# Final Report

## CSYE 7105 – High-Performance Computing and Parallel Machine Learning

**Team 17: Shaurya Agrawal**

### 1. Introduction

This report focuses on processing audio data to make it usable for creating a model for speaker recognition. Speaker recognition is a fundamental task of speech processing and has a wide application in the real world. It is important for audio-based information retrieval such as meeting recordings, telephone calls, subtitles generation, etc. The process of building a model can be carried out by using acoustic features of the audio such as linear predictive cepstral coefficients, the perceptual linear prediction coefficient, and the Mel-frequency cepstral coefficients. The author proposes to use the Mel-frequency cepstral coefficients as a suitable feature to build the learning model. Speech processing has a variety of implementations such as identifying the speaker, verifying the speaker, speech-to-text application. The paper focuses on the technique for speaker identification.

### 2. Description of the dataset

The dataset initially consisted of audio recordings of 50 speakers for 60 minutes for each speaker, broken into chunks of 1 minute. Thus, roughly making it around 3000 audio files in .wav format, 160 KHz, mono channel. Each one-minute file was then broken down into chunks of 3 seconds, 5 seconds, and 10 seconds. The reason behind this has been to get more data for the learning model.

#### 2.1 Methods to break down the audio files

The idea behind breaking down the audio files is to generate many samples for each speaker for the learning model. This would eventually increase the accuracy of the model, which is what the final goal of the process is. Pydub library was used and specifically the AudioSegment function for splitting the audio files into chunks. Pyudub allows a convenient way to manipulate audio files.

#### 2.2 Challenges

In the initial steps splitting the audio seemed to be a time-consuming effort as it took more than 24 hours to split audio recordings of all 50 speakers. Thus, when a subset of data of 5 speakers was selected. Output files of the 3-second chunks for 50 speakers were counted to be little more than 97000. The final subset has 1560 wav files.

### 3. Relevant Topics

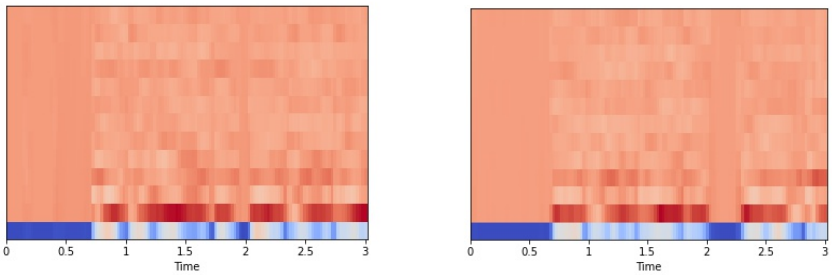
#### 3.1 MFCC – Mel-frequency cepstral coefficients

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. The cepstral is a representation of the audio clip as a non-linear “spectrum of the spectrum”. The difference between the cepstrum and the Mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more

closely than the linearly spaced frequency bands used in the normal spectrum. This frequency warping can allow for a better representation of sound.

Librosa package is essentially used for sound and audio analysis. Firstly, the audio or the wav files are to be converted to a floating-point time series, which was achieved by usin librosa.load(). This audio loading function takes the input file in wav or mp3 format and reads it as a time series of floating points. Further, the spectral feature; MFCC of the audio file is to be extracted which requires the use of librosa.feature.mfcc(). Librosa.feature.mfcc() only takes the audio file in the form of np.ndarray of shape(n). This array is already available by the use of librosa.load(). In general number of coefficients to be extracted for the audio file is 13, it can be more than that or lesser. The project includes flexibility of defining the number of coefficients as per requirements.

Eventually librosa.display.specshow() plots the coefficients extracted from the feature extraction process. MFCC feature plot figure for every speaker, after picking a chunk at random is shown below.

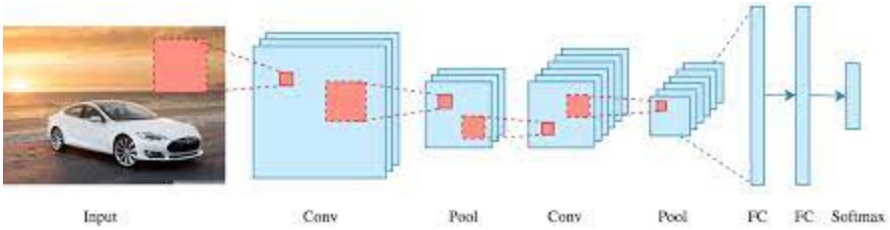



**3.2 Alternative**

Another way to proceed would have been to use the spectrograms of the wav files. Spectrogram are also the visual representation of the spectrum of frequencies of a signal with respect to time. It is also called a sonograph or a voiceprint when generated for an audio signal. A spectrogram is less informative than the MFCC plots in general.

