

ACCENT RECOGNITION USING DEEP LEARNING

Term Project

Presented to

Dr. Mark Stamp

Department of Computer Science
San Jose State University

In Partial Fulfillment of the
Requirements of the class
CS 271

By

Harshit Trehan and Akshay Kajale

April 30th, 2020

INTRODUCTION

Automatic identification of foreign accents is valuable for many speech systems, such as speech recognition, speaker identification, voice conversion etc. Accent classification refers to the problem of inferring the native language of a speaker from his or her foreign accented speech. Identifying idiosyncratic differences in speech production is important for improving the robustness of existing speech analysis systems. For example, automatic speech recognition (ASR) exhibits lower performance when evaluated on foreign accented speech. In addition to ASR applications, accent identification is also useful for forensic speaker profiling by identifying the speaker's regional origin and ethnicity in applications involving targeted marketing.

The goal in this project is to classify various types of accents, specifically foreign accents, by the native language of the speaker. This project allows to detect the demographic and linguistic backgrounds of the speakers by comparing different speech outputs with the speech accent archive dataset in order to determine which variables are key predictors of each accent. The speech accent archive demonstrates that accents are systematic rather than merely mistaken speech. Given a recording of a speaker speaking a known script of English words, this project predicts the speaker's native language.

For the purpose of this project, we have focused on countries with the most abundant audio samples, and the languages that have distinctly different origins. We chose to work with English, Arabic and Mandarin.

BACKGROUND

Since the main goal of this project is to predict the native language of the speaker, we would need a substantial amount of data in the form of audio files which will include the speaking of people belonging to different regions. The audio sample should cover all vowels and maximum number of consonants.

As per our project requirement, initially we used a dataset available on kaggle.com. On inspecting the data we found that it was incomplete and of poor quality so we decided to extract data on our own from the source at <http://accent.gmu.edu>.

“The Speech Accent Archive, is a collection of more than 2400 audio samples from people for over 177 countries speaking the same English paragraph. The paragraph contains most of the consonants, vowels, and clusters of standard American English.”

The next step was to identify categories of audio data that we wanted for our project, and we decided to stick with the categories that had the most amount of samples since Deep Learning techniques require large amounts of data. So, we decided to use English, Mandarin and Arabic audio samples.

Next, we downloaded and pre-processed the data as follows.

DATA PREPROCESSING

1. To download the data from <http://accent.gmu.edu>, we wrote a python script which downloaded all the data samples of English, Arabic and Mandarin. We also created a bio_metadata.CSV file which contains following columns:
 - a. href
 - b. language_num
 - c. native_language

	A	B	C	D
1	href	language_num	native_language	
2	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=256	mandarin1	mandarin	
3	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=257	mandarin2	mandarin	
4	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=258	mandarin3	mandarin	
5	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=259	mandarin4	mandarin	
6	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=260	mandarin5	mandarin	
7	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=261	mandarin6	mandarin	
...
371	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=749	english220	english	
372	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=754	english221	english	
373	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=764	english222	english	
374	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=767	english223	english	
375	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=768	english224	english	
376	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=771	english225	english	
377	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=772	english226	english	
378	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=773	english227	english	
379	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=774	english228	english	
...
851	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1569	arabic55	arabic	
852	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1677	arabic56	arabic	
853	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1699	arabic57	arabic	
854	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1700	arabic58	arabic	
855	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1702	arabic59	arabic	
856	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1711	arabic60	arabic	
857	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1753	arabic61	arabic	
858	http://accent.gmu.edu/browse_language.php?function=detail&speakerid=1768	arabic62	arabic	

2. First, we have to get the .wav files to and store them in a dataframe. We have used the following function to load .wav files in the program.

```
def get_wav(language_num):  
    y, sr = librosa.load('../audio/{0}.wav'.format(language_num))  
    return(librosa.core.resample(y=y, orig_sr=sr, target_sr=RATE, scale=True))
```

3. After getting .wav files, we are extracting the features like Spectral Centroid, Zero Crossing rate and MFCC. We are using the librosa library to extract these features.

```
def get_features(wav):  
    sc = librosa.feature.spectral_centroid(y=wav, sr=RATE, hop_length=int(RATE/40), n_fft=int(RATE/40))  
    zrc = librosa.feature.zero_crossing_rate(y=wav, frame_length=int(RATE/40), hop_length=int(RATE/40))  
    mfcc = librosa.feature.mfcc(y=wav, sr=RATE, hop_length=int(RATE/40), n_fft=int(RATE/40), n_mfcc=N_MFCC)  
    features = np.append(mfcc, np.append(sc, zrc, axis=0), axis=0)  
    return features
```

Here Rate is Sampling rate which we have considered as 8kHz, 16kHz and 24kHz for the experiment purpose. Hop_length is a sliding window that we have calculated as 25 ms, which is standard for feature extraction. n_fft's are a number of Fourier transforms which represent the width of the window, and are taken to be the same as hop_length. n_mfcc is the number of coefficients to be extracted from the audio sample, we have kept it as 13 out of 39 which is a standard value for any audio classification. The first 12 are cepstrum coefficients and 13th is the energy term.

4. After getting the features as a numpy matrix, we again need to divide these features in the segments as the dimensions of each matrix are not the same. We will divide each feature in segments to make their dimensionality equal.

```
def make_segments(mfccs, labels):  
    segments = []  
    seg_labels = []  
    for mfcc, label in zip(mfccs, labels):  
        for start in range(0, int(mfcc.shape[1] / COL_SIZE)):  
            segments.append(mfcc[:, start * COL_SIZE:(start + 1) * COL_SIZE])  
            seg_labels.append(label)  
    return (segments, seg_labels)
```

Here mfccs is the X dataframe and labels is the Y dataframe. Here we have kept COL_SIZE as 80 for classic and deep learning models. So after

segmenting, our X dataframe becomes a list of numpy arrays in which each array is of size 13x80 and Y becomes a list of numpy arrays where each list is of dimension 1x3, containing the binary representation of the class label, i.e., 0s in non-class columns and 1 in the correct class column.

Different Machine learning techniques have been applied to this data set to generate predictions as far as possible. Some of the classic techniques which we have implemented are Random Forest, K Nearest Neighbours, Support Vector Classifier. Also, we have implemented two Deep learning techniques namely Convolutional Neural Network(CNN) and Recurrent Neural Network (RNN).

IMPLEMENTATION

The implementation part will focus on 5 different techniques out of which 3 are classic Machine Learning algorithms and 2 are Deep Learning algorithms.

