Information Security Analysis and Audit

Project Review 2

Topic: DATA SECURE SMART HOME AUTOMATION SYSTEM

My task: Language Brain using deep learning

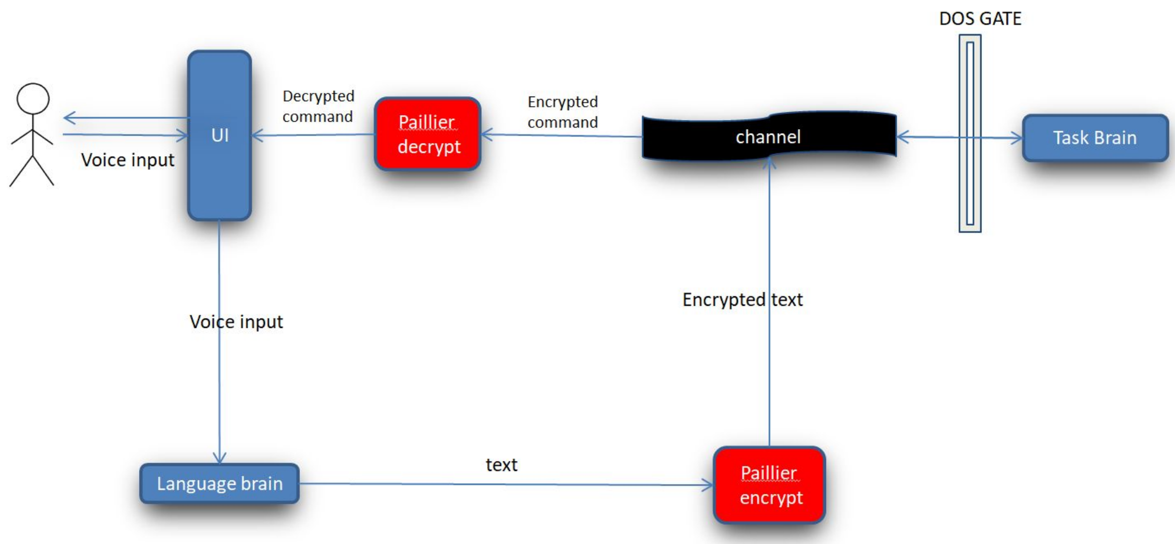
Name: Rajat gupta
Registration no: 18BIT0006
Slot: G1

Design of the system and Description

Ans:

DESIGN:

Entire model :



a) Langauge brain:

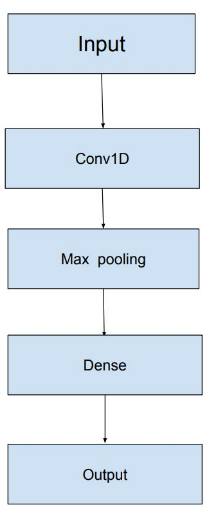


Figure 2: Model

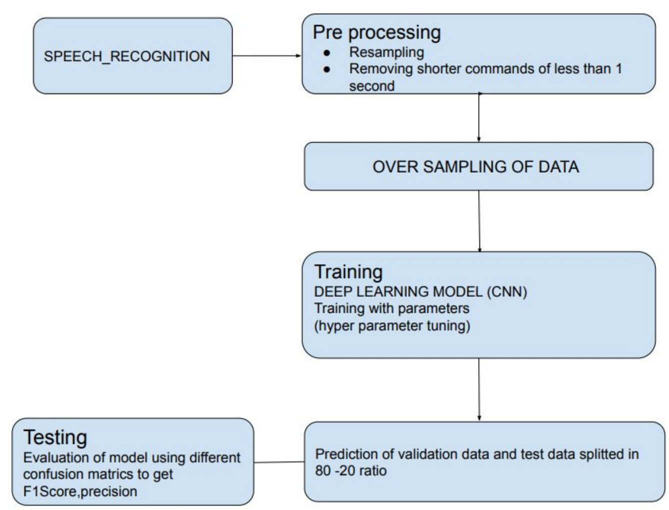


Figure 3: Architectu

b) Encryption

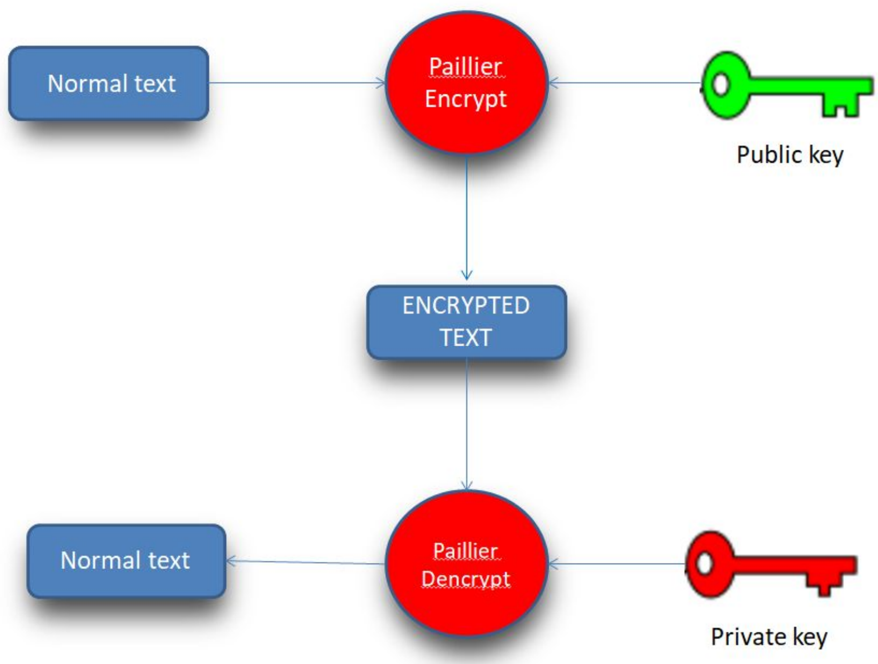


Figure 4: Architecture

c) Task Brain:

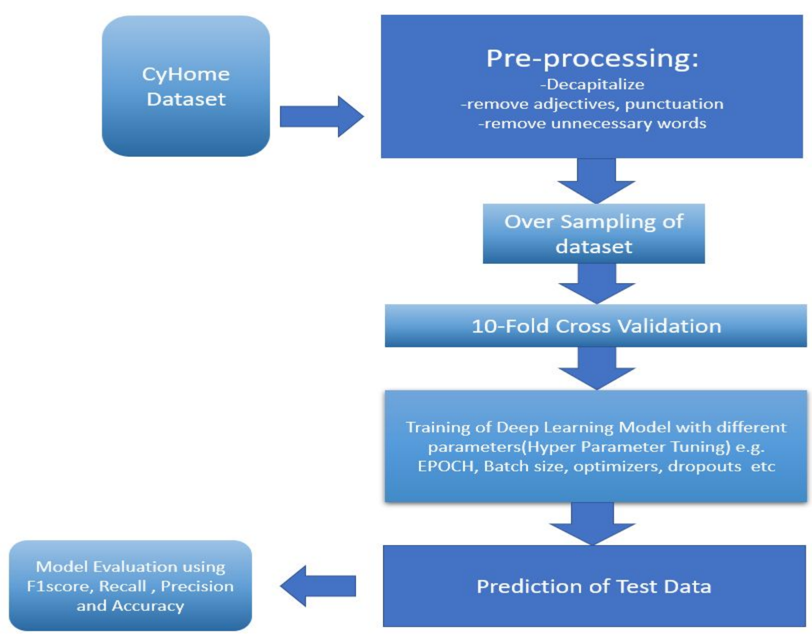


Figure 5: Architecture