**3.3 Convolutional Neural Network (CNN)**

A convolutional neural network (CNN/ConvNet) is a class of deep neural networks used to perform imagery analysis. Mathematically, convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. For CNN, an RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane.



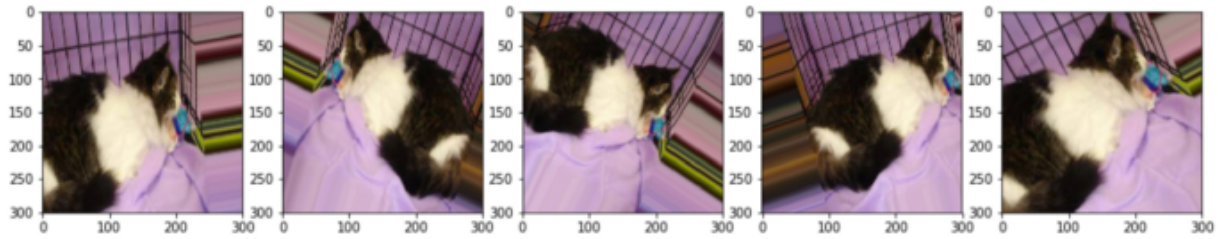

Since Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The initial layers extract basic features which are then passed on to the further layers. As we get deeper into the layers, the network becomes capable of identifying more complex features.

### 3.4 Data Augmentation to the images

It is beneficial for the model to learn more and more features and that is possible by providing more data in the image form. Thus, to create diverse scenarios from the real world about the object, the Data Augmentation technique is used. In this technique, images can be randomly zoomed by some factor, can be rotated by certain degrees, translated horizontally or vertically, shear-based transformations can be applied, flipping the image and various other ways. This leads to generating different inputs for the model to learn from and get better at learning features. Below shown is how one image can produce multiple images when augmentation transformation is performed.

```
Labels: ['cat', 'cat', 'cat', 'cat', 'cat']
```



### 3.4 ResNet50 Architecture

ResNet-50 is a convolutional neural network that is 50 layers deep. A pretrained version of the network can be loaded which is trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224*224.

### 3.4.1 Softmax activation

We use softmax activation in the neural network when a multi-class classifier is to be built or to solve the problem of assigning an instance to more than two possible classes. This is used as the output function in the last layer of the neural network To understand the softmax function, we must look at the output of the (n-1)th layer. In this layer, the values get multiplied by some weights, passed through an activation function, and aggregated into a vector that contains one value per every class of the model.

### 3.4.2 Sparse Categorical Entropy

This parameter in the model produces a category index of the *most likely* matching category. We use Sparse Categorical Entropy when the categories are mutually exclusive and when the number of categories is large(>3).

### 3.4.3 ADAM optimizer

Training network is iterative process where we want to minimize loss function, which mathematically is:

$\min_{\theta} J(\theta), \theta \in \mathbb{R}^d$, where θ is the trainable variable. ADAM(Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for careful search.

