequency. The MFCC model takes the first 12 coefficients of the signal after applying the idft operations.It also takes the energy of the signal sample as the feature.Along with these 13 features, the MFCC technique will consider the first order derivative and second order derivatives of the features which constitute another 26 features.So, in total it generates 39 features from each audio signal.



[iy] (tea) [ae] (cat) [uw] (moo)

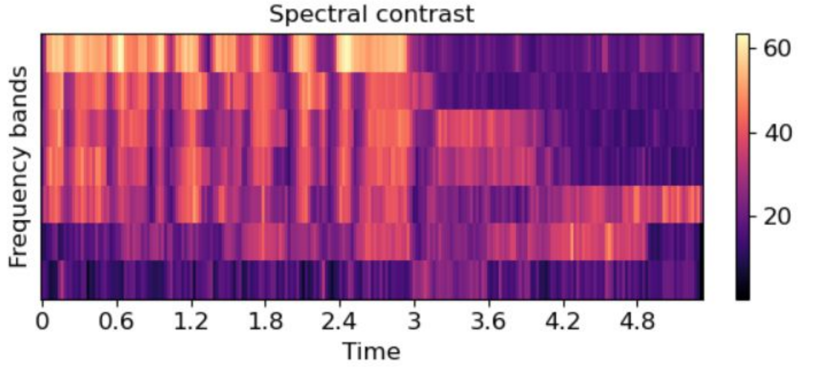
CHROMAGRAM:-

Mostly used in the music industry, the term ***chroma feature*** or ***chromagram*** closely relates to the twelve different pitch classes Chroma-based features are a powerful tool for analyzing music whose pitches can be meaningfully categorized (often into twelv categories).
One main property of chroma features is that they capture harmonic and melodic characteristics of music, while being robu to changes in timbre and instrumentation.



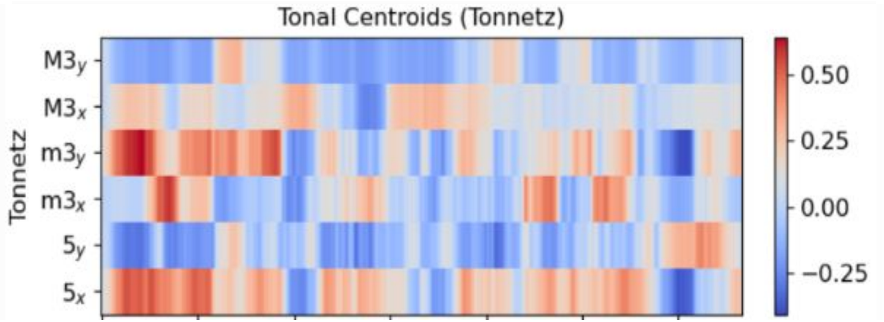
SPECTRAL CONTRAST:-

Spectral contrast is defined as the decibel difference between peaks and valleys in the spectrum.
Each frame of a spectrogram S is divided into sub-bands. For each sub-band, the energy contrast is estimated by comparing the mean energy in the top quantile (peak energy) to that of the bottom quantile (valley energy).



TONNETZ:-

The Tonnetz is **a pictorial representation of the notes in the plane that reveal affinities and structures between notes of input audio signal. In our function,** It projects chroma features onto a 6-dimensional basis representing the perfect fifth, minor third, and major third(These are the intervals between semitones of a particulai note of the major or minor scale) each as two-dimensional coordinates.

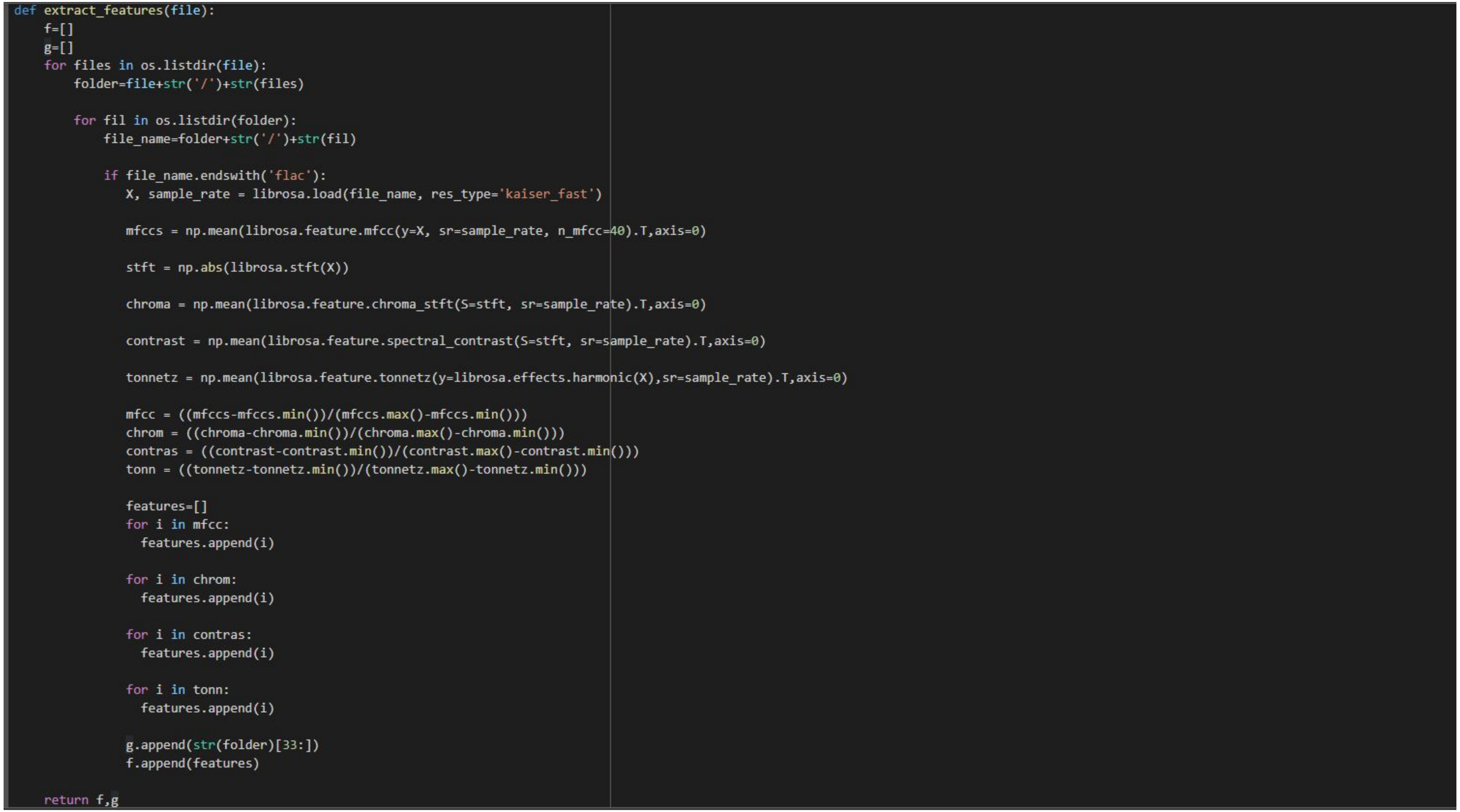


Procedure:

- Then we defined a function named “extract_features” which is responsible for extracting different voice characteristic features like **Mel-frequency cepstral coefficients (MFCCs)**, **chromagram**, **spectral contrast** and **tonal centroid features(tonnetz)**.

- The function **extract_features** takes a directory path as input and iterates over all the files in subdirectories of the given directory. For each file, it extracts audio features such as Mel-frequency cepstral coefficients (MFCCs), chromagram, spectral contrast, and tonal centroid features using the Librosa library.

