Speech Recognition using Convolutional Neural Network

By
Rachana Swamy (rms816)
Shubham Panchadhar (sp5047)
Vidyarini Kanagasabapathi (vk1198)

Table of Contents

PROBLEM STATEMENT

To implement simple Speech Recognition System which can detect and classify spoken digits (0-9) using Convolutional Neural Network.

INTRODUCTION

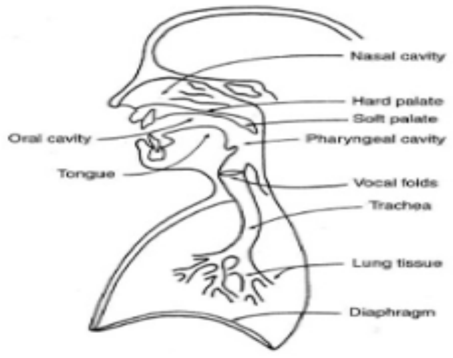
Speech recognition is the process of identifying spoken words by a speaker based on some features extracted from the voice samples. Information such as words, accent, gender, etc. can be obtained by using Speech processing.

There are two types of Speech recognition systems. One is the “speaker independent” system and the other is the “speaker dependent” system. Speaker independent system does not require training of data for speech recognition whereas “speech dependent system” needs training data samples to train the model in identifying the speech of a person. The training of data improves the accuracy of the model to recognize better.

Few uses of speech processing include- Domestic appliance control, Call Routing, Speech-to-text convertor and in simple Data Entry applications.

FEATURE EXTRACTION

The first step in any automatic speech recognition system is to extract necessary features. Feature extraction basically involves determining the important components of the audio signal that can help in identifying the linguistic content and discard all the other stuff/noise which carries information like background noise, emotion etc.



It is important to understand a speech signal and its generation in order to understand the extraction of audio features. Speech sound is generated by humans using the vocal folds and is filtered by the shape of the vocal tract (including tongue, oral cavity, larynx, etc.). The sound depends on this shape. Hence, in order to get an accurate representation of the phoneme being produced, we must be able to determine the shape of the vocal tract. The shape of the vocal tract exhibits itself in the envelope of the short time power spectrum and the MFCCs accurately represents this envelope.

MEL FREQUENCY CEPSTRAL COEFFICIENTS

Mel Frequency Cepstral Coefficients (MFCCs) are widely used for feature extraction in automatic speech and speaker recognition. The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear.

The formula for converting from frequency to Mel scale is:

$$M(f) = 1125 \ln(1 + f/700) \qquad (1)$$

To go from Mels back to frequency:

$$M^{-1}(m) = 700(\exp(m/1125) - 1) \qquad (2)$$

Steps to calculate MFCCS:

1. Frame the signal into short frames.

   An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn't change much. This is why we frame the signal into 20-40ms frames. If the frame is much shorter, we don't have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame.

2. For each frame calculate the periodogram estimate of the power spectrum.

   This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates (which wobbles small hairs), different nerves fire informing the brain that certain frequencies are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.

3. Apply the Mel filterbank to the power spectra, sum the energy in each filter.

   The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition (ASR). In particular the cochlea cannot discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason, we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filterbank: the first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The Mel scale tells us exactly how to space our filterbanks and how wide to make them. See below for how to calculate the spacing.

4. Take the logarithm of all filterbank energies.

   This is also motivated by human hearing: we don't hear loudness on a linear scale. Generally, to double the perceived volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes our features match more closely what humans actually hear. Why the logarithm and not a cube root? The logarithm allows us to use cepstral mean subtraction, which is a channel normalization technique.

5. Take the DCT of the log filterbank energies and keep DCT coefficients 2-13, discard the rest.

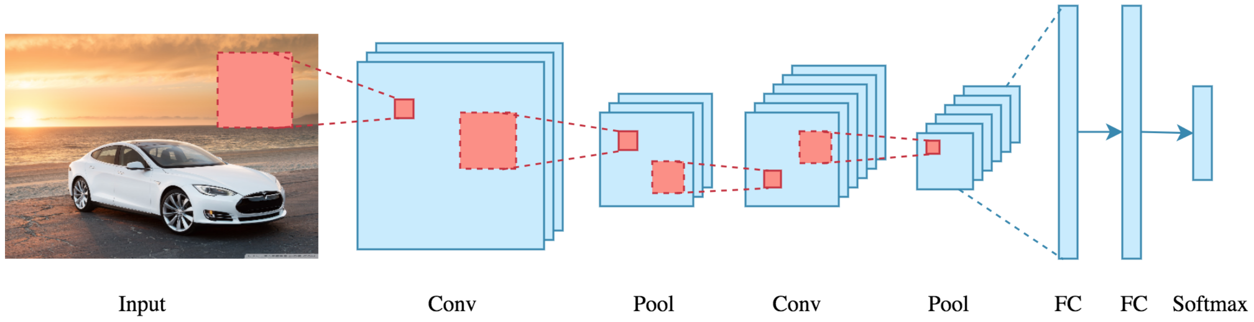
   The final step is to compute the DCT of the log filterbank energies. There are 2 main reasons this is performed. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade ASR performance, so we get a small improvement by dropping them.

INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS:

Convolutional Neural Networks (CNN) are arguably the most popular deep learning architecture frameworks. Increased interests in deep learning are due to its immense popularity and effectiveness of convnets. CNNs are now the go-to model for every image/audio/video related problem. One of the main advantages of a Neural Network model in comparison to simpler Classifiers like SVMs or Naive Bayes is that it automatically detects the most important features without any human supervision. For example, given any audio dataset, it can distinctly extract necessary features including spectrograms and other necessary information like pitch, intensity, etc. CNNs are also computationally efficient. It uses special convolution and pooling operations to perform parameter sharing. This enables CNN models to run on any device, making them universally attractive.

As we are dealing with audio files, they follow the principle of spatiality i.e. positions of relative data points matters a lot. Spectrograms like MFCC capture variations in audio waveforms and at the same time preserve spatial interdependence. Convolutional Neural Networks are perfect in the sense that they follow the above principles and conserving and recording variations at the same time.

Architecture of CNN :



1. Input Layer: The input to a CNN architecture is generally a Tensor which can be either an image/audio/video.

2. A hidden layer is what transforms the inputs to generate and extract more complex features from the data for the output layer to make a better assessment. Most of the real-world classification problems are solved using 2 hidden layers, situations where 3 or more hidden layers are used is quite rare.

3. The convolutional layer is the heart of CNN architecture. Convolution is a mathematical operation to merge/convolve two different sets of information. The convolution is applied on the input data using a convolution filter/kernel to produce a feature map. But in reality, these convolutions are performed in 3D. We perform multiple convolutions on an input, each using a different filter and resulting in a distinct feature map. We then stack all these feature maps together and that becomes the final output of the convolution layer.

4. Pooling operations are used to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats overfitting. The most common type of pooling is max pooling which just takes the max value in the pooling window while preserving the feature map

5. Rectified Linear Unit (ReLU) -is an activation function to capture all non-linearity in the dataset. Signals less than 0 or negatives will be dropped while preserving the gradient descent even when the input layer size increases

6. Fully connected layer assigns weights to individual pooled functions and outputs a sum product of these max pooled activations to provide a dense activation function.

Apart from the general architecture of CNN, our model incorporates additional feature operations to improve accuracy. We have defined the purpose of the operations below and our final model code is provided in the **Implementation section**.

1. Dropout: Overfitting the dataset can be avoided by making the CNN architecture more generalized. Dropout is the most popular regularization technique which achieves significant improvements in accuracy.This is done by switching off neurons which are randomly firing during the training process.Dropout prevents the network to be dependent on smaller number of neurons and forces all neurons to operate independently.

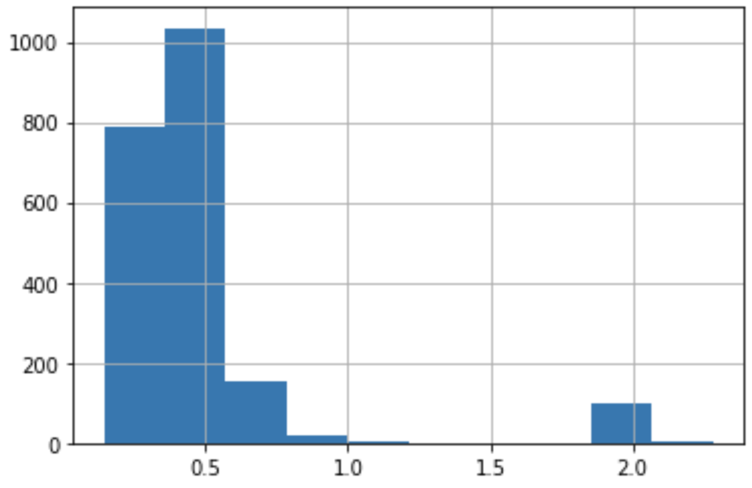
2. Adam Optimizer is used in our model to improve the learning of weights and optimize the architecture as effectively as possible. Adam optimizers are becoming quite popular in deep learning architectures due to its very low memory requirements and it's less dependency on hyperparameter tuning.