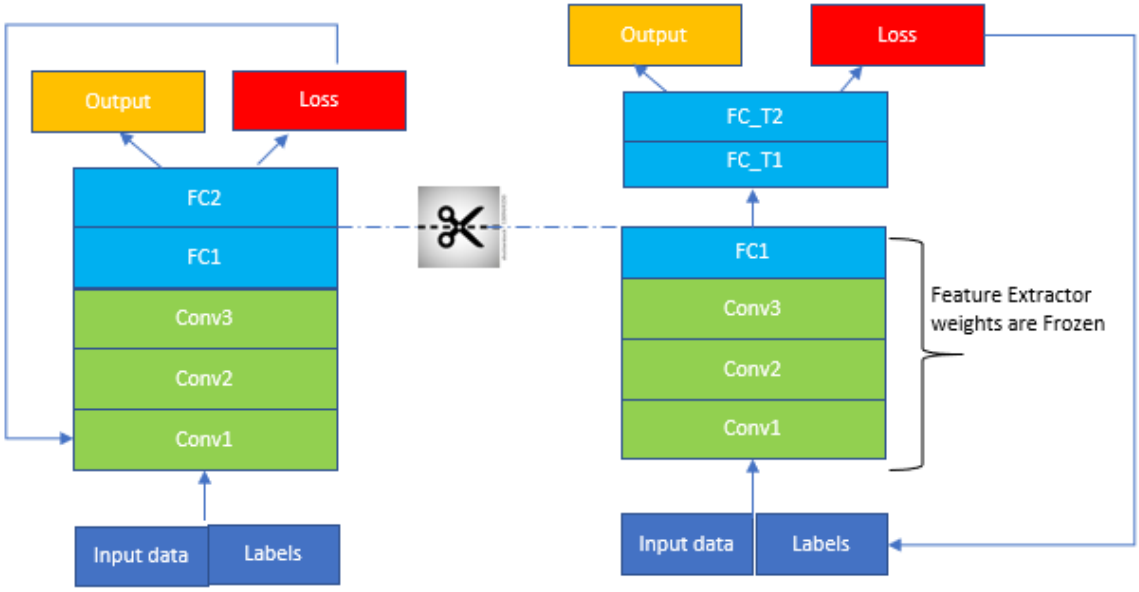
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \rightarrow (\text{mean}),$$
$$v_t = \beta_1 v_{t-1} + (1-\beta_2) g_t^2 \rightarrow (\text{uncentered variance}),$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t,$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

where $\beta 1$ = 0.9, $\beta 2$ = 0.999, $\varepsilon$ = 10−8.

### 3.5 Transfer Learning

Transfer Learning is the method of reusing a developed model for one task on a second task. Thus, to prevent efforts to create a model from scratch for a similar problem for which a model is already prepared, we use the pre-trained model as the starting point. We incorporate our layers at the end of the network to train the model to extract specific features according to our needs.




### 4. Model Procedure

1. Initially, the preprocessing of the data was performed on the complete dataset of 50 speakers, 60 minutes of audio for each speaker. Later, a subset of the data was selected which could provide the same insights. Therefore, moved forward with selecting 5 speakers randomly and 15 minutes of audio for each.

2. As it is, the model learns the feature better when there are more data points available. Thus, to accomplish this, the one-minute wav audio files have been split into 3-second chunks using *pydub package.,* which were then saved as wav file.

```
# Function to split the input audio files from directory into the desired duration of the chunk

dataset_folder_name = '5_speakers_audio_data'
def split_audio_file(initial_directory, final_directory, duration):#enter duration in milliseconds
    for folder in os.listdir(os.path.join(initial_directory,dataset_folder_name)):
        for file in os.listdir(os.path.join(initial_directory,dataset_folder_name,folder)):

            filepath = os.path.join(initial_directory, '5_speakers_audio_data', folder, file);
            filename_prefix = os.path.splitext(os.path.basename(file))[0]
            myaudio = AudioSegment.from_file(filepath, "wav")
            chunk_length_ms = duration # pydub calculates in millisec
            chunks = make_chunks(myaudio, chunk_length_ms) #Make chunks of 3 sec

            #Export all of the individual chunks as wav files

            for i, chunk in enumerate(chunks):
                chunk_name = final_directory+"/"+filename_prefix+"_chunk{0}.wav".format(i)
                print(chunk_name)
                chunk.export(chunk_name, format="wav")
```

3. For the Deep Learning Model, it was efficient to have the audio in form of images, thus using *Librosa* package the Mel-Frequency Cepstral Coefficients are extracted for all the wav files. These coefficients are then represented as a pyplot which is then saved as an image in *jpg* format for every audio chunk file.

```
# Generate the mfccs as pyplot and saving the plot as .jpg format file in the directory

audio_file_folder_name = '5_speakers_audio_data_3sec'

def generate_mfcc(intial_directory, final_directory, no_of_coefficients):
    for file in os.listdir(os.path.join(initial_directory,audio_file_folder_name)):

        filepath = initial_directory+'/'+audio_file_folder_name+'/'+file;
        mfcc_image_prefix = os.path.splitext(os.path.basename(file))[0]
        #filepath = os.path.join(initial_directory, audio_file_folder_name, file);
        print("Currently processing ", file)

        y, sr = librosa.load(filepath)
        mfccs = librosa.feature.mfcc(y, n_mfcc = no_of_coefficients, sr = sr)
        #plt.figure(figsize = (25,10))
        librosa.display.specshow(mfccs, x_axis = 'time', sr = sr)
        fig = plt.gcf()
        for i, mfccs in enumerate(mfccs):

            fig.savefig(final_directory+"/"+mfcc_image_prefix+'.jpg')
```