- **K Nearest Neighbour(KNN)**

KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are nearer to each other. KNN captures the idea of similarity with mathematics - calculating the distance between two points.

```
def knn_classifier(X, Y):
    print("Shape of X", X.shape)

    candidate_acc = []
    neighbors = [50, 75, 100, 125]
    p = [1, 2]

    for n in neighbors:
        for dist in p:
            classifier = KNeighborsClassifier(n_neighbors=n, metric='minkowski', p=dist)
            cv = KFold(n_splits=10, shuffle=True)

            for train_index, test_index in cv.split(X):
                X_train, X_test, Y_train, Y_test = X[train_index], X[test_index], Y[train_index], Y[test_index]
                classifier.fit(X_train, Y_train)

            scores = cross_val_score(classifier, X, Y, cv=5)
            best_acc = np.max(np.array(scores))
            candidate_acc.append(best_acc)
            print('For neighbors = %d and distance type = %d, best 10 fold cross validation accuracy = %f' % (n, dist, best_acc))

    return classifier
```

Experiments:

1. The first step here is to load the MFCC's and its corresponding labels after performing segmentation.
2. As the dimension of our data is almost 10 to the order 6, we have taken the value of k as 50, 75, 100 and 125. Also we have experimented with the value of p for Manhattan and Euclidean distance. As well with the sampling rate as 8kHz, 16kHz and 24kHz.
3. We need to try and experiment with different values of k as we keep on running the model.
4. We have performed 10 fold cross validation on our data.

Results:

Sampling Rate: 8000

```
Loading wav files....  
Feature Extraction...  
Shape of X (5057, 1300)  
For neighbors = 50 and distance type = 1, best 10 fold cross validation accuracy = 0.470356  
For neighbors = 50 and distance type = 2, best 10 fold cross validation accuracy = 0.462451  
For neighbors = 75 and distance type = 1, best 10 fold cross validation accuracy = 0.468379  
For neighbors = 75 and distance type = 2, best 10 fold cross validation accuracy = 0.458498  
For neighbors = 100 and distance type = 1, best 10 fold cross validation accuracy = 0.465347  
For neighbors = 100 and distance type = 2, best 10 fold cross validation accuracy = 0.466403  
For neighbors = 125 and distance type = 1, best 10 fold cross validation accuracy = 0.456522  
For neighbors = 125 and distance type = 2, best 10 fold cross validation accuracy = 0.450593  
  
In [2]: |
```

Best Accuracy: neighbours = 50, Distance Calculation type = Manhattan

Sampling Rate: 16000

```
Loading wav files....  
Feature Extraction...  
Shape of X (5057, 1300)  
For neighbors = 50 and distance type = 1, best 10 fold cross validation accuracy = 0.477228  
For neighbors = 50 and distance type = 2, best 10 fold cross validation accuracy = 0.463366  
For neighbors = 75 and distance type = 1, best 10 fold cross validation accuracy = 0.457426  
For neighbors = 75 and distance type = 2, best 10 fold cross validation accuracy = 0.442688  
For neighbors = 100 and distance type = 1, best 10 fold cross validation accuracy = 0.443564  
For neighbors = 100 and distance type = 2, best 10 fold cross validation accuracy = 0.445545  
For neighbors = 125 and distance type = 1, best 10 fold cross validation accuracy = 0.437624  
For neighbors = 125 and distance type = 2, best 10 fold cross validation accuracy = 0.444664  
  
In [2]:
```

Best Accuracy: neighbours = 50, Distance Calculation type = Manhattan

Sampling Rate: 24kHz

```
Loading wav files....  
Feature Extraction...  
Shape of X (5057, 1300)  
For neighbors = 50 and distance type = 1, best 10 fold cross validation accuracy = 0.473267  
For neighbors = 50 and distance type = 2, best 10 fold cross validation accuracy = 0.438735  
For neighbors = 75 and distance type = 1, best 10 fold cross validation accuracy = 0.473267  
For neighbors = 75 and distance type = 2, best 10 fold cross validation accuracy = 0.444664  
For neighbors = 100 and distance type = 1, best 10 fold cross validation accuracy = 0.469307  
For neighbors = 100 and distance type = 2, best 10 fold cross validation accuracy = 0.438735  
For neighbors = 125 and distance type = 1, best 10 fold cross validation accuracy = 0.459406  
For neighbors = 125 and distance type = 2, best 10 fold cross validation accuracy = 0.438735  
  
In [2]:
```

Best Accuracy: neighbours = 50, Distance Calculation type = Manhattan

- **Random Forest Classification**

It is an ensemble tree-based learning algorithm. The Random Forest Classifier is a set of decision trees from a randomly selected subset of training set. It aggregates the votes from different decision trees to decide the final class of the test object.

```
def rf_classifier(X, Y):
    print("Shape of X", X.shape)

    candidate_acc = []
    n_tree = [500, 750, 1000, 1250]

    for n in n_tree:
        classifier = RandomForestClassifier(n_estimators = n, criterion = 'entropy', random_state = 0)
        cv = KFold(n_splits=10, shuffle=True)

        for train_index, test_index in cv.split(X):
            X_train, X_test, Y_train, Y_test = X[train_index], X[test_index], Y[train_index], Y[test_index]
            classifier.fit(X_train, Y_train)

        scores = cross_val_score(classifier, X, Y, cv=5)
        best_acc = np.amax(scores)
        candidate_acc.append(best_acc)
        print('For number of trees = %d, best 10 fold cross validation accuracy = %f' % (n, best_acc))

    return classifier
```

Experiments:

1. The first step here is to load the MFCC's and its corresponding labels after performing segmentation.
2. As the dimension of our data is almost 10 to the order 6, we have taken the of n_estimators i.e number of forests as 500, 750, 1000 and 1250. As the KNN gave us the best accuracy at sampling rate 16kHz, we decided to keep it constant.

Results:

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/librosa/util/decorators.py:2: NumbaDeprecationWarning: An import was requested from a module that has moved location.
Import of 'jit' requested from: 'numba.decorators', please update to use 'numba.core.decorators' or pin to Numba version 0.48.0. This alias will not be present in Numba version 0.50.0.
  from numba.decorators import jit as optional_jit
Loading wav files...
Feature Extraction...
Shape of X (5057, 1500)
For number of trees = 500, best 10 fold cross validation accuracy = 0.564787
For number of trees = 750, best 10 fold cross validation accuracy = 0.572700
For number of trees = 1000, best 10 fold cross validation accuracy = 0.583581
For number of trees = 1250, best 10 fold cross validation accuracy = 0.579624
Shape of X (5057, 1500)
```

Best Accuracy: n_estimators = 1000

- **Support Vector Classifier.**

Support Vector Classifier” (SVC) is a supervised ML Algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVC algorithm, we plot each data item as a point in n-dimensional space.