- The extracted features are then normalized and stored in a list **‘f’**. Additionally, the name of the subdirectory containing the file is stored in another list ‘**g’**. Finally, the function returns two lists f and g containing the extracted features and the corresponding subdirectory names respectively.

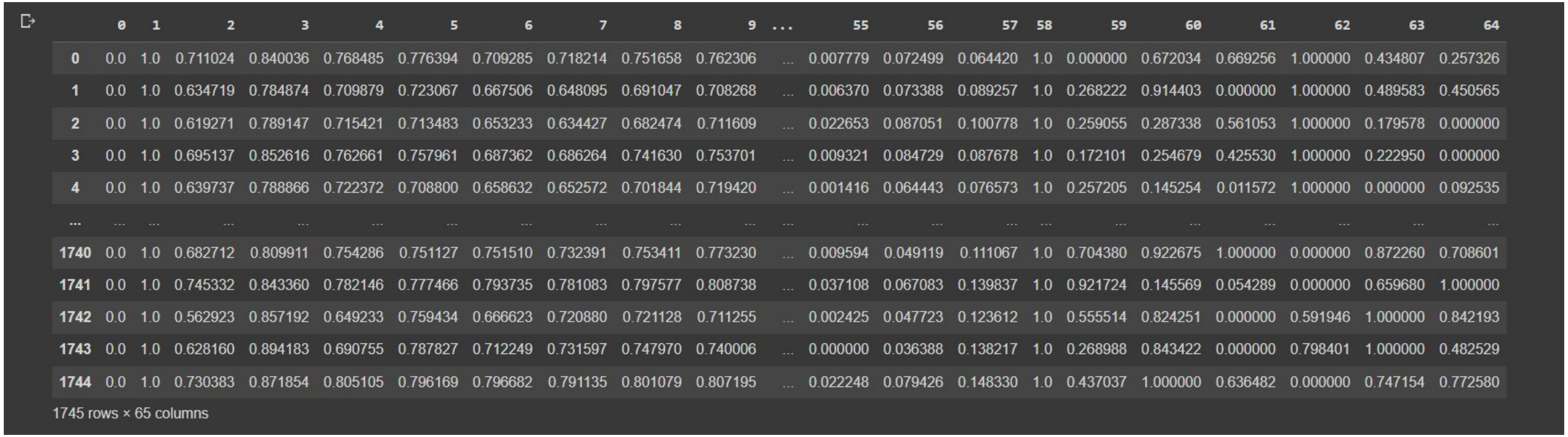


```python
def extract_features(file):
    f=[]
    g=[]
    for files in os.listdir(file):
        folder=file+str('/')+str(files)

        for fil in os.listdir(folder):
            file_name=folder+str('/')+str(fil)

            if file_name.endswith('flac'):
               X, sample_rate = librosa.load(file_name, res_type='kaiser_fast')

               mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)

               stft = np.abs(librosa.stft(X))

               chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)

               contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)

               tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),sr=sample_rate).T,axis=0)

               mfcc = ((mfccs-mfccs.min())/(mfccs.max()-mfccs.min()))
               chrom = ((chroma-chroma.min())/(chroma.max()-chroma.min()))
               contras = ((contrast-contrast.min())/(contrast.max()-contrast.min()))
               tonn = ((tonnetz-tonnetz.min())/(tonnetz.max()-tonnetz.min()))

               features=[]
               for i in mfcc:
                 features.append(i)

               for i in chrom:
                 features.append(i)

               for i in contras:
                 features.append(i)

               for i in tonn:
                 features.append(i)

               g.append(str(folder)[33:])
               f.append(features)

    return f,g
```

Procedure:

Now, we apply the above function over our dataset folder path through which it extracts and returns the features and then inserts it into a dataframe.

A single row of the data frame represents all the features of a single audio file and this will be the input.



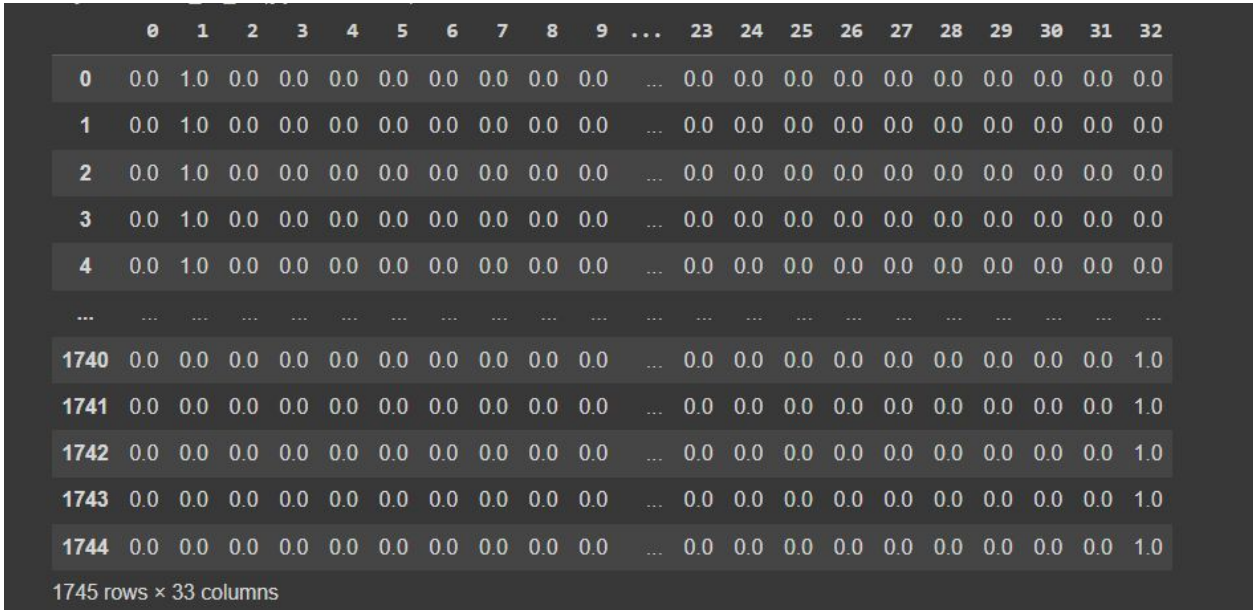
	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	64
0	0.0	1.0	0.711024	0.840036	0.768485	0.776394	0.709285	0.718214	0.751658	0.762306	...	0.007779	0.072499	0.064420	1.0	0.000000	0.672034	0.669256	1.000000	0.434807	0.257326
1	0.0	1.0	0.634719	0.784874	0.709879	0.723067	0.667506	0.648095	0.691047	0.708268	...	0.006370	0.073388	0.089257	1.0	0.268222	0.914403	0.000000	1.000000	0.489583	0.450565
2	0.0	1.0	0.619271	0.789147	0.715421	0.713483	0.653233	0.634427	0.682474	0.711609	...	0.022653	0.087051	0.100778	1.0	0.259055	0.287338	0.561053	1.000000	0.179578	0.000000
3	0.0	1.0	0.695137	0.852616	0.762661	0.757961	0.687362	0.686264	0.741630	0.753701	...	0.009321	0.084729	0.087678	1.0	0.172101	0.254679	0.425530	1.000000	0.222950	0.000000
4	0.0	1.0	0.639737	0.788866	0.722372	0.708800	0.658632	0.652572	0.701844	0.719420	...	0.001416	0.064443	0.076573	1.0	0.257205	0.145254	0.011572	1.000000	0.000000	0.092535
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1740	0.0	1.0	0.682712	0.809911	0.754286	0.751127	0.751510	0.732391	0.753411	0.773230	...	0.009594	0.049119	0.111067	1.0	0.704380	0.922675	1.000000	0.000000	0.872260	0.708601
1741	0.0	1.0	0.745332	0.843360	0.782146	0.777466	0.793735	0.781083	0.797577	0.808738	...	0.037108	0.067083	0.139837	1.0	0.921724	0.145569	0.054289	0.000000	0.659680	1.000000
1742	0.0	1.0	0.562923	0.857192	0.649233	0.759434	0.666623	0.720880	0.721128	0.711255	...	0.002425	0.047723	0.123612	1.0	0.555514	0.824251	0.000000	0.591946	1.000000	0.842193
1743	0.0	1.0	0.628160	0.894183	0.690755	0.787827	0.712249	0.731597	0.747970	0.740006	...	0.000000	0.036388	0.138217	1.0	0.268988	0.843422	0.000000	0.798401	1.000000	0.482529
1744	0.0	1.0	0.730383	0.871854	0.805105	0.796169	0.796682	0.791135	0.801079	0.807195	...	0.022248	0.079426	0.148330	1.0	0.437037	1.000000	0.636482	0.000000	0.747154	0.772580

1745 rows × 65 columns

Hot Label Encoding

We used hot label encoding, which involves creating a binary vector for each unique value in the categorical data.