4. Individual lists are being created for each speakers' audio MFCC images. This is to efficiently divide the dataset into train, validation, and test data and to generate class labels for each speaker.

```
train_imgs = [*speaker_0000_train, *speaker_0001_train, *speaker_0002_train, *speaker_0003_train, *speaker_0004_train]
val_imgs = [*speaker_0000_val, *speaker_0001_val, *speaker_0002_val, *speaker_0003_val, *speaker_0004_val]
train_files_with_labels = []
for file in train_imgs:
    temp_list  = [file, int(file[11])];
    train_files_with_labels.append(temp_list);
train_files_with_labels = pd.DataFrame(train_files_with_labels) # generating label for train set
train_files_with_labels.to_csv('/home/agrawal.shau/audio_dataset/train.csv');
val_files_with_labels = []
for file in val_imgs:
    temp_list  = [file, int(file[11])];
    val_files_with_labels.append(temp_list);
val_files_with_labels = pd.DataFrame(val_files_with_labels) # generating label for validation set
val_files_with_labels.to_csv('/home/agrawal.shau/audio_dataset/validation.csv');
print(f"Number of train images = {len(train_imgs)} \nValidation images = {len(val_imgs)}")
```

```
Number of train images = 1150
Validation images = 150
```

```
test_imgs = [*speaker_0000_test, *speaker_0001_test, *speaker_0002_test, *speaker_0003_test, *speaker_0004_test]
test_files_with_labels = []
for file in test_imgs:
    temp_list  = [file, int(file[11])];
    test_files_with_labels.append(temp_list);
test_files_with_labels = pd.DataFrame(test_files_with_labels) # generating labels for test set
test_files_with_labels.to_csv('/home/agrawal.shau/audio_dataset/test.csv');
print(f"Number of Test images = {len(test_imgs)}")
```

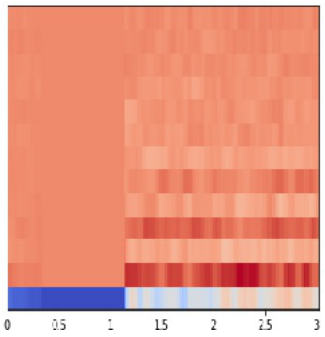
```
Number of Test images = 164
```

5. Images are then resized to 300*300 and have three channels Red, Green, and Blue. Pixels values of the images are set to be between 0 and 255 as the Deep Neural Networks best with smaller inputs. Thus scaling images between values 0 and 1.

```
[22]: # Scaling each image with values between 0 and 1
      train_imgs_scaled = train_imgs.astype('float32')
      validation_imgs_scaled = val_imgs.astype('float32')
      train_imgs_scaled /= 255
      validation_imgs_scaled /= 255
      # visualize a sample image
      print(train_imgs[0].shape)
      array_to_img(train_imgs[0])
```

```
(300, 300, 3)
```

```
[22]:
```




6. Normally, in a computer vision problem, data augmentation for images(Refer to 3.4) is applied to feed in more similar images and improve the learning of the model. But in this case, the images are the plots, and thus applying augmentation of data will be a blunder and deviate the results. In general, augmentation transformations are not applied to the test data, the original images are being sent to the model for evaluation.

7. Now, the pre-trained model of ResNet50 is loaded which has 50 layers of neural network. This is by far one of the most trusted models for image classification. The top layers, which are the classifier are not loaded as our own layers are to be connected for our task-specific classification. The weights of this pre-trained model are kept frozen.

8. As established, the initial layers extract the basic features, and the higher layers are responsible for extracting specific features (Refer 3.4). Now we use this Since Imagenet's low-level features are not the same for our case, so just training the last few fully connected layers would not give accurate predictions due to lack of full knowledge such as texture, color, shape, etc. Thus, we froze the mid-layers and trained the initial few convolutional blocks and the last fully connected layers.