```
def ovr_classifier(X, Y):
    print("Shape of X", X.shape)

    candidate_acc = []
    C = [1, 2, 3, 4, 5]
    kernel = ['linear', 'rbf']

    for C in C:
        for kernel in kernel:
            classifier = OneVsRestClassifier(SVC(kernel=kernel, C=C))
            cv = KFold(n_splits=10, shuffle=True)

            for train_index, test_index in cv.split(X):
                X_train, X_test, Y_train, Y_test = X[train_index], X[test_index], Y[train_index], Y[test_index]
                classifier.fit(X_train, Y_train)

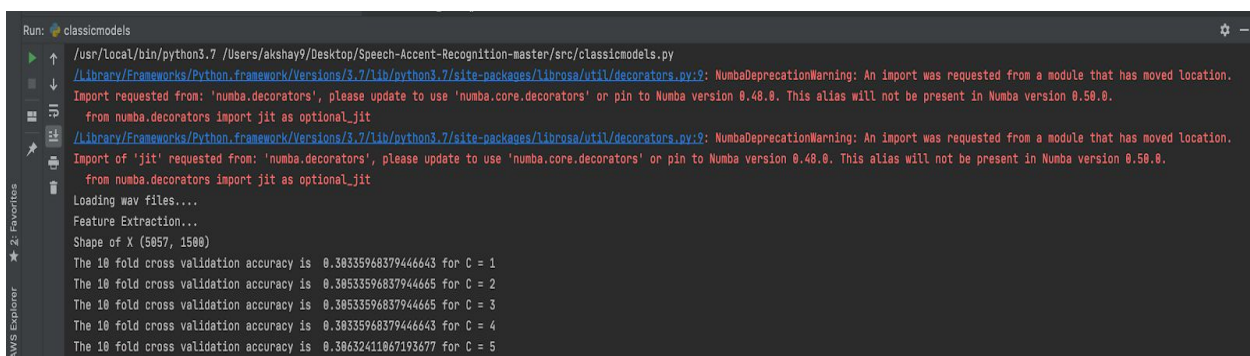
            scores = cross_val_score(classifier, X, Y, cv=10)
            best_acc = np.amax(scores)
            candidate_acc.append(best_acc)
            print('For C = %d and kernel = %s, best 10 fold cross validation accuracy = %f' % (C, kernel, best_acc))

    return classifier
```

Experiments:

1. The first step here is to load the MFCC's and its corresponding labels after performing segmentation.
2. As the dimension of our data is almost 10 to the order 6, we have experimented on the value of C as 1,2,3,4 and 5 . As the KNN gave us the best accuracy at sampling rate 16000, we decided to keep it constant.
3. Here we have used 10 fold cross validation for the data.

Result:



```
Run: classicmodels
/usr/local/bin/python3.7 /Users/akshay9/Desktop/Speech-Accent-Recognition-master/src/classicmodels.py
...
Loading wav files...
Feature Extraction...
Shape of X (5057, 1500)
The 10 fold cross validation accuracy is 0.38335968379446643 for C = 1
The 10 fold cross validation accuracy is 0.38533596837944665 for C = 2
The 10 fold cross validation accuracy is 0.38533596837944665 for C = 3
The 10 fold cross validation accuracy is 0.38335968379446643 for C = 4
The 10 fold cross validation accuracy is 0.38632411067193677 for C = 5
```

Best Accuracy at C = 5

DEEP LEARNING

As we know that audio data is very complex because of its dimensions and features, none of the classical techniques are good enough to work with audio data. Only Random Forest was able to give the best possible accuracy which was approximately 57%.

We aimed at using the classic techniques as a baseline threshold for our complex Neural Nets built with TensorFlow.

Using results from our previous experiments, we set the sampling rate at 16000 and decided to use only MFCCs to train the Deep Learning Algorithms.

Hence to get better accuracy we implemented two Deep learning techniques namely Convolutional Neural Network(CNN) and Long Short Term Memory(LSTM).

Convolutional Neural Network (CNN)

The Convolutional neural networks are regularized versions of multilayer perceptron (MLP). They were developed based on the working of the neurons of the animal visual cortex.

```

def train_model(X_train,y_train,X_validation,y_validation, batch_size=128): #64
    """
    Trains 2D convolutional neural network
    :param X_train: Numpy array of mfccs
    :param y_train: Binary matrix based on labels
    :return: Trained model
    """

    # Get row, column, and class sizes
    rows = X_train[0].shape[0]
    cols = X_train[0].shape[1]
    val_rows = X_validation[0].shape[0]
    val_cols = X_validation[0].shape[1]
    num_classes = len(y_train[0])

    # input image dimensions to feed into 2D ConvNet Input layer
    input_shape = (rows, cols, 1)

    X_train = X_train.reshape(X_train.shape[0], rows, cols, 1)

    X_validation = X_validation.reshape(X_validation.shape[0],val_rows,val_cols,1)

    print('X_train shape:', X_train.shape)
    print(X_train.shape[0], 'training samples')

    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3,3), activation='relu',
                    data_format="channels_last",
                    input_shape=input_shape))

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy',

train_model()

```

```

print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'training samples')

model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3), activation='relu',
                data_format="channels_last",
                input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

# Stops training if accuracy does not change at least 0.005 over 10 epochs
es = EarlyStopping(monitor='acc', min_delta=.005, patience=10, verbose=1, mode='auto')

# Creates log file for graphical interpretation using TensorBoard
tb = TensorBoard(log_dir='../logs', histogram_freq=0, batch_size=batch_size, write_graph=True, write_grads=True,
                 write_images=True, embeddings_freq=0, embeddings_layer_names=None,
                 embeddings_metadata=None)

# Image shifting
datagen = ImageDataGenerator(width_shift_range=0.05)

# Fit model using ImageDataGenerator
model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),
                   steps_per_epoch=len(X_train) / 32
                   , epochs=EPOCHS,
                   validation_data=(X_validation, y_validation), callbacks=[es, tb])

return model

```


CNN Training:

```
Epoch 60/150
865/864 [=====] - 20s 23ms/step - loss: 0.6533 - accuracy: 0.7197 - val_loss: 0.7762 - val_accuracy: 0.6699
Epoch 61/150
865/864 [=====] - 20s 23ms/step - loss: 0.6524 - accuracy: 0.7198 - val_loss: 0.7916 - val_accuracy: 0.6840
Epoch 62/150
865/864 [=====] - 20s 23ms/step - loss: 0.6521 - accuracy: 0.7226 - val_loss: 0.7880 - val_accuracy: 0.6800
Epoch 63/150
865/864 [=====] - 20s 23ms/step - loss: 0.6470 - accuracy: 0.7226 - val_loss: 0.7931 - val_accuracy: 0.6743
Epoch 64/150
865/864 [=====] - 20s 23ms/step - loss: 0.6516 - accuracy: 0.7198 - val_loss: 0.8021 - val_accuracy: 0.6544
Epoch 65/150
865/864 [=====] - 20s 23ms/step - loss: 0.6445 - accuracy: 0.7252 - val_loss: 0.8110 - val_accuracy: 0.6613
Epoch 66/150
865/864 [=====] - 20s 23ms/step - loss: 0.6432 - accuracy: 0.7257 - val_loss: 0.9169 - val_accuracy: 0.6510
Epoch 67/150
```