Hot label encoding is commonly used in machine learning models as it allows categorical data to be represented as numerical data, which most machine learning algorithms and models can handle.



```
        0    1    2    3    4    5    6    7    8    9  ...   23   24   25   26   27   28   29   30   31   32
0     0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1     0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2     0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3     0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4     0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
1740  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1741  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1742  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1743  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
1744  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0

1745 rows × 33 columns
```

Procedure:

Moving further, we used the `train_test_split` function from scikit-learn to split a dataset into training, validation, and test sets. This is a common technique used in machine learning to evaluate and compare different models.

1. `vzz` is a feature matrix (also known as X) containing the input data for the machine learning model.
2. `yy` is a target array (also known as y) containing the output labels for the machine learning model.
3. The `train_test_split` function splits the `vzz` and `yy` arrays into two sets: one for training and one for testing. The `test_size` parameter specifies the percentage of data to use for the test set (in this case, 10% of the data).
4. The `random_state` parameter ensures that the random splitting of the data is reproducible.
5. The resulting splits are assigned to the variables `X_trainn`, `X_test`, `y_trainn`, and `y_test`.

After this initial split, the code performs another `train_test_split` on the `X_trainn` and `y_trainn` arrays, but this time it uses a smaller test size of 10% to create a validation set. The resulting splits are assigned to `X_train`, `X_val`, `y_train`, and `y_val`.

Overall, this code is splitting the original dataset into three sets: a training set, a validation set, and a test set. The training set is used to train the machine learning model, the validation set is used to tune the model hyperparameters, and the test set is used to evaluate the model's performance on new, unseen data.

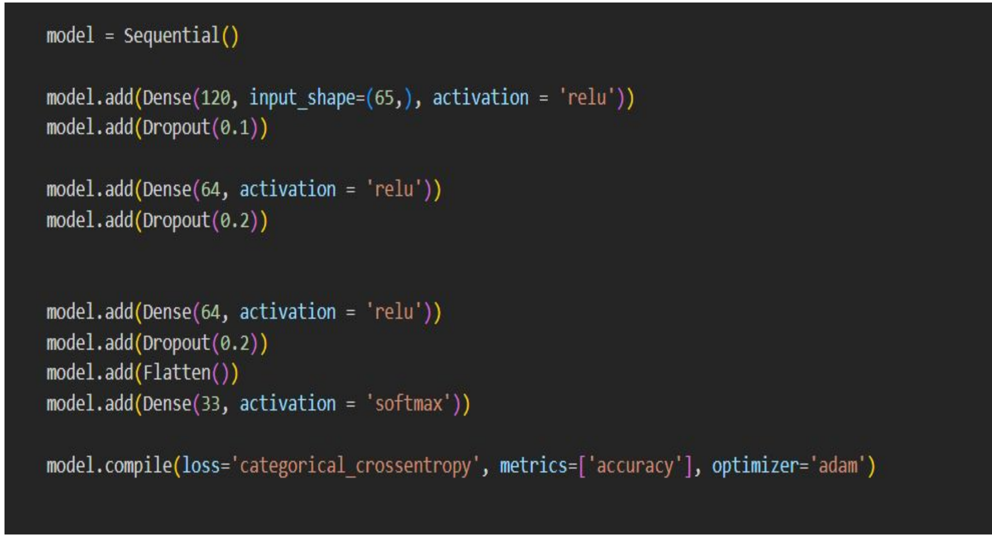
Procedure

We then trained a Keras based neural network as shown which is a breakdown of layers and parameters.

`Sequential()` creates a linear stack of layers for the neural network model. ‘`Dense(120, input shape=(65,), activation = 'relu')` creates a fully connected layer `Dropout(0.1)` randomly drops 10% of the input units during training, which helps to prevent overfitting.`Flatten()` flattens the output from the previous layer into a 1D array, which can then be fed into the output layer.

`Dense(33, activation = 'softmax')` creates the output layer with 33 units and the softmax activation function