3. Batch Normalization allows each layer of the network to learn by itself a little bit more independently of other layers

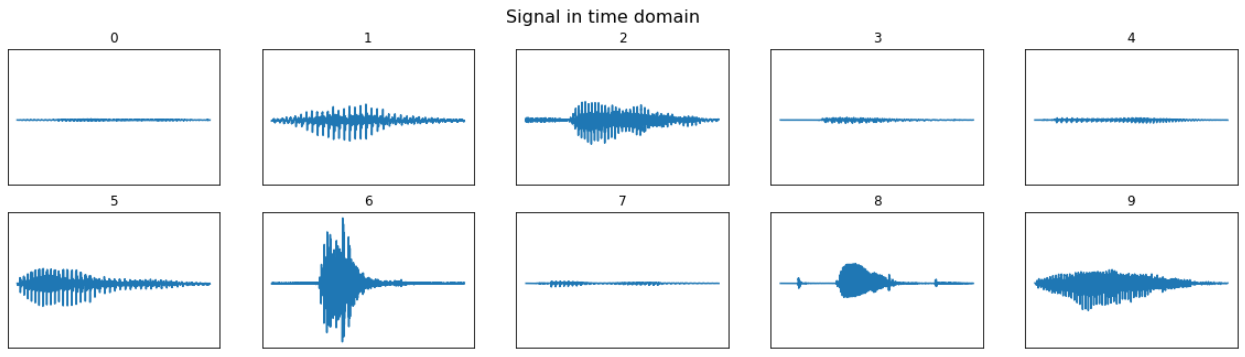
4. Categorical Cross entropy Loss function: Since we are essentially dealing with a multi-class classification problem, the natural choice for a loss function would be a categorical cross-entropy. It evaluates the Neural Network efficiency

DATA

For the project, to train our CNN model we used audio samples extracted from two sources: the publicly available "**Free Spoken Digit Dataset**" for digits and the audio samples of the 10 digits (0-9) recorded in our voices. The audio samples consist of a total of 2100 recordings by 5 speakers in wav files with a sampling rate of 8kHz. Most of the wav files have a duration of less than 0.5 seconds.



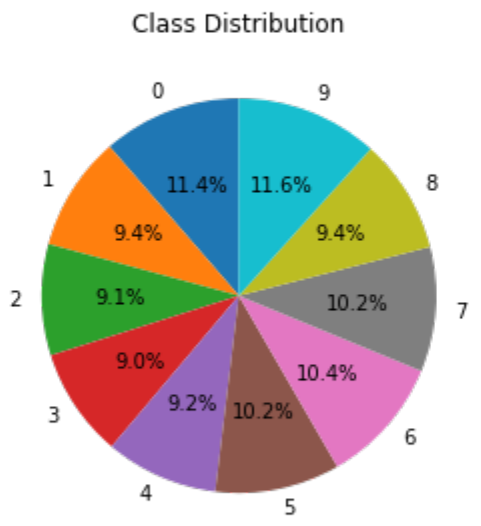
We also took an audio recording at random of each digit to plot the signal in time domain and view the class distribution of the samples based on the average length of the audio file.



Data Exploration:

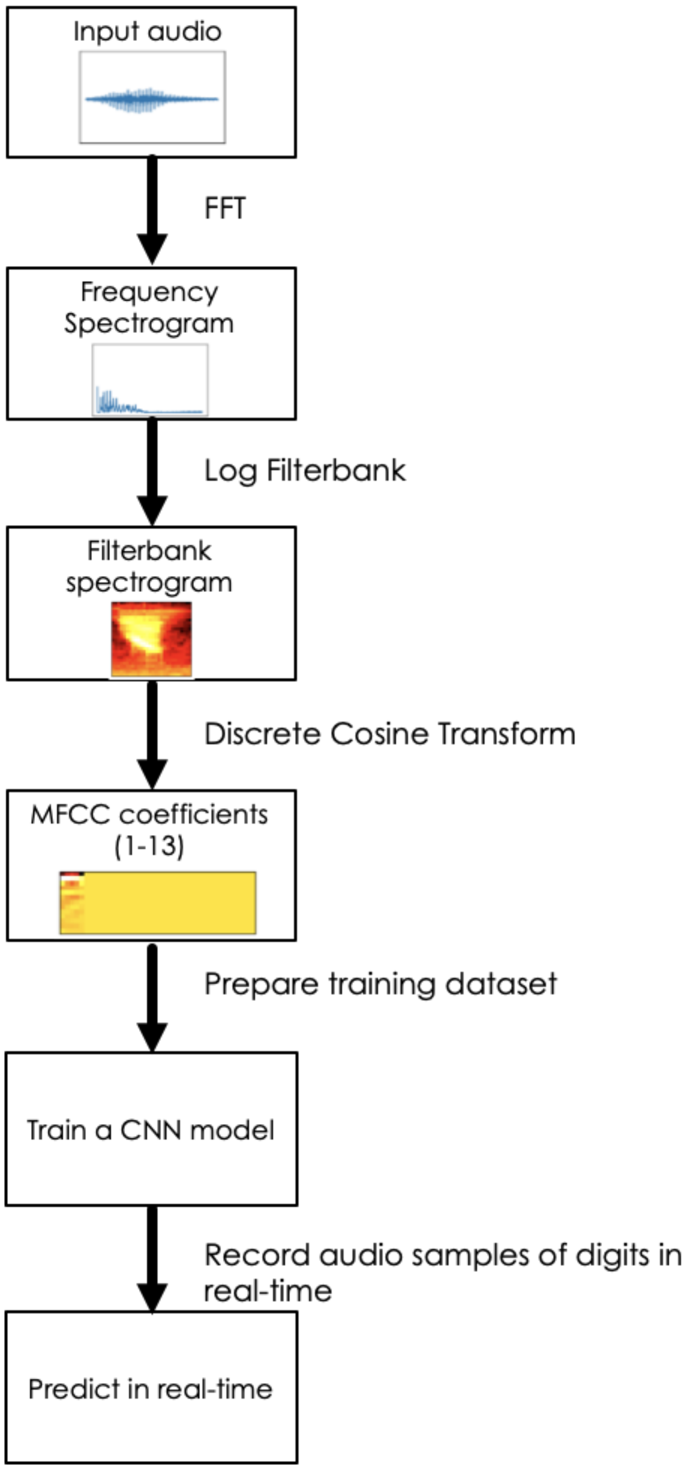
After labelling the files according to the digit, we calculated the average length of the audio files of each class.

Digit	Average length
0	0.543734
1	0.450204
2	0.435789
3	0.428216
4	0.442020
5	0.489369
6	0.499733
7	0.488319
8	0.448235
9	0.556947



IMPLEMENTATION

FLOWCHART



FEATURE EXTRACTION CODE SNIPPETS:

The below code snippet is used to compute the FFT and filter bank coefficients. We will be using Python's librosa library to first convert the analog audio signal to digital NumPy array. The digital NumPy array for each audio file will then be passed to an envelope function to remove dead spots in the recordings and adjust for ambient noise.

After computing the FFT and frequency spectrogram of the signal, we will compute the Log Filter bank coefficients and applying DCT to the resulting spectrogram.

```
def calculate_fft(y, rate):
    n = len(y)
    freq = np.fft.rfftfreq(n, d=1/rate)
    mag_y = abs(np.fft.rfft(y)/n)  #normalise value by dividing by length
    return mag_y, freq

# remove dead spots in the data
def envelope(y, rate, threshold):
    mask = []
    y = pd.Series(y).apply(np.abs)
    ymean = y.rolling(window=int(rate/10), min_periods = 1, center = True).mean()
    for mean in ymean:
        if mean>threshold:
            mask.append(True)
        else:
            mask.append(False)
    return mask
```

```
signals = {}
fft = {}
fbank = {}
mfccs = {}

for c in classes.index:
    wav_file = df1[df1.Digit == c].iloc[0,0]
    print(wav_file)
    signal, rate = librosa.load('../Data/'+wav_file,sr = None)
    mask = envelope(signal, rate, 0.001)
    signal = signal[mask]
    signals[c] = signal
    fft[c] = calculate_fft(signal, rate)

    fbank[c] = logfbank(signal, rate, nfilt=26).T
    #mfccs[c] = mfcc(signal,rate,numcep=13,nfilt=26).T
    mfccs[c] = getmfcc(signal)
    #mfccs[c] = librosa.feature.mfcc(signal, sr=8000 ,n_mfcc=13)
```

The below function is used to get the MFCC coefficients of any of the audio wav files present in our dataset.

```
def getmfcc(y, max_pad_len=40):
    mask = envelope(y, 8000, 0.001)
    y = y[mask]
    mfcc = librosa.feature.mfcc(y, sr=8000 ,n_mfcc=13)
    pad_width = max_pad_len - mfcc.shape[1]
    if pad_width>=0:
        mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc = mfcc[:,0:40]
    return mfcc
```

The first 13 MFCC coefficients are used for feature extraction since it contains majority of the information signal. To prepare our final Tensor to be passed to the CNN model, we then prepare a single MFC array consisting of MFCCs of all the audio wav files by using the below function:

```
def get_data(x):
    y = []
    mfccs = []
    for c in x.index:
        signal, rate = librosa.load('../Data/'+c,sr = None)
        y.append(x.Digit[c])
        mfccs.append(getmfcc(signal))
    return np.asarray(mfccs), to_categorical(y)
```

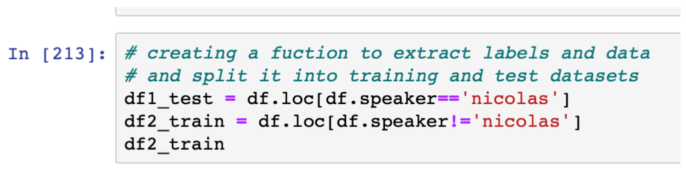
The above function also computes our classification output y using the to_categorical function which computes individual probabilities of all the digits present in any wav file.