```
Epoch 5/150
865/864 [=====] - 20s 23ms/step - loss: 0.8518 - accuracy: 0.6038 - val_loss: 0.8038 - val_accuracy: 0.6671
Epoch 6/150
865/864 [=====] - 20s 23ms/step - loss: 0.8347 - accuracy: 0.6149 - val_loss: 0.8313 - val_accuracy: 0.6174
Epoch 7/150
865/864 [=====] - 21s 24ms/step - loss: 0.8246 - accuracy: 0.6222 - val_loss: 0.8082 - val_accuracy: 0.6676
Epoch 8/150
865/864 [=====] - 20s 24ms/step - loss: 0.8133 - accuracy: 0.6296 - val_loss: 0.7781 - val_accuracy: 0.6706
Epoch 9/150
865/864 [=====] - 20s 24ms/step - loss: 0.8063 - accuracy: 0.6338 - val_loss: 0.7806 - val_accuracy: 0.6601
Epoch 10/150
865/864 [=====] - 21s 24ms/step - loss: 0.7985 - accuracy: 0.6382 - val_loss: 0.7718 - val_accuracy: 0.6643
Epoch 11/150
865/864 [=====] - 20s 23ms/step - loss: 0.7929 - accuracy: 0.6423 - val_loss: 0.7816 - val_accuracy: 0.6600
Epoch 12/150
865/864 [=====] - 20s 24ms/step - loss: 0.7833 - accuracy: 0.6497 - val_loss: 0.7741 - val_accuracy: 0.6741
Epoch 13/150
865/864 [=====] - 20s 23ms/step - loss: 0.7788 - accuracy: 0.6507 - val_loss: 0.8082 - val_accuracy: 0.6556
```


Model Summary:

```
Model Summary:
Model: "sequential_1"
Layer (type)                Output Shape                Param #
-----
conv2d_1 (Conv2D)           (None, 11, 28, 32)         320
max_pooling2d_1 (MaxPooling2 (None, 5, 14, 32)         0
conv2d_2 (Conv2D)           (None, 3, 12, 64)         18496
max_pooling2d_2 (MaxPooling2 (None, 1, 6, 64)         0
dropout_1 (Dropout)         (None, 1, 6, 64)          0
flatten_1 (Flatten)         (None, 384)                0
dense_1 (Dense)             (None, 128)                49280
dropout_2 (Dropout)         (None, 128)                0
dense_2 (Dense)             (None, 3)                  387
-----
Total params: 68,483
Trainable params: 68,483
Non-trainable params: 0
-----
Test Accuracy of CNN model: 0.7525252525252525
In [2]:
```

Train Accuracy: 76.29%

Validation Accuracy: 67%

Test Accuracy: 75.252%

CNNs are suitable for classification prediction problems where inputs are assigned a class or label. They are also suitable for Audio data where there is a high dimensionality and number of features involved. Besides that CNN is best suited for images, leveraging its power to classify spoken digit sounds. It also provides Mel Coefficients which is the major feature for audio classification. Due to above reasons, we choose to implement CNN for our accent recognition.

Long Short Term Memory

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more.

```
model = Sequential()
model.add(LSTM(activation='tanh',units=64, recurrent_activation='sigmoid',
              dropout=0.5, recurrent_dropout=0, unroll = False, use_bias = True,
              return_sequences=True, stateful=False, input_shape=input_shape))

model.add(LSTM(activation='tanh',units=64, recurrent_activation='sigmoid',
              dropout=0.5, recurrent_dropout=0, unroll = False, use_bias = True,
              return_sequences=True, stateful=False, input_shape=input_shape))

model.add(LSTM(activation='tanh',units=64, recurrent_activation='sigmoid',
              dropout=0.5, recurrent_dropout=0, unroll = False, use_bias = True,
              return_sequences=True, stateful=False, input_shape=input_shape))

model.add(Flatten())
# Output layer
model.add(Dense(num_classes, activation='sigmoid'))

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

tb = TensorBoard(log_dir=logdir, histogram_freq=0, write_graph=True,
                 write_images=True, embeddings_freq=0, embeddings_layer_names=None,
                 embeddings_metadata=None)

model.fit(X_train, y_train,
        batch_size=batch_size, epochs=EPOCHS,
        callbacks = [tb], validation_data=(X_validation,y_validation))

print("\n\nModel Summary:\n")
model.summary()

return model
```

Model Summary:

```
Model Summary:
Model: "sequential_1"

```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 13, 64)	24320
lstm_2 (LSTM)	(None, 13, 64)	33024
lstm_3 (LSTM)	(None, 13, 64)	33024
flatten_1 (Flatten)	(None, 832)	0
dense_1 (Dense)	(None, 3)	2499