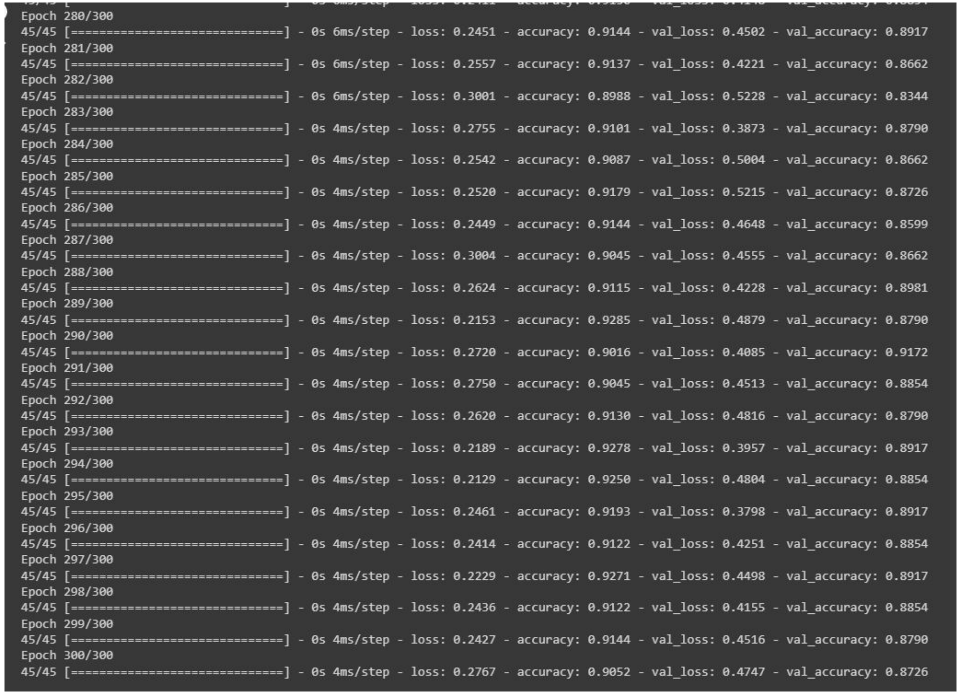
`model.compile(loss='categorical crossentropy', metrics=['accuracy'], optimizer='adam')` compiles the model with the categorical cross-entropy loss function, the accuracy metric, and the Adam optimizer



```
model = Sequential()

model.add(Dense(120, input_shape=(65,), activation = 'relu'))
model.add(Dropout(0.1))

model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))


model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(33, activation = 'softmax'))

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

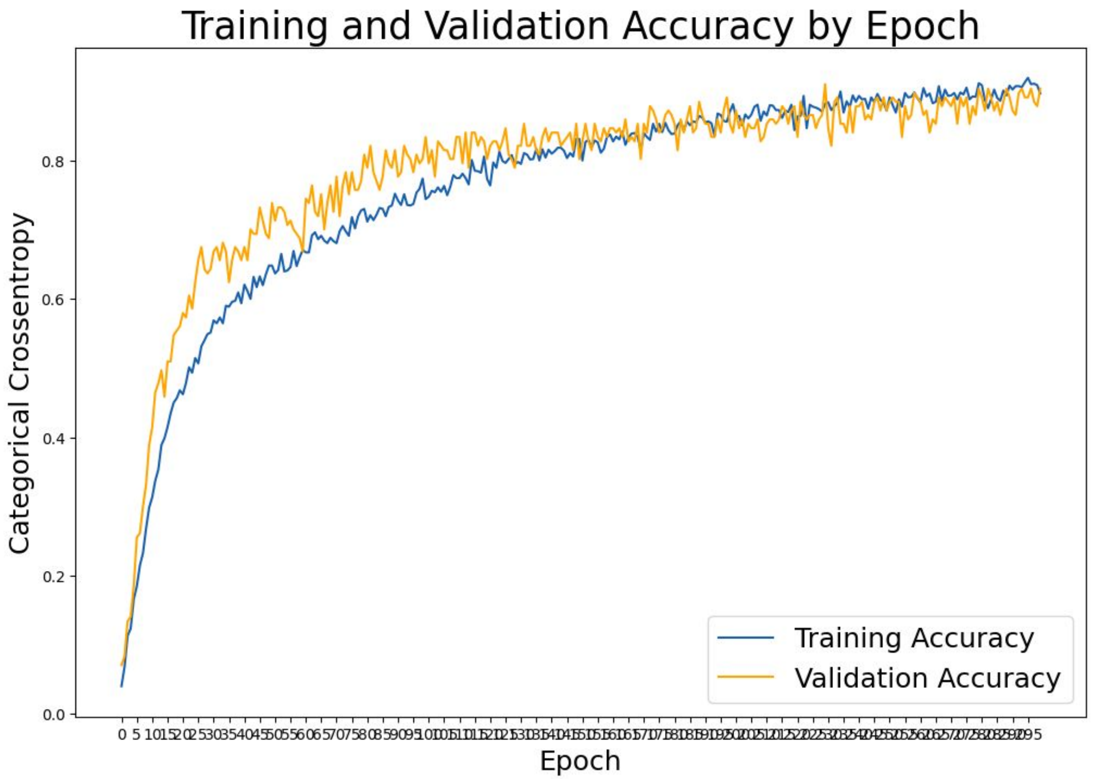
Training NN model

We then trained a neural network model with 300 epochs and got **90.52%** training accuracy and **87.26%** validation accuracy.



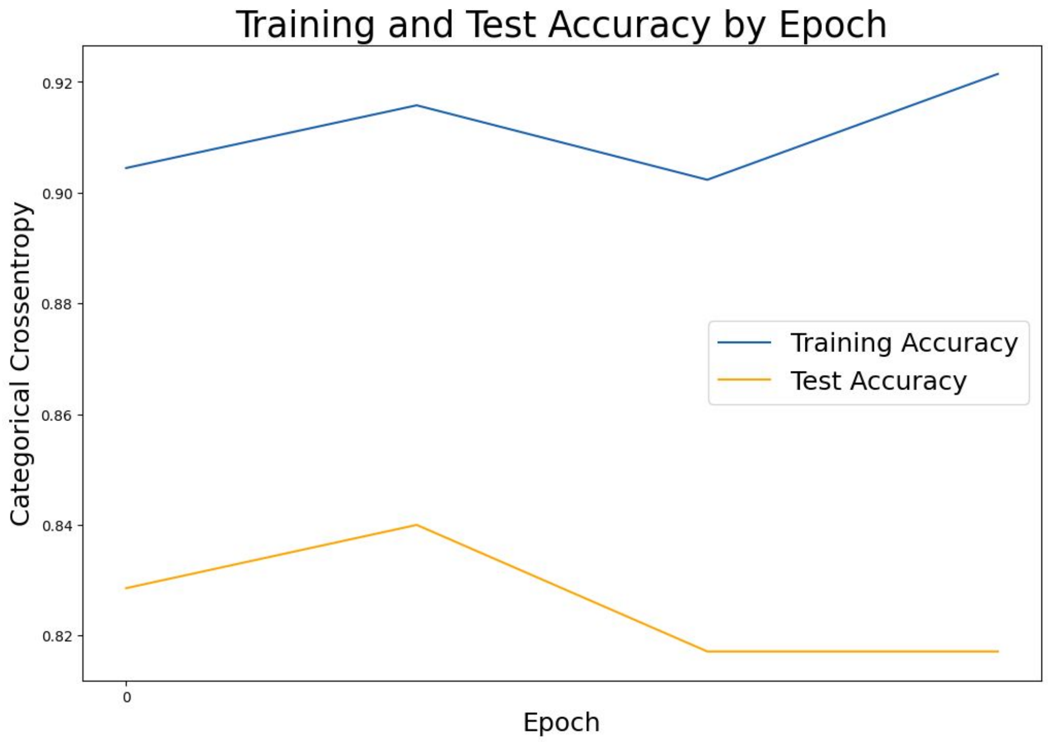
```
Epoch 280/300
45/45 [==============================] - 0s 6ms/step - loss: 0.2451 - accuracy: 0.9144 - val_loss: 0.4502 - val_accuracy: 0.8917
Epoch 281/300
45/45 [==============================] - 0s 6ms/step - loss: 0.2557 - accuracy: 0.9137 - val_loss: 0.4221 - val_accuracy: 0.8662
Epoch 282/300
45/45 [==============================] - 0s 6ms/step - loss: 0.3001 - accuracy: 0.8988 - val_loss: 0.5228 - val_accuracy: 0.8344
Epoch 283/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2755 - accuracy: 0.9101 - val_loss: 0.3873 - val_accuracy: 0.8790
Epoch 284/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2542 - accuracy: 0.9087 - val_loss: 0.5004 - val_accuracy: 0.8662
Epoch 285/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2520 - accuracy: 0.9179 - val_loss: 0.5215 - val_accuracy: 0.8726
Epoch 286/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2449 - accuracy: 0.9144 - val_loss: 0.4648 - val_accuracy: 0.8599
Epoch 287/300
45/45 [==============================] - 0s 4ms/step - loss: 0.3004 - accuracy: 0.9045 - val_loss: 0.4555 - val_accuracy: 0.8662
Epoch 288/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2624 - accuracy: 0.9115 - val_loss: 0.4228 - val_accuracy: 0.8981
Epoch 289/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2153 - accuracy: 0.9285 - val_loss: 0.4879 - val_accuracy: 0.8790
Epoch 290/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2720 - accuracy: 0.9016 - val_loss: 0.4085 - val_accuracy: 0.9172
Epoch 291/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2750 - accuracy: 0.9045 - val_loss: 0.4513 - val_accuracy: 0.8854
Epoch 292/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2620 - accuracy: 0.9130 - val_loss: 0.4816 - val_accuracy: 0.8790