   **Note**: 13 is an arbitrarily chosen number since we chose to train 3 convolutional blocks. In the future, we can change this number depending on how it affects our results, so it can be called a hyperparameter that needs tuning.

```
from tensorflow.keras.applications.resnet50 import ResNet50
from keras.models import Model
import keras
restnet = ResNet50(include_top=False, weights='imagenet', input_shape=(Img_Height,Img_Width,3))
output = restnet.layers[-1].output
output = keras.layers.Flatten()(output)
restnet = Model(restnet.input, output)
for layer in restnet.layers[13:]:
    layer.trainable = False
restnet.summary()
```

```
conv5_block3_3_conv (Conv2D)   (None, 10, 10, 2048  1050624     ['conv5_block3_2_relu[0][0]']
                               )

conv5_block3_3_bn (BatchNormal  (None, 10, 10, 2048  8192       ['conv5_block3_3_conv[0][0]']
ization)                       )

conv5_block3_add (Add)         (None, 10, 10, 2048  0           ['conv5_block2_out[0][0]',
                               )                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 10, 10, 2048  0           ['conv5_block3_add[0][0]']
                               )

flatten_1 (Flatten)            (None, 204800)       0           ['conv5_block3_out[0][0]']

==================================================================================================
Total params: 23,587,712
Trainable params: 50,944
Non-trainable params: 23,536,768
__________________________________________________________________________________________________
```

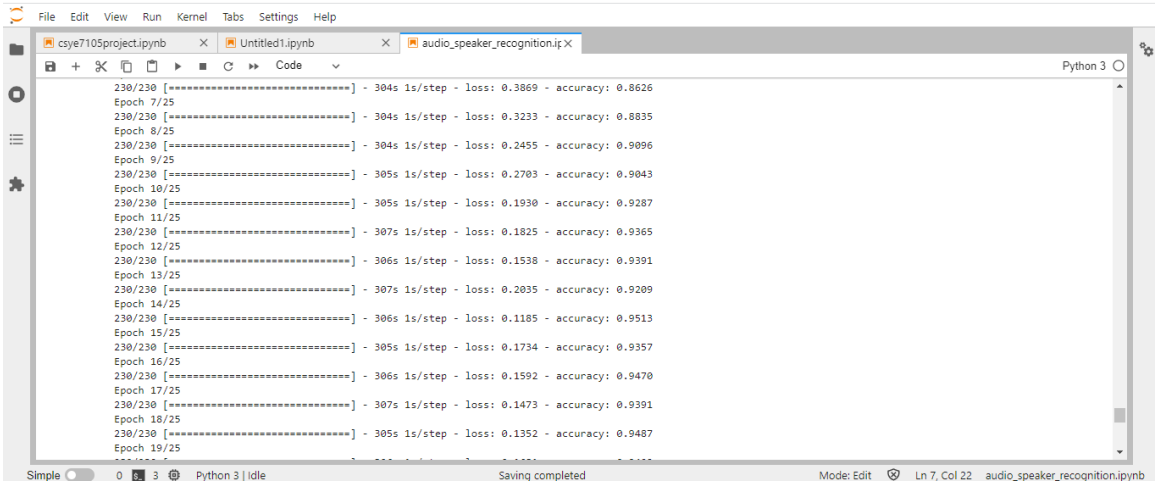
9. Now using the transfer learning, the pre-trained model is used to create our model, in which we add our own fully connected layer and the classifier using the *softmax* activation and using sparse categorical entropy as we have 5 classes to classify the images in.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
model = Sequential()
model.add(restnet)
model.add(Dense(512, activation='relu'))#, input_dim=(Img_Height,Img_Width,3)))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=optimizers.Adam(lr=1e-4),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model_1 (Functional)        (None, 204800)            23587712