```

Total params: 92,867
Trainable params: 92,867
Non-trainable params: 0
6785/6785 [=====] - 4s 550us/step
Test Accuracy of RNN model: 0.5743551850318909

```

Testing Accuracy: 57.43%

Training Accuracy: 72%

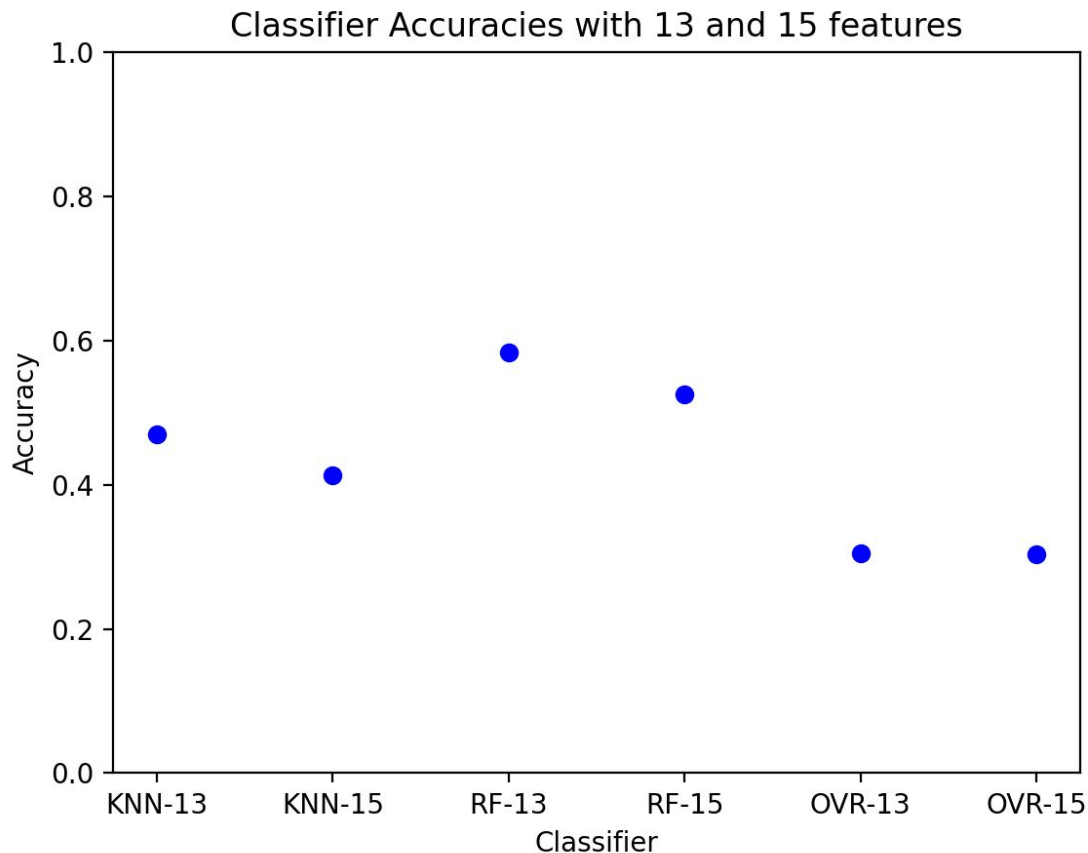
RNNs work by evaluating sections of an input in comparison with the sections both before and after the section being classified through the use of weighted memory and feedback loops. RNNs are useful because they are not limited by the length of an input and can use temporal context to better predict meaning. That is the reason why we choose RNN for our audio classification.

RESULTS AND DISCUSSION

- **Sampling Rate**

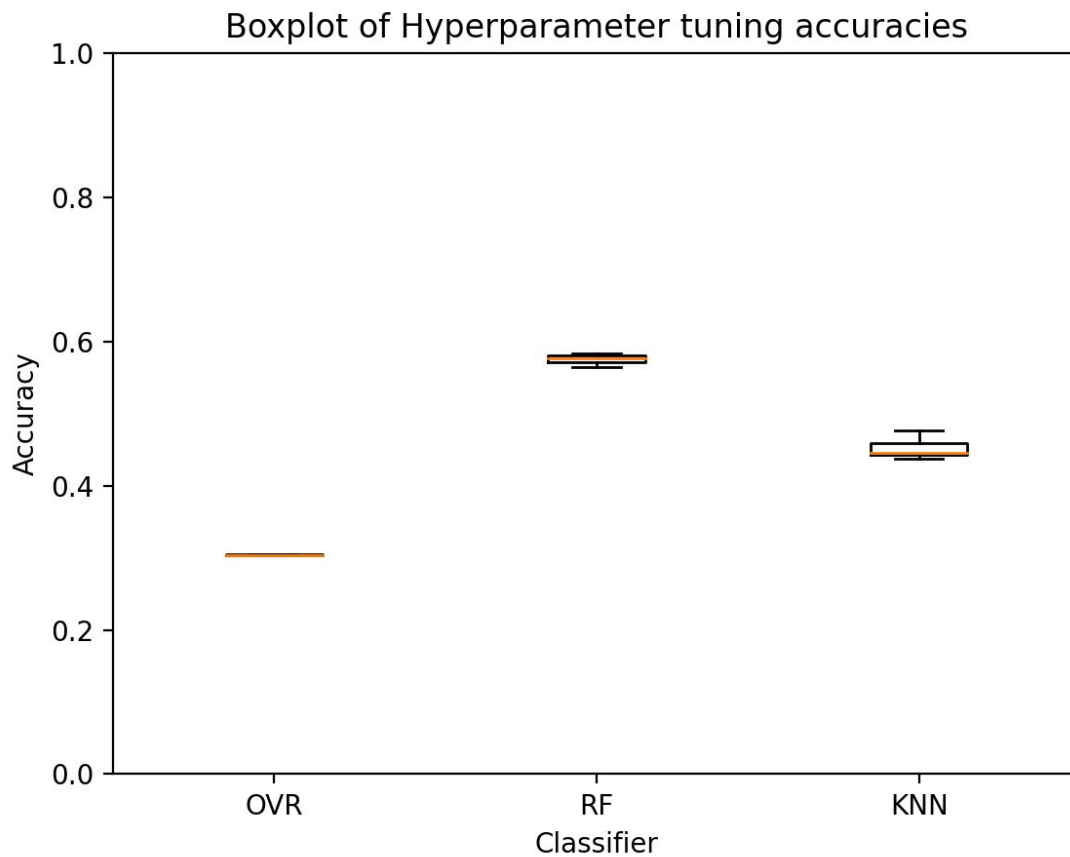
On experimenting with different sampling rates for audio files and with some research, we chose 16kHz as the sampling rate for our audio files and resamples all our audio samples to 16kHz. Experiment results using kNN classifiers pointed in this direction.

- **Classifiers vs Accuracies (Classic Techniques) with 13 and 15 Features**



Here we can observe that in the case of KNN, accuracy using 13 features is better as compared to 15. Same case is with random forest. OVR has approximately equal accuracy in both the cases. So based on these results we decided to stick with the 13 MFCC values for the rest of our experiments.

- **Boxplot of Accuracies Based on Hyperparameter Tuning:**



As you can see in the boxplot, by tuning different hyperparameters like `no_of_trees`, regularization parameter, `no_of_neighbours` etc, we get a box of accuracies. As you can see, random forest has the highest accuracy while OVR has the lowest. KNN has a visible change in accuracy after tuning `no_of_neighbours`, sampling rate and `p(euclidean/manhattan)` distance. So the best set of hyperparameters for our experimental setup for each of these classifiers is:

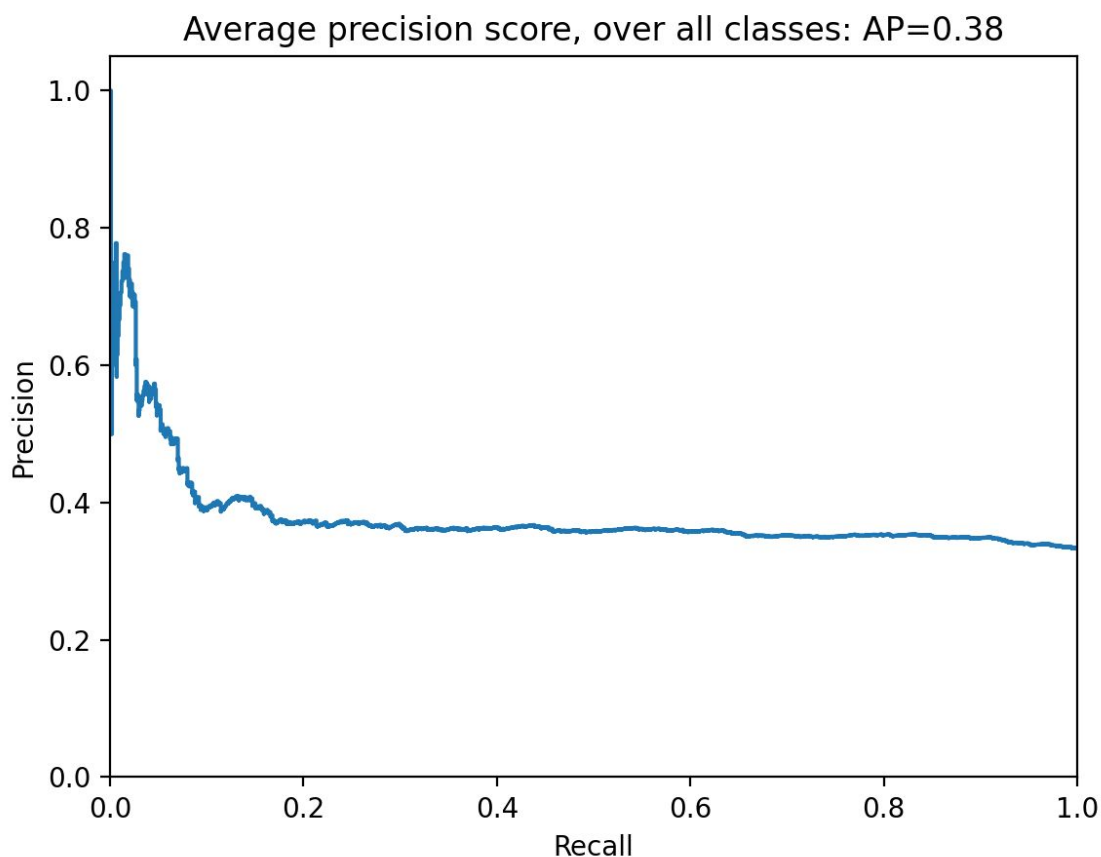
KNN: neighbours = 50, Distance Calculation type = Manhattan

Random Forest: Decision Tree Numbers = 1000

SVC: Kernel = linear, C = 5

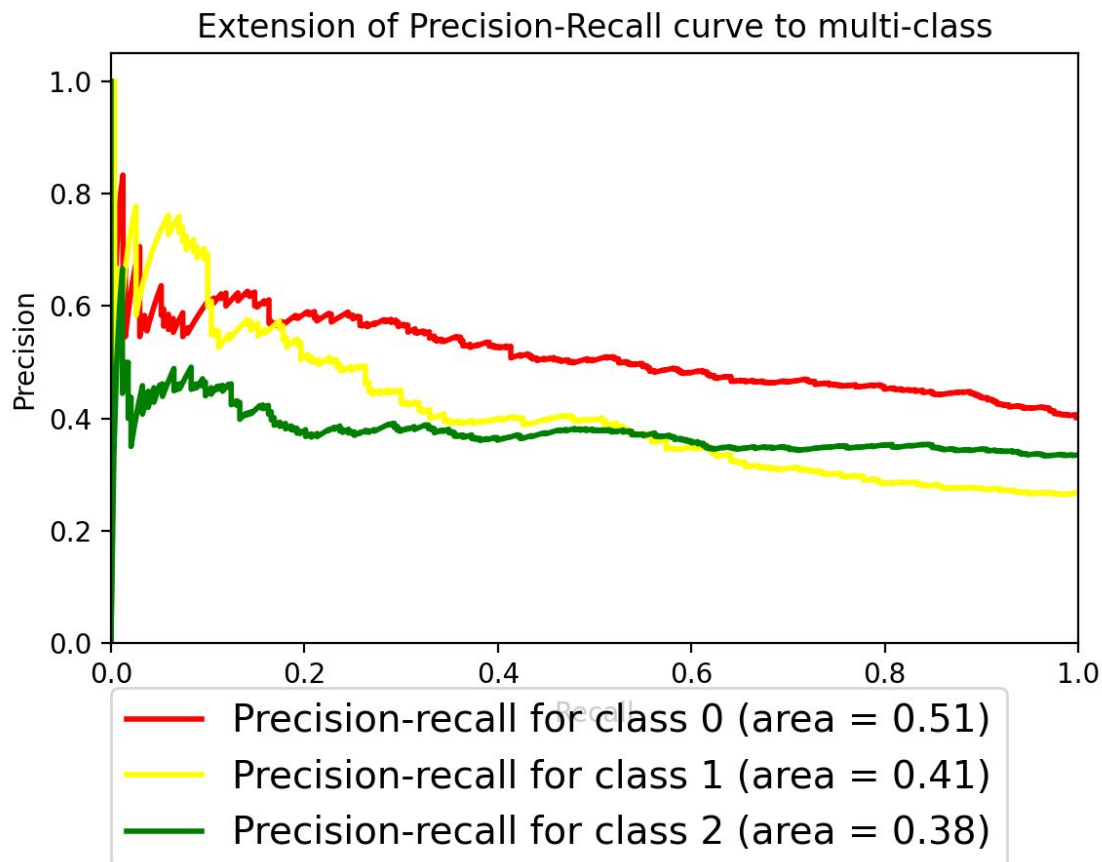
- **Calculating Precision-Recall using OVR (OneVsRestClassifier)**

Since our problem was not a binary classification problem but rather a multi-class problem, we were unable to properly generate a confusion matrix and get the Precision-Recall statistics. However, Scikit-learn library has an ensemble model, `OneVsRestClassifier` using which we can specify a binary classifier (we used **LinearSVM**) and give the number of classes, the model creates classifiers equal to the number of our label and fits each of them to one class as a One vs rest binary classifier. So we took advantage of that and calculated Precision-Recall scores.



The Average Precision-Recall scores were: 0.38

Again, this shows that classic ML techniques are not a good fit for problems such as speech recognition and classification.



We also got average precision-recall scores for each label.

Class 0: Arabic: PR Score = 0.51

Class 1: English: PR Score = 0.41

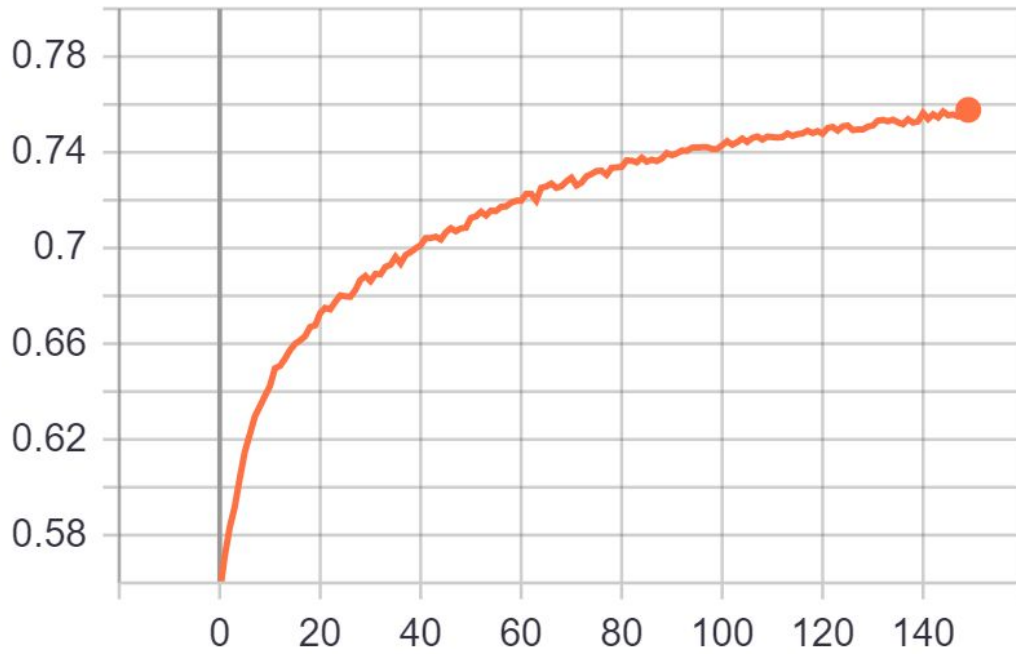
Class 2: Mandarin: PR Score = 0.38

Here, we see that relatively, Arabic accent/native language is a little easier to classify as compared to Mandarin accents.

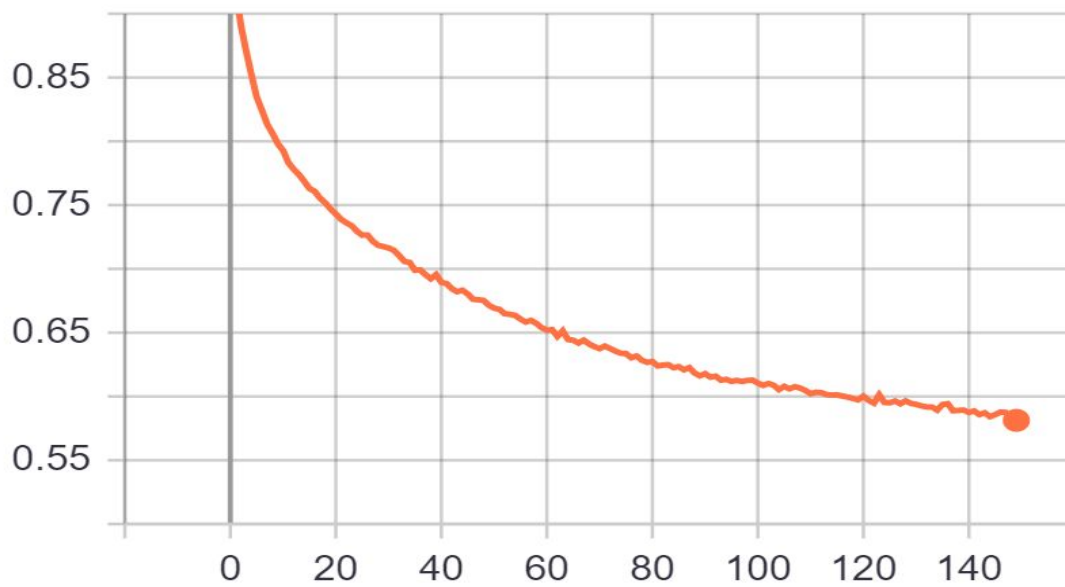
CNN RESULTS

- Training Metrics

epoch_accuracy



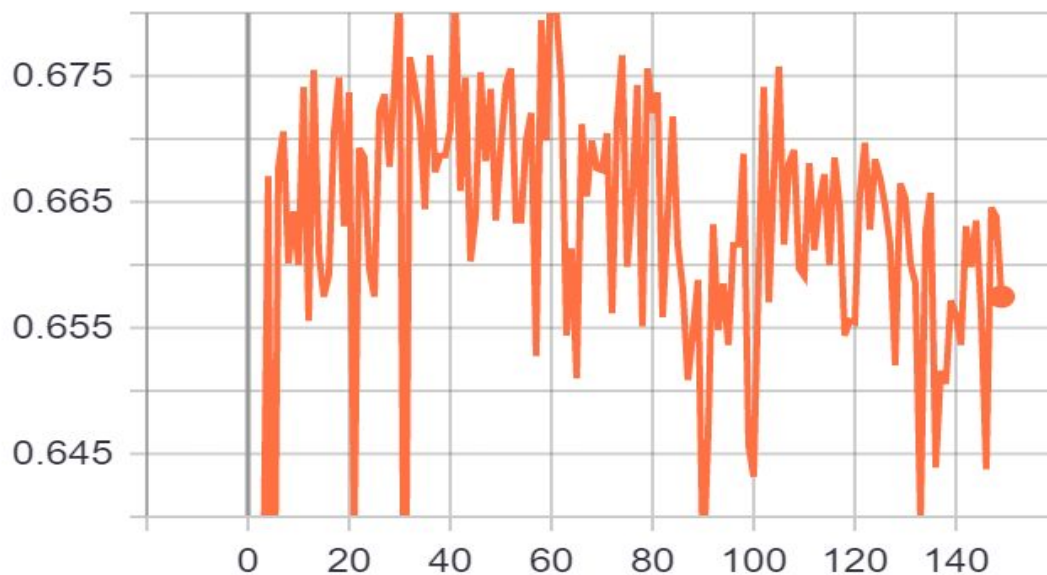
epoch_loss



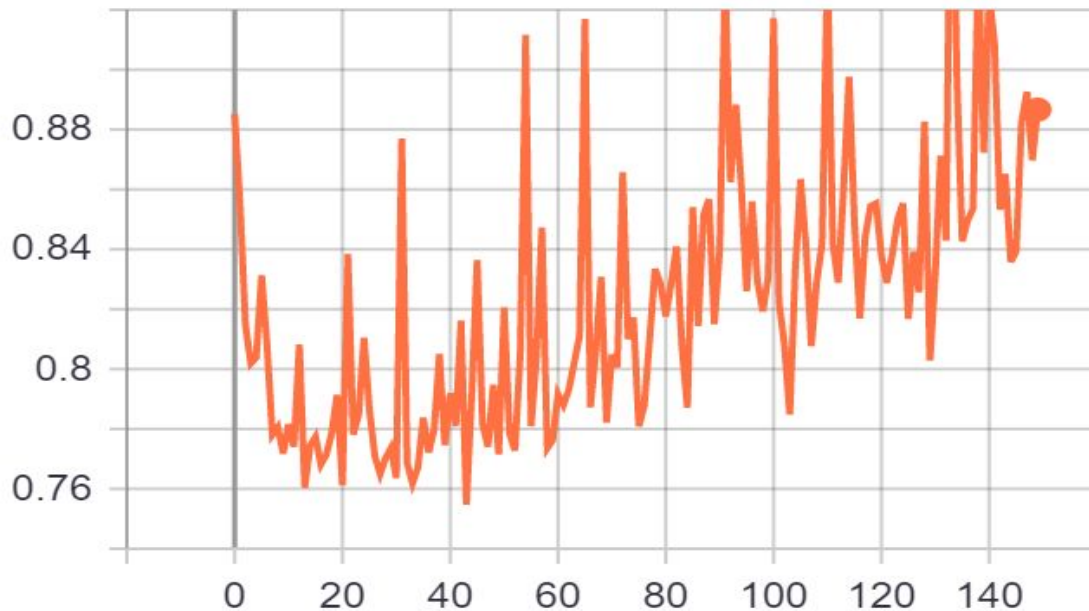