Epoch 293/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2189 - accuracy: 0.9278 - val_loss: 0.3957 - val_accuracy: 0.8917
Epoch 294/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2129 - accuracy: 0.9250 - val_loss: 0.4804 - val_accuracy: 0.8854
Epoch 295/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2461 - accuracy: 0.9193 - val_loss: 0.3798 - val_accuracy: 0.8917
Epoch 296/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2414 - accuracy: 0.9122 - val_loss: 0.4251 - val_accuracy: 0.8854
Epoch 297/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2229 - accuracy: 0.9271 - val_loss: 0.4498 - val_accuracy: 0.8917
Epoch 298/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2436 - accuracy: 0.9122 - val_loss: 0.4155 - val_accuracy: 0.8854
Epoch 299/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2427 - accuracy: 0.9144 - val_loss: 0.4516 - val_accuracy: 0.8790
Epoch 300/300
45/45 [==============================] - 0s 4ms/step - loss: 0.2767 - accuracy: 0.9052 - val_loss: 0.4747 - val_accuracy: 0.8726
```

Plotting Graph



Plotting Graph

We got test accuracy as **84%.**

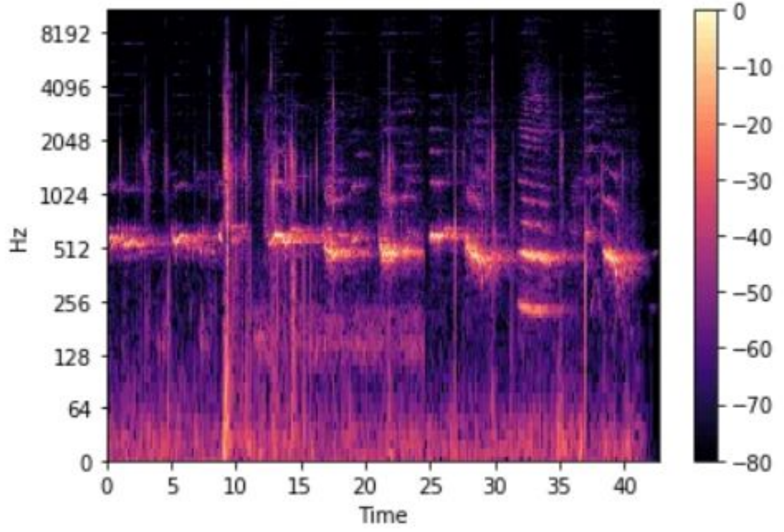


Convolutional Neural Network (CNN) Method 2

- We used the same dataset folder named “finalaudio” i.e. https://drive.google.com/drive/folders/1fTlgHdwP5lwMEmSED3qoHtEvuD-FWgP6?usp=share_link
- It similarly consists of voice samples of 33 classes or speakers with a total of 1744 files.

- We installed the following libraries:-

1. **Tensorflow**
2. **Keras**
3. **Pandas**
4. **Numpy**
5. **Librosa**
6. **Matplotlib**
7. **Scikit- Learn**
8. **CSV**
9. **Os**
10. **IPython.Display**
11. **glob**



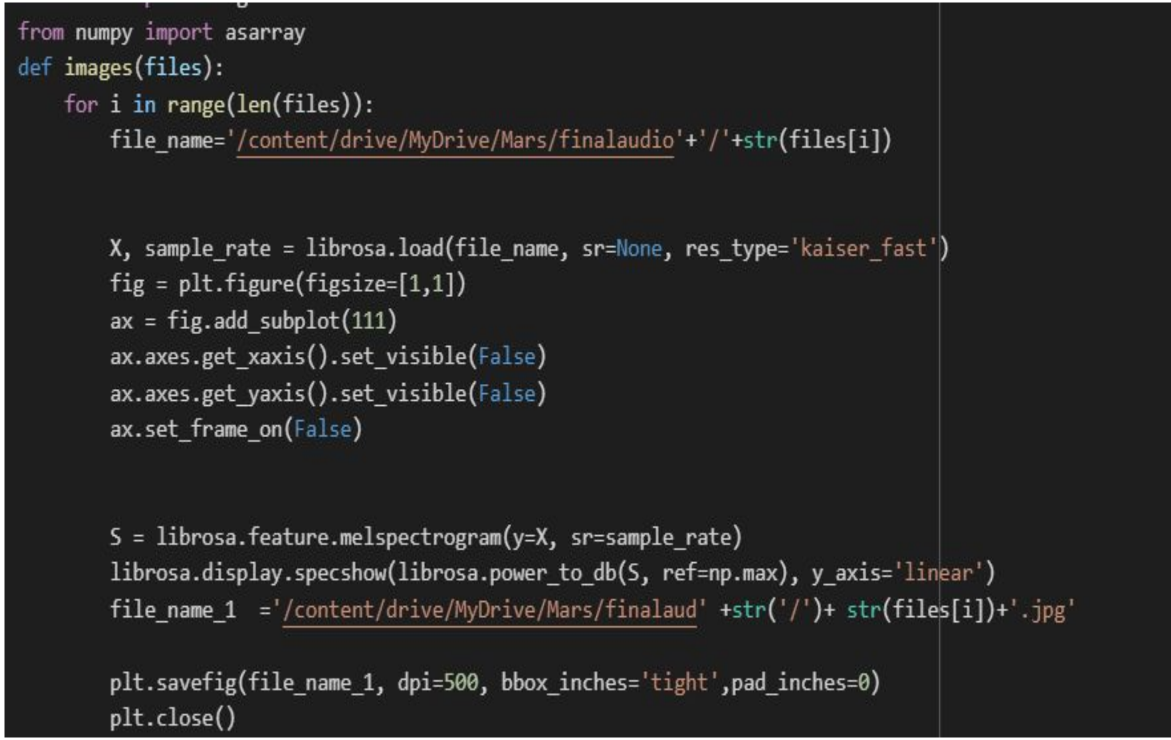
MEL-SPECTROGRAM EXAMPLE

Converting audio files to mel spectrogram

Then we mounted the Google Drive to Collab for including our dataset. Further we created a function which works as follows:-

For each file in the input list, the function loads the audio data using the librosa library.The function creates a figure with a size of 1 by 1 and adds a subplot to the figure. It then sets the x-axis and y-axis visibility to false, and removes the frame of the subplot.

The function calculates the mel spectrogram of the audio data using librosa.feature.melspectrogram. The power spectrogram is converted to a logarithmic scale. The function saves the resulting image as a JPEG file with a dpi of 500, and a filename that is created using the input file name by appending ".jpg" to it.

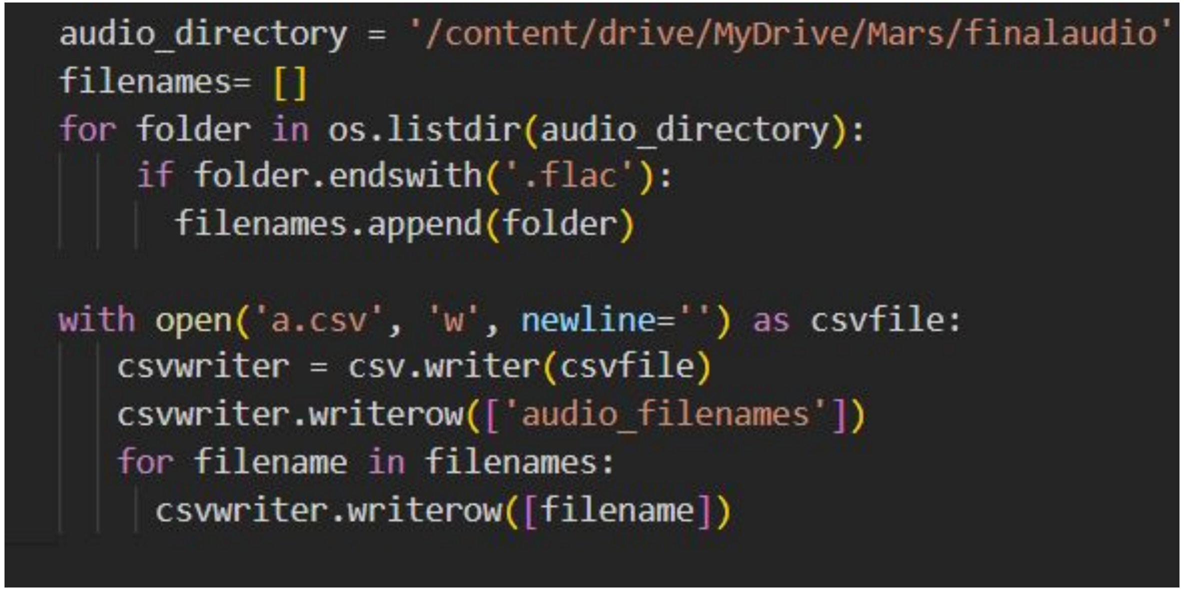