Once we have obtained our Tensor of MFC array and our classification output y, we need to divide them intro training dataset and test dataset. Our training dataset will be used to train and fit our CNN model and our testing dataset will be used to predict, optimize parameters and improve our overall accuracy while reducing the validation loss.

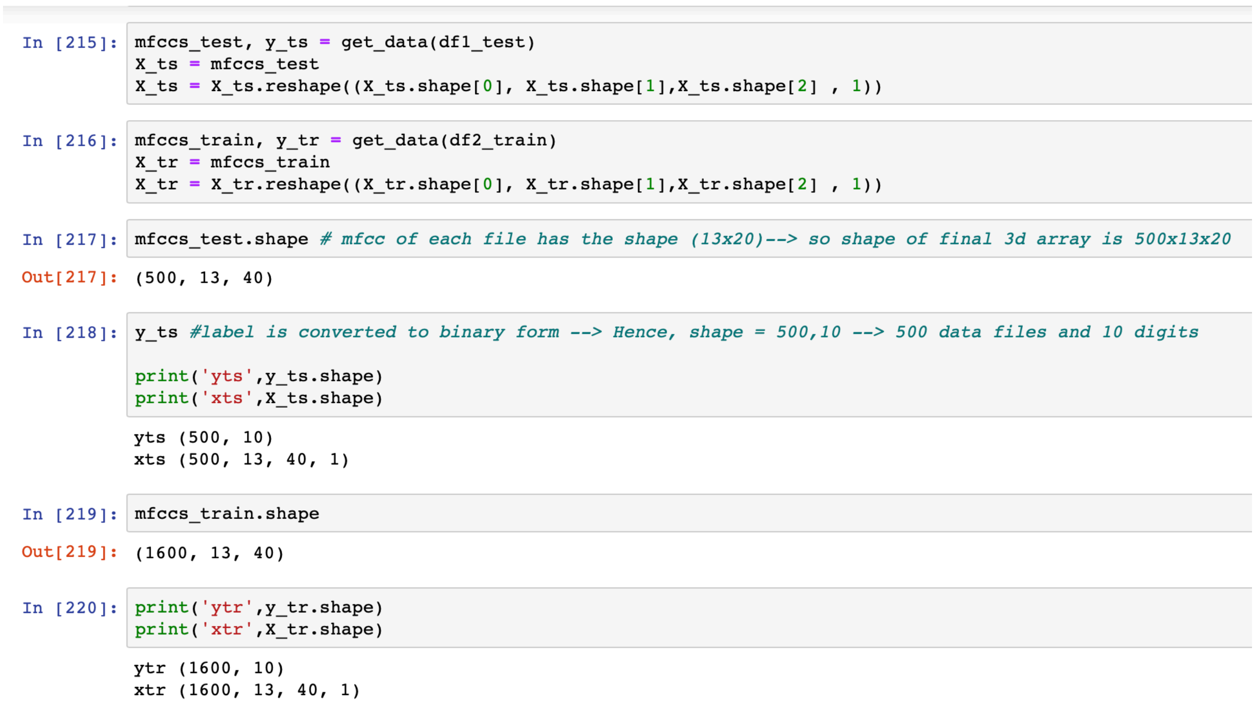
Instead of using the traditional test_train split function provided by the sklearn library we have planned to use a different splitting approach. We will be using 4 of our speakers' wav files as training dataset and our last speaker will form a part of the testing dataset.

Then, we will record our own voices and check how our classifier works for real-time audio data.

Necessary appropriations are done to obtain a Tensor that will be fed to the CNN model. The shape of each tensor (X_train, X_test) can be found in the code provided.