From the above graphs we can see that the training accuracy increased consistently and the loss function value decreased which is a good sign for the classifier. But since training accuracy doesn't really give a lot of information about the Neural Net's performance, we won't discuss it.

Validation Metrics:

epoch_accuracy



epoch_loss



From the graphs above we can see that the validation accuracy went up to 68% around 110 epochs but then it took a dip. Similarly, loss function reached a minimum at around 40 epochs, but then it went up. This shows some signs of overfitting in the model.

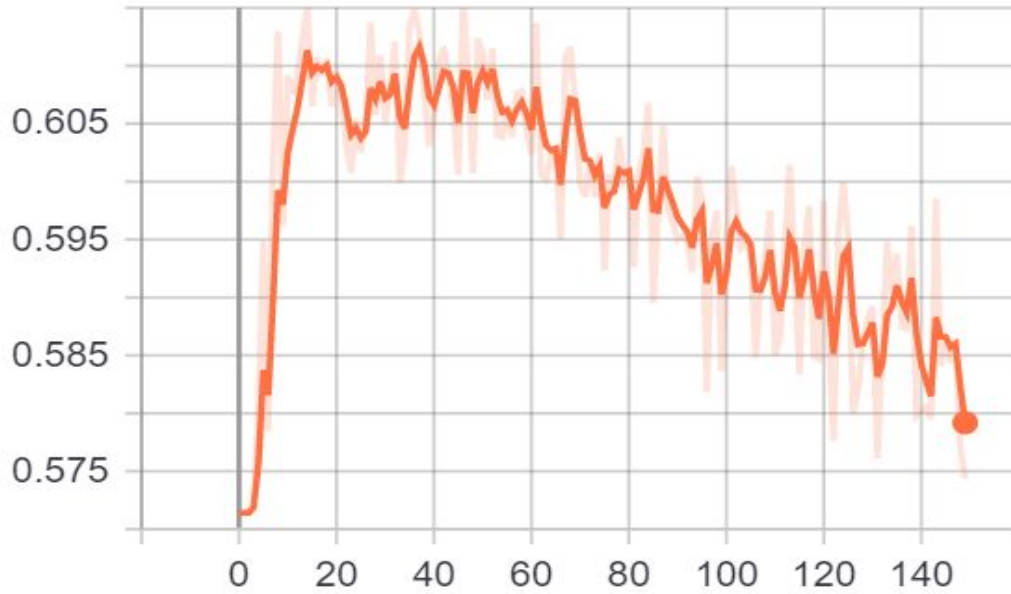
These results were not as promising as expected but on the testing data, we got test accuracy of 75% for audio files.

Batch testing and disparity in amount of data seem to be the factors for lower validation testing but increase in testing accuracy, as some batches which contain data from English audio samples have larger numbers of training data and their accuracy is higher, where Mandarin samples are comparatively limited and so their accuracy is lower, but advantage of advantage increases the testing accuracy.

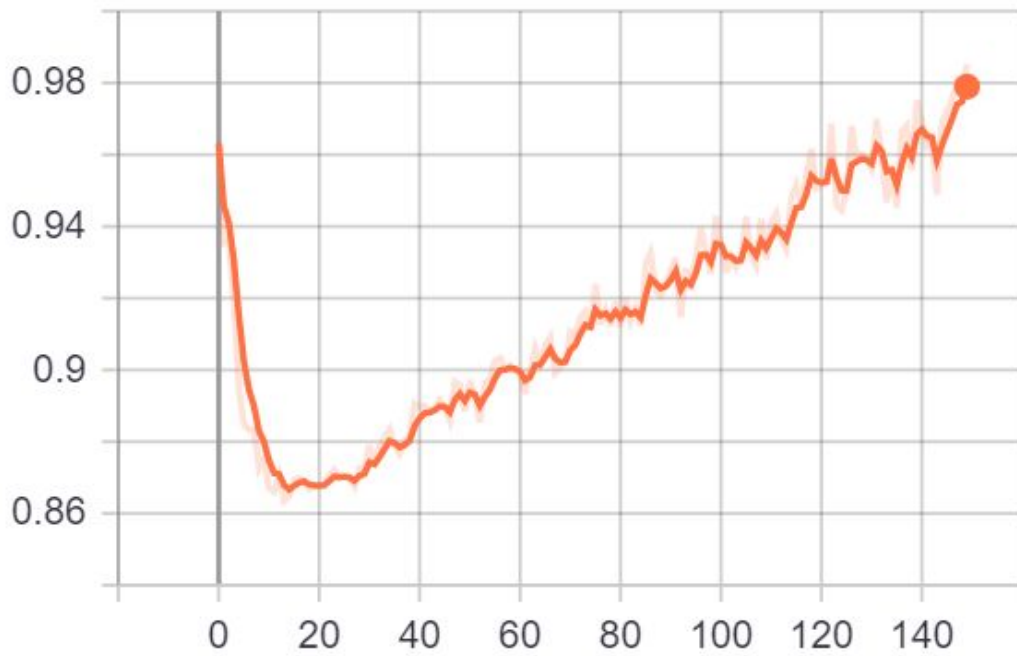
RNN-LSTM RESULTS

Validation Metrics:

epoch_accuracy



epoch_loss



As we can see in the graphs above, the validation accuracy took a huge dip after about 40-50 epochs. Similarly the value of loss function spiked after some 15 epochs.

These are really disappointing results and clearly the model is overfitting the data. The RNN model is computationally really expensive and it took a long time for it to run and we were unable to fully tune it and failed to improve the results for this. Theoretically, and practically as well, LSTM architecture models are one of the best ones for speech audio data but our model only gave 57% on testing data despite batching. This is much less than the CNN model and way less than what was expected.