```python
from numpy import asarray
def images(files):
    for i in range(len(files)):
        file_name='/content/drive/MyDrive/Mars/finalaudio'+'/'+str(files[i])


        X, sample_rate = librosa.load(file_name, sr=None, res_type='kaiser_fast')
        fig = plt.figure(figsize=[1,1])
        ax = fig.add_subplot(111)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        ax.set_frame_on(False)


        S = librosa.feature.melspectrogram(y=X, sr=sample_rate)
        librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='linear')
        file_name_1  ='/content/drive/MyDrive/Mars/finalaud' +str('/')+ str(files[i])+'.jpg'

        plt.savefig(file_name_1, dpi=500, bbox_inches='tight',pad_inches=0)
        plt.close()
```

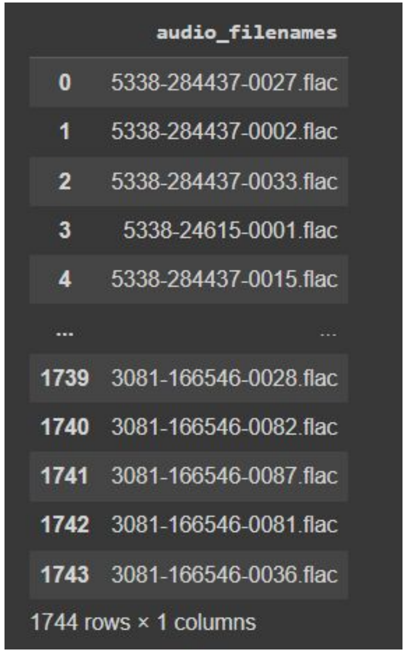
Creating a CSV file:

We created a CSV by appending the filenames in it and creating a column named as ‘audio_filenames’.

Then converted it into a dataframe with 1744 rows .



```
audio_directory = '/content/drive/MyDrive/Mars/finalaudio'
filenames= []
for folder in os.listdir(audio_directory):
    if folder.endswith('.flac'):
      filenames.append(folder)

with open('a.csv', 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(['audio_filenames'])
    for filename in filenames:
      csvwriter.writerow([filename])
```

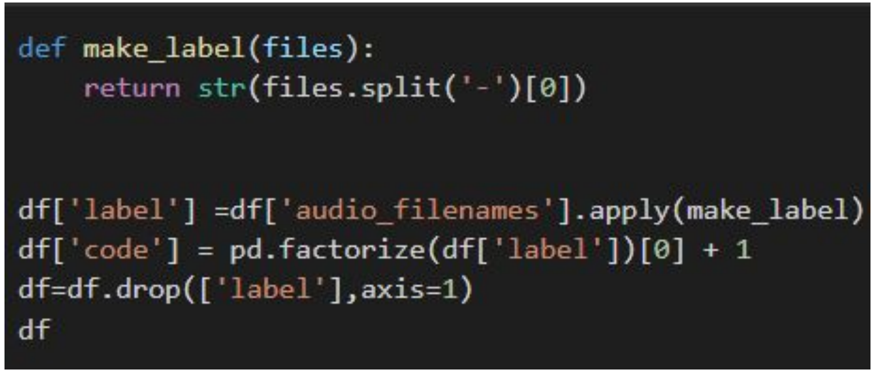


	audio_filenames
0	5338-284437-0027.flac
1	5338-284437-0002.flac
2	5338-284437-0033.flac
3	5338-24615-0001.flac
4	5338-284437-0015.flac
...	...
1739	3081-166546-0028.flac
1740	3081-166546-0082.flac
1741	3081-166546-0087.flac
1742	3081-166546-0081.flac
1743	3081-166546-0036.flac

1744 rows × 1 columns

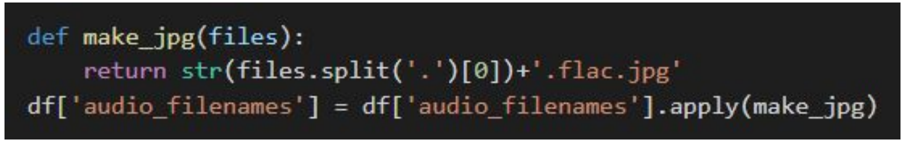
Procedure:

We defined a function named ‘make_label’ which is responsible for splitting the file names and thus obtaining the [0] index term which is the label of the speaker and then applying to the dataframe with column named as ‘code’.



```
def make_label(files):
    return str(files.split('-')[0])


df['label'] =df['audio_filenames'].apply(make_label)
df['code'] = pd.factorize(df['label'])[0] + 1
df=df.drop(['label'],axis=1)
df
```

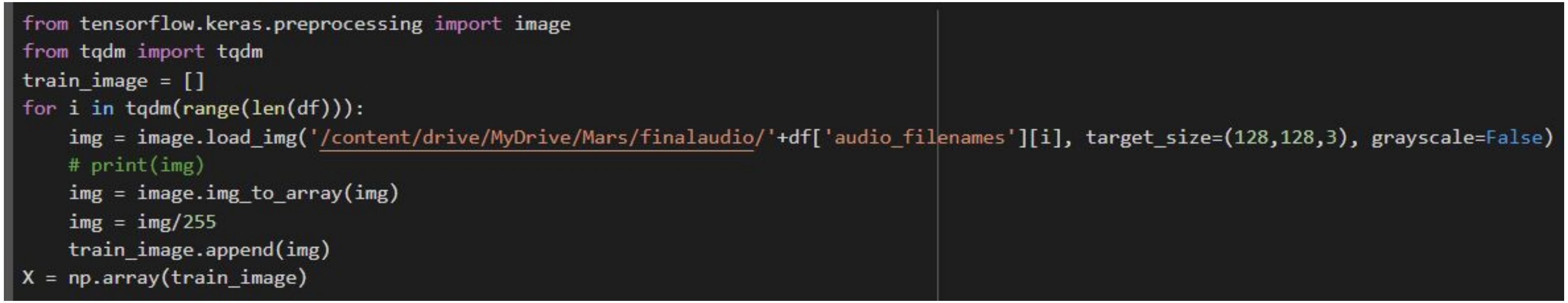
We defined a function named ‘make_jpg’ through which we could change the names of the files from ‘.flac’ to ‘.flac.jpg’.



```
def make_jpg(files):
    return str(files.split('.')[0])+'.flac.jpg'
df['audio_filenames'] = df['audio_filenames'].apply(make_jpg)
```

Converting images to array

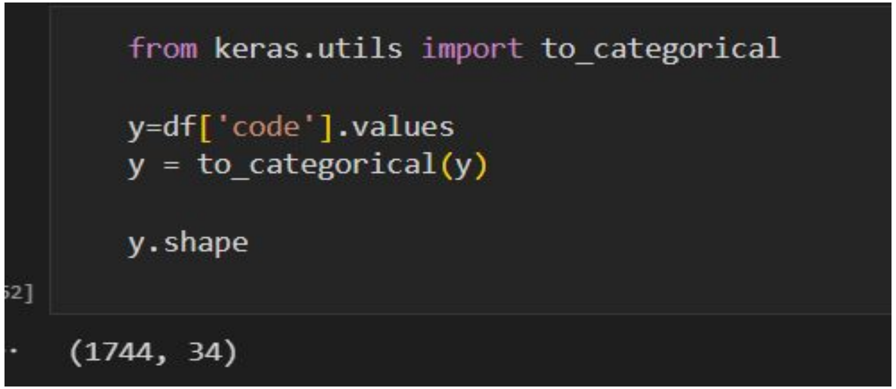
The code loads a set of images from a directory, resizes them to a target size of (128,128), and converts them to NumPy arrays using Keras' image preprocessing module. The resulting arrays are normalized by dividing the pixel values by 255 and appended to a list called `train_image`. Finally, the list of NumPy arrays is converted to a single NumPy array `X` that can be used as input data for a machine learning model. The purpose of this code is to prepare image data for use in a deep learning model by converting it to a numerical format that can be processed by the model.