 dense_9 (Dense)             (None, 512)               104858112

 dropout_6 (Dropout)         (None, 512)               0

 dense_10 (Dense)            (None, 512)               262656

 dropout_7 (Dropout)         (None, 512)               0

 dense_11 (Dense)            (None, 5)                 2565

=================================================================
Total params: 128,711,045
Trainable params: 105,174,277
Non-trainable params: 23,536,768
_________________________________________________________________
```

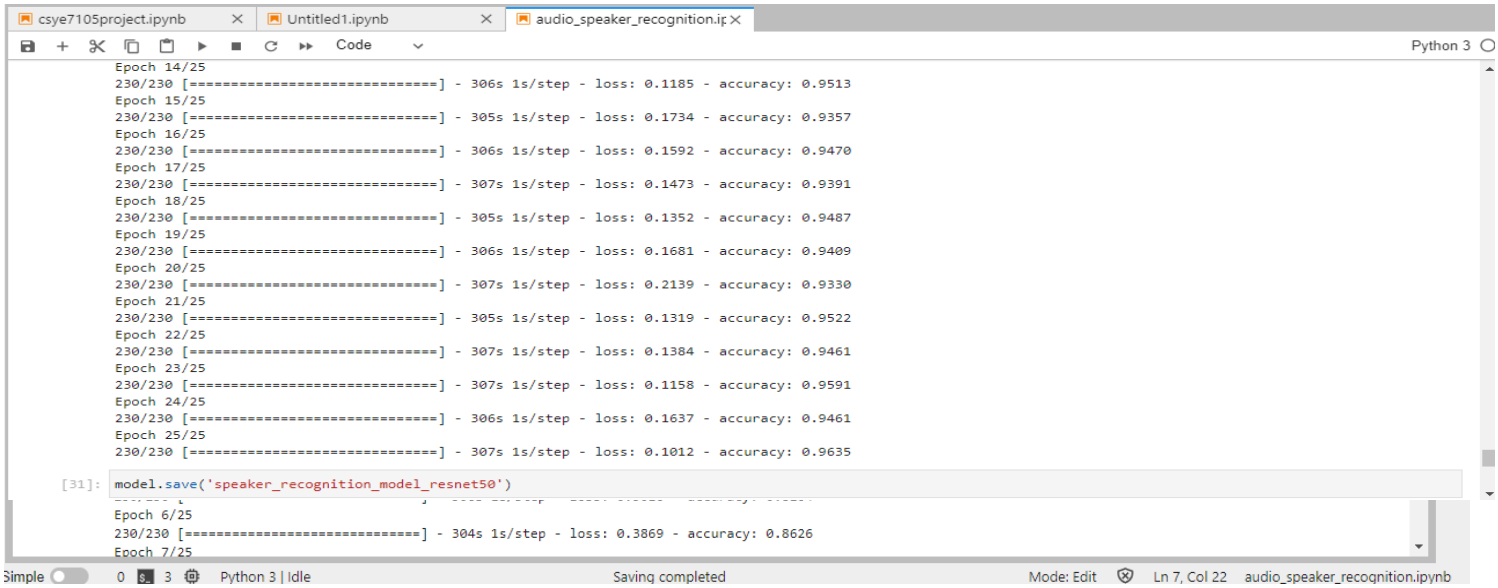
The last three layers are changed with the task-specific classifying layers.

10. Now we put the model to run and it is evident that the loss and accuracy for each *epoch*, where the *steps per epoch* for every epoch is 230.




**5. Result and Analysis**

As for the initial epochs, the accuracy was very less nearing 0.53-0.66 and loss being very high nearing 0.86. As the model learns the features of the speaker, the accuracy improves and the final accuracy turns out to be 0.9635 with a loss of 0.1012. Below shown is the progress of the model.




**6. Discovery Cluster**

The GPU partition with 32 GB of memory was used for generating the MFCC plots for the 3-second audio chunks. The Deep Learning models can be parallelized, and the calculation can be processed out faster. Multiple GPUs can be also used to run models with MFCC, LSTM, and Spectrogram for 3-second, 5-second, and 10-second audio chunks, and thus the efficiency of the models can be analyzed more extensively.

Furthermore, the model training can be parallelized with Tesnsorflow package's API. The model is parallized for 1, 2, and 4 GPUs.

**7. Scope**

These project implementations can be used to make a model learn the features of the speaker's voice by giving a variety of chunk sizes such as chunks of 5 seconds or 10 seconds. Also, the popular LSTM architecture can be used as it is a well-suited network for classifying, processing and making predictions on a time series of data.

**8. Credits and References**


- Associate Teaching Professor Handan liu, Northeastern University
- Teaching Assistant Shu-Ya Hsu, Graduate Student, Northeastern University
- Gaurav Mishra, Graduate Student, Northeastern University

- Dataset: https://www.kaggle.com/vjcalling/speaker-recognition-audio-dataset
- Librosa: https://librosa.org/doc/latest/index.html
- Tensorflow.Keras: https://www.tensorflow.org/api_docs/python/tf/keras
- Transfer Learning: https://towardsdatascience.com/deep-learning-using-transfer-learning-python-code-for-resnet50-8acdfb3a2d38
- Images from python notebook, Google Images