CONCLUSION

Our results are not satisfactory. The classic ML techniques performed as expected but Neural networks, especially RNN-LSTM underperformed and in the future we will tweak the parameters, features and input vectors to improve the performance. There was overfitting of data in both the CNN model as well as RNN model.

Scores of ML techniques were slightly improved using Hyperparameter optimization and classic ML techniques don't work well on speech audio data, except HMMs.

However, test scores of CNNs were promising and practically, CNNs have proved to give similar performances and can be considered satisfactory.

FUTURE WORK

We would like to obtain better quality data and improve our LSTM model by tweaking the parameters and finding the right permutations for better results. Also, overfitting was a problem and we would like to eliminate that.

Additionally, historically, HMMs have been used for decades for speech recognition tasks and we wanted to implement that but it is really hard to build and train HMM models for speech recognition as they require huge amounts of data and vast computational resources.

References:

[1]https://medium.com/@jonathan_hui/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9

[2]https://www.tensorflow.org/api_docs

[3]Kashif, Kaleem, Wu, Yizhi, and Michael, Adjeisah. *Proceedings of the 2019 The World Symposium on Software Engineering - WSSE 2019*. ACM, 2019. The 2019 The World Symposium. Web.

[4]https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html

[5] Rizwan, Muhammad, and Anderson, David V. "A Weighted Accent Classification Using Multiple Words." *Neurocomputing* 277 (2018): 120-28. Web.

[6]<https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a9>