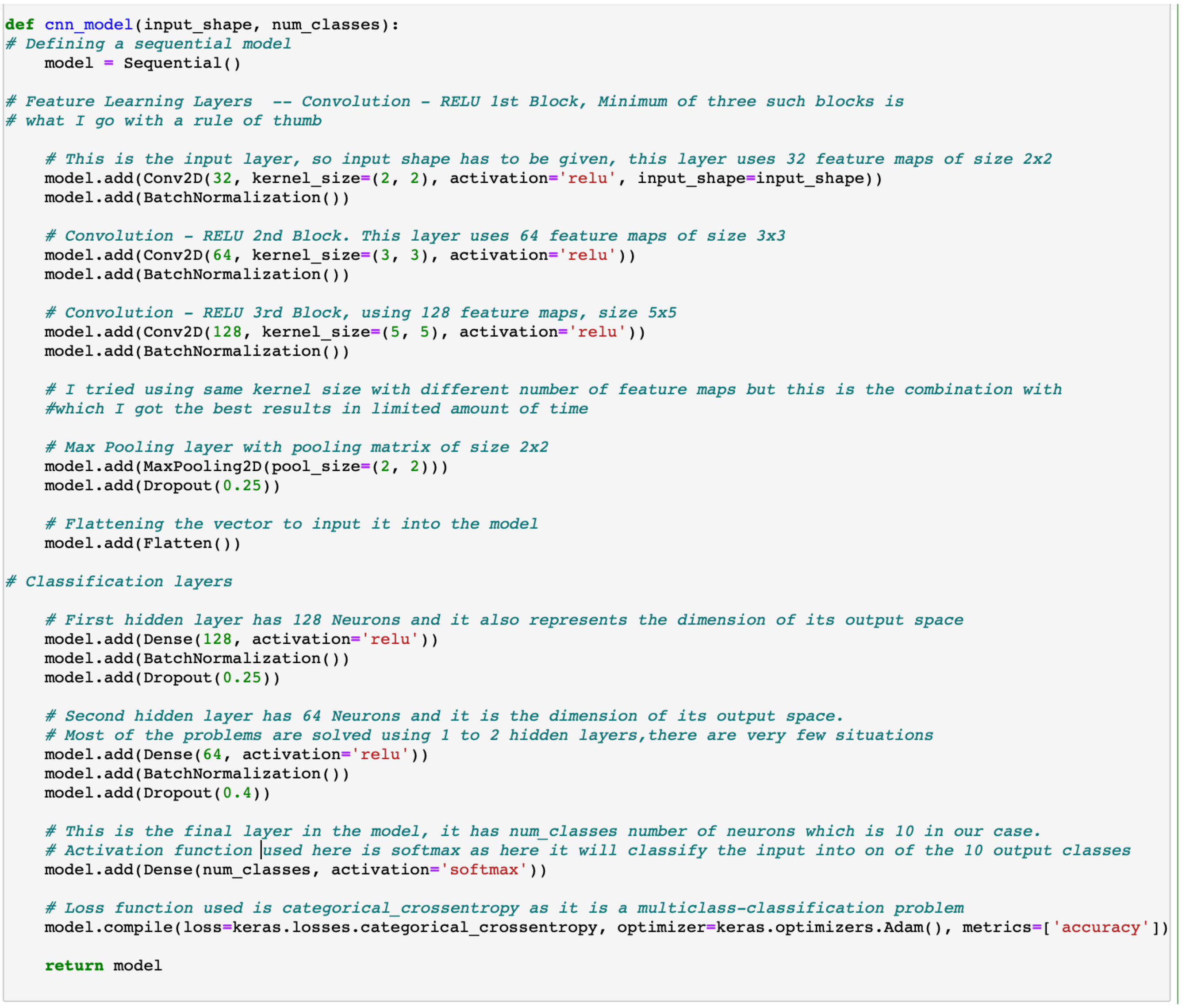


```
In [213]: # creating a fuction to extract labels and data
          # and split it into training and test datasets
          df1_test = df.loc[df.speaker=='nicolas']
          df2_train = df.loc[df.speaker!='nicolas']
          df2_train
```



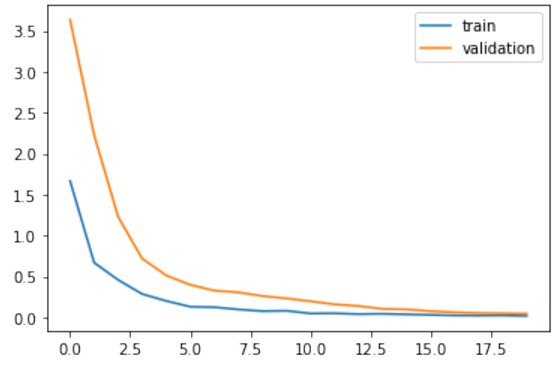
```
In [215]: mfccs_test, y_ts = get_data(df1_test)
          X_ts = mfccs_test
          X_ts = X_ts.reshape((X_ts.shape[0], X_ts.shape[1],X_ts.shape[2] , 1))

In [216]: mfccs_train, y_tr = get_data(df2_train)
          X_tr = mfccs_train
          X_tr = X_tr.reshape((X_tr.shape[0], X_tr.shape[1],X_tr.shape[2] , 1))

In [217]: mfccs_test.shape # mfcc of each file has the shape (13x20)--> so shape of final 3d array is 500x13x20

Out[217]: (500, 13, 40)

In [218]: y_ts #label is converted to binary form --> Hence, shape = 500,10 --> 500 data files and 10 digits

          print('yts',y_ts.shape)
          print('xts',X_ts.shape)

          yts (500, 10)
          xts (500, 13, 40, 1)

In [219]: mfccs_train.shape

Out[219]: (1600, 13, 40)

In [220]: print('ytr',y_tr.shape)
          print('xtr',X_tr.shape)

          ytr (1600, 10)
          xtr (1600, 13, 40, 1)
```