```
from tensorflow.keras.preprocessing import image
from tqdm import tqdm
train_image = []
for i in tqdm(range(len(df))):
    img = image.load_img('/content/drive/MyDrive/Mars/finalaudio/'+df['audio_filenames'][i], target_size=(128,128,3), grayscale=False)
    # print(img)
    img = image.img_to_array(img)
    img = img/255
    train_image.append(img)
X = np.array(train_image)
```

Model and its compilation

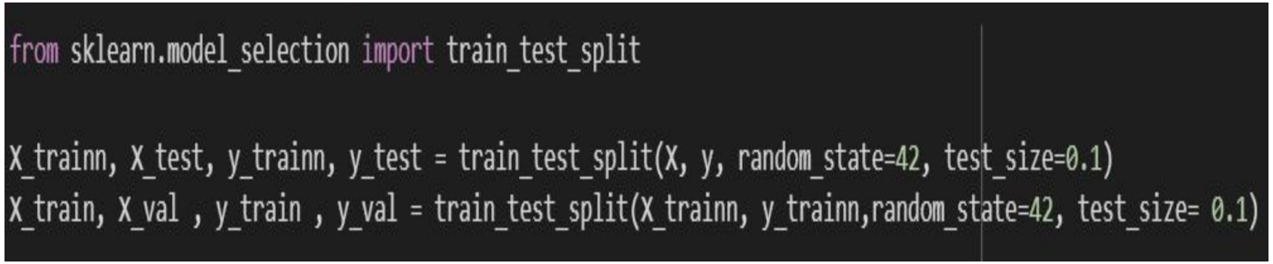
Here we converted a 1-dimensional array of integer labels `y` into a one-hot encoded format that can be used as target data.



```
from keras.utils import to_categorical

y=df['code'].values
y = to_categorical(y)

y.shape
```

```
(1744, 34)
```

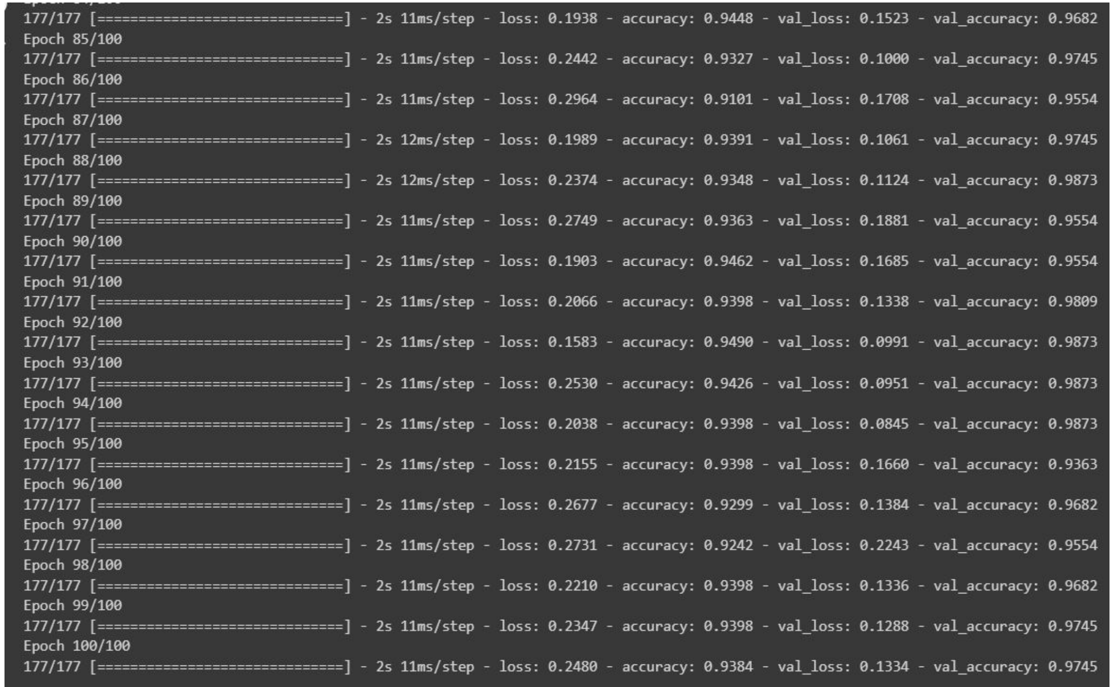
The above code uses Scikit-learn's `train_test_split()` function to split the input data `x` and target data `y` into three sets: training set, validation set, and test set.