CNN model code snippet:



```
def cnn_model(input_shape, num_classes):
# Defining a sequential model
    model = Sequential()

# Feature Learning Layers  -- Convolution - RELU 1st Block, Minimum of three such blocks is
# what I go with a rule of thumb

    # This is the input layer, so input shape has to be given, this layer uses 32 feature maps of size 2x2
    model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=input_shape))
    model.add(BatchNormalization())

    # Convolution - RELU 2nd Block. This layer uses 64 feature maps of size 3x3
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization())

    # Convolution - RELU 3rd Block, using 128 feature maps, size 5x5
    model.add(Conv2D(128, kernel_size=(5, 5), activation='relu'))
    model.add(BatchNormalization())

    # I tried using same kernel size with different number of feature maps but this is the combination with
    #which I got the best results in limited amount of time

    # Max Pooling layer with pooling matrix of size 2x2
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    # Flattening the vector to input it into the model
    model.add(Flatten())

# Classification layers

    # First hidden layer has 128 Neurons and it also represents the dimension of its output space
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    # Second hidden layer has 64 Neurons and it is the dimension of its output space.
    # Most of the problems are solved using 1 to 2 hidden layers,there are very few situations
    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))

    # This is the final layer in the model, it has num_classes number of neurons which is 10 in our case.
    # Activation function used here is softmax as here it will classify the input into on of the 10 output classes
    model.add(Dense(num_classes, activation='softmax'))

    # Loss function used is categorical_crossentropy as it is a multiclass-classification problem
    model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(), metrics=['accuracy'])

    return model
```

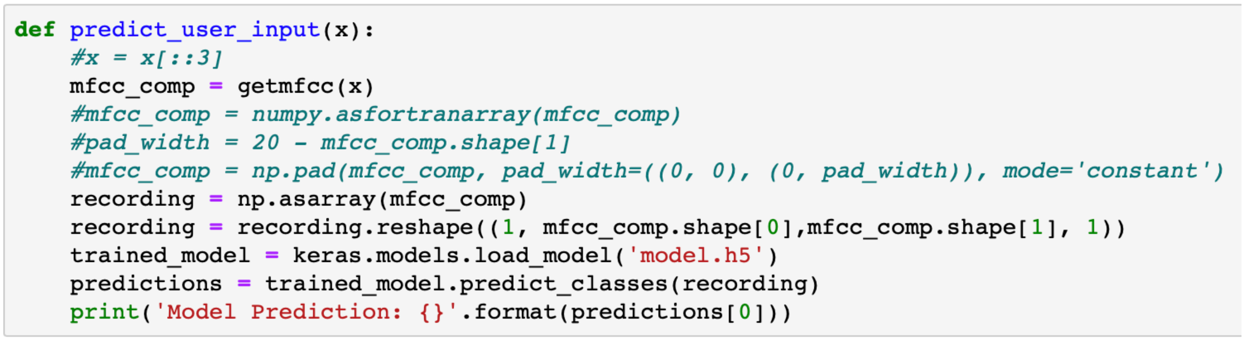
Below shows a plot of our training and validation loss. Note how it decreases.



REAL-TIME RESULT:

Predicting Real-Time Audio:

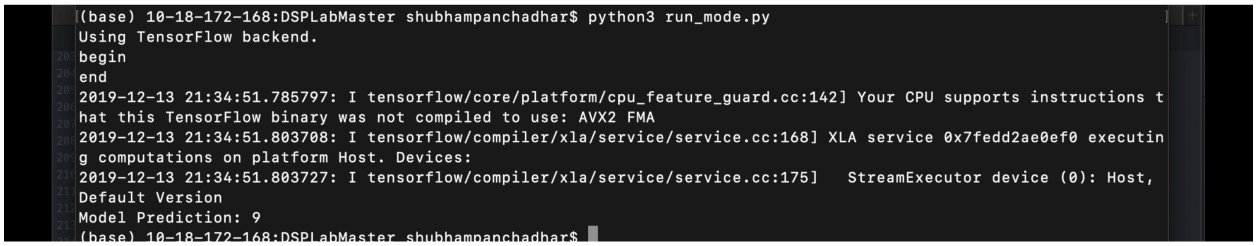
Below is a function which takes in user-recorded voice as input, does necessary feature extraction by finding the MFCC, reshapes the MFC array and is then fed to our saved CNN model. A prediction is made on the audio file recorded and the function returns the output i.e. the spoken digit.



```python
def predict_user_input(x):
    #x = x[::3]
    mfcc_comp = getmfcc(x)
    #mfcc_comp = numpy.asfortranarray(mfcc_comp)
    #pad_width = 20 - mfcc_comp.shape[1]
    #mfcc_comp = np.pad(mfcc_comp, pad_width=((0, 0), (0, pad_width)), mode='constant')
    recording = np.asarray(mfcc_comp)
    recording = recording.reshape((1, mfcc_comp.shape[0],mfcc_comp.shape[1], 1))
    trained_model = keras.models.load_model('model.h5')
    predictions = trained_model.predict_classes(recording)
    print('Model Prediction: {}'.format(predictions[0]))
```

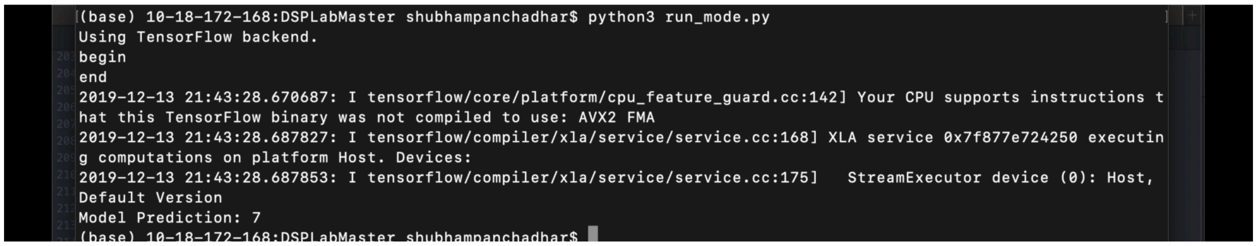
The recording below contains my voice where I say the digit 9

When this recording is fed to our CNN classifier, it predicts the correct digit spoken. Below is our terminal output for the same.



```
(base) 10-18-172-168:DSPLabMaster shubhampanchadhar$ python3 run_mode.py
Using TensorFlow backend.
begin
end
2019-12-13 21:34:51.785797: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions t
hat this TensorFlow binary was not compiled to use: AVX2 FMA
2019-12-13 21:34:51.803708: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7fedd2ae0ef0 executin
g computations on platform Host. Devices:
2019-12-13 21:34:51.803727: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host,
Default Version
Model Prediction: 9
(base) 10-18-172-168:DSPLabMaster shubhampanchadhar$
```

Another example where there is another real-time voice recording of the digit 7 and our classifier has correctly predicted the digit.



```
(base) 10-18-172-168:DSPLabMaster shubhampanchadhar$ python3 run_mode.py
Using TensorFlow backend.
begin
end
2019-12-13 21:43:28.670687: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions t
hat this TensorFlow binary was not compiled to use: AVX2 FMA
2019-12-13 21:43:28.687827: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7f877e724250 executin
g computations on platform Host. Devices:
2019-12-13 21:43:28.687853: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): Host,
Default Version
Model Prediction: 7
(base) 10-18-172-168:DSPLabMaster shubhampanchadhar$
```

CONCLUSION AND FUTURE WORK

This project outlines a method to recognize spoken digits in real-time. Our CNN classifier model can also be extended to words if they can be limited to a specific set of chosen words.

One of the problems we faced was fewer data samples because of which the accuracy of predicting certain digits was poor in real time. If our datasets had more recordings we could achieve a higher accuracy with the appropriate set of weights.

REFERENCES

"Speech Recognition." ***Wikipedia*****. Wikimedia Foundation, 30 Jan. 2013. Web. 12 Feb. 2013.**

Santosh V. Chapaneri. ***Spoken Digits Recognition using Weighted MFCC and Improved Features for Dynamic Time Warping.*** **International Journal of Computer Applications (0975 – 8887) Volume 40– No.3, February 2012**

https://www.youtube.com/watch?v=Z7YM-HAz-IY&list=PLhA3b2k8R3t2Ng1WW_7MiXeh1pfQJQi_P

http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

http://datagenetics.com/blog/november32012/index.html

Machine Learning Lectures

http://www.stanford.edu/class/cs229/

http://www.youtube.com/watch?v=UzxYlbK2c7E