```
from sklearn.model_selection import train_test_split

X_trainn, X_test, y_trainn, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
X_train, X_val , y_train , y_val = train_test_split(X_trainn, y_trainn,random_state=42, test_size= 0.1)
```

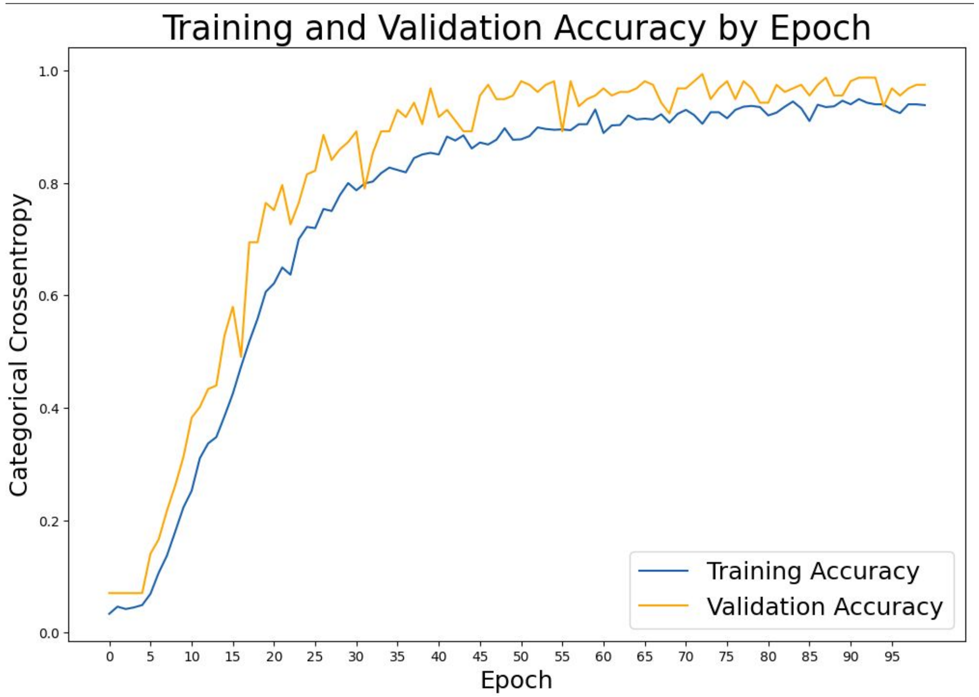
Training the CNN model

We trained using normal CNN model with 100 epochs and with a batch size of 8 , we got **93.84%** training accuracy and **97.45%** validation accuracy.



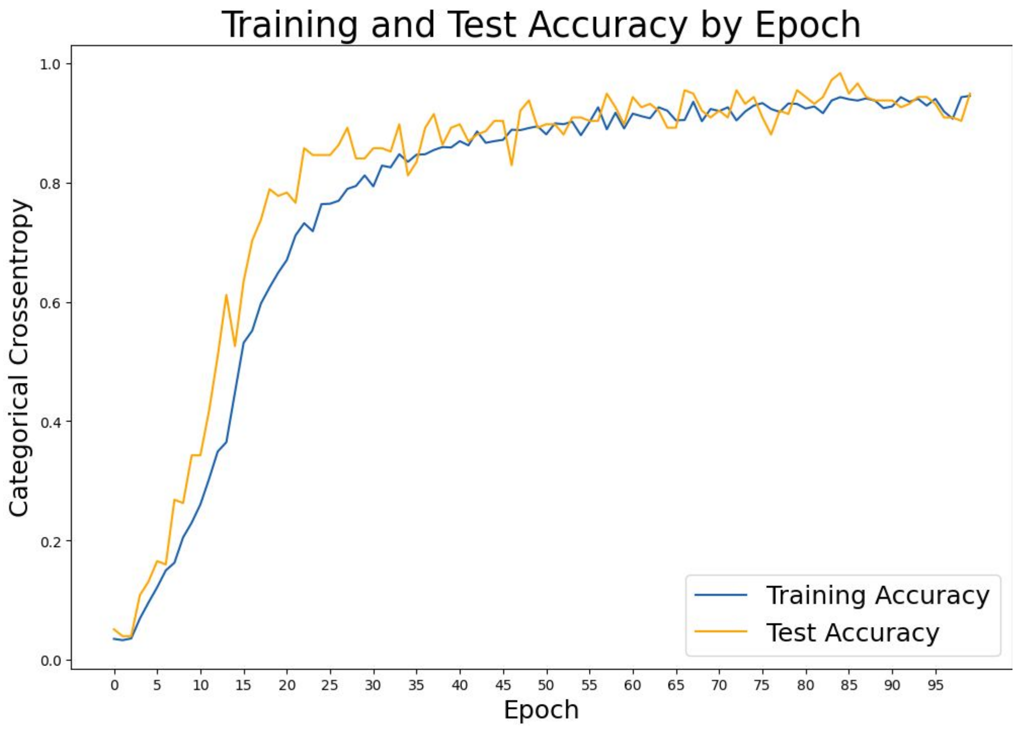
```
177/177 [==============================] - 2s 11ms/step - loss: 0.1938 - accuracy: 0.9448 - val_loss: 0.1523 - val_accuracy: 0.9682
Epoch 85/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2442 - accuracy: 0.9327 - val_loss: 0.1000 - val_accuracy: 0.9745
Epoch 86/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2964 - accuracy: 0.9101 - val_loss: 0.1708 - val_accuracy: 0.9554
Epoch 87/100
177/177 [==============================] - 2s 12ms/step - loss: 0.1989 - accuracy: 0.9391 - val_loss: 0.1061 - val_accuracy: 0.9745
Epoch 88/100
177/177 [==============================] - 2s 12ms/step - loss: 0.2374 - accuracy: 0.9348 - val_loss: 0.1124 - val_accuracy: 0.9873
Epoch 89/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2749 - accuracy: 0.9363 - val_loss: 0.1881 - val_accuracy: 0.9554
Epoch 90/100
177/177 [==============================] - 2s 11ms/step - loss: 0.1903 - accuracy: 0.9462 - val_loss: 0.1685 - val_accuracy: 0.9554
Epoch 91/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2066 - accuracy: 0.9398 - val_loss: 0.1338 - val_accuracy: 0.9809
Epoch 92/100
177/177 [==============================] - 2s 11ms/step - loss: 0.1583 - accuracy: 0.9490 - val_loss: 0.0991 - val_accuracy: 0.9873
Epoch 93/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2530 - accuracy: 0.9426 - val_loss: 0.0951 - val_accuracy: 0.9873
Epoch 94/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2038 - accuracy: 0.9398 - val_loss: 0.0845 - val_accuracy: 0.9873
Epoch 95/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2155 - accuracy: 0.9398 - val_loss: 0.1660 - val_accuracy: 0.9363
Epoch 96/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2677 - accuracy: 0.9299 - val_loss: 0.1384 - val_accuracy: 0.9682
Epoch 97/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2731 - accuracy: 0.9242 - val_loss: 0.2243 - val_accuracy: 0.9554
Epoch 98/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2210 - accuracy: 0.9398 - val_loss: 0.1336 - val_accuracy: 0.9682
Epoch 99/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2347 - accuracy: 0.9398 - val_loss: 0.1288 - val_accuracy: 0.9745
Epoch 100/100
177/177 [==============================] - 2s 11ms/step - loss: 0.2480 - accuracy: 0.9384 - val_loss: 0.1334 - val_accuracy: 0.9745
```

Plotting Graph



Plotting Graph

When got test accuracy as **94.86%.**

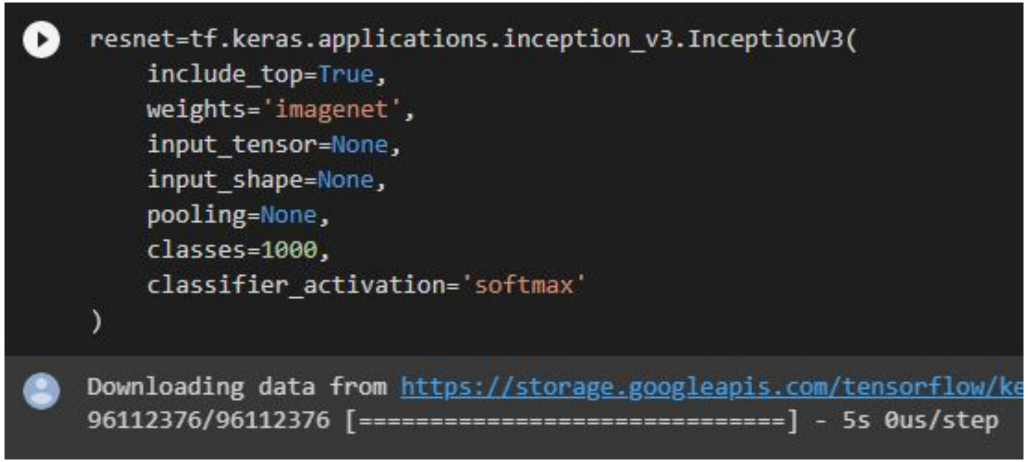


CNN vs FFNN

- **Convolutional Neural Networks (CNNs)** are often used for speech recognition tasks because they can effectively process input data with **spatial relationships**, such as **spectrograms or mel-spectrograms.**
- In contrast, Feedforward Neural Networks (FFNN) are limited in their ability to process input data with spatial relationships, as they do not consider the spatial structure of the input. FFNN can only process input data in a **one-dimensional sequential manner.**
- CNNs, on the other hand, have a specialized architecture that allows them to recognize patterns in two-dimensional data, such as spectrograms. **By using convolutional layers that filter input data in a way that preserves spatial relationships, CNNs can effectively identify features in audio data that are relevant to speech recognition.**
- Additionally, CNNs use **pooling layers that reduce the dimensionality of the input**, which can help to reduce the computational requirements of the model and prevent overfitting.
- This specialized architecture allows CNNs to achieve higher accuracy than FFNNs for voice recognition.

Inception V3

- We downloaded the pre-trained model of Inception V3 .
- Changed the last layer with 34 classes according to our model.
- Then trained the model but whenever we tried to run, the RAM used to get exceeded and the entire notebook got crashed .



```
resnet=tf.keras.applications.inception_v3.InceptionV3(
    include_top=True,
    weights='imagenet',
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation='softmax'
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/ke
96112376/96112376 [==============================] - 5s 0us/step
```

Difficulties Faced

- We faced problem while iterating in folders and then into audio files.
- While extracting the features from the audio files , we got the error of resampy not downloaded.
- Saving the Mel-Spectrogram images in the same folder.
- While training both the models, we were getting very inefficient accuracy but after eventually tuning, exploring and choosing the appropriate parameters, we reached these results.

THANK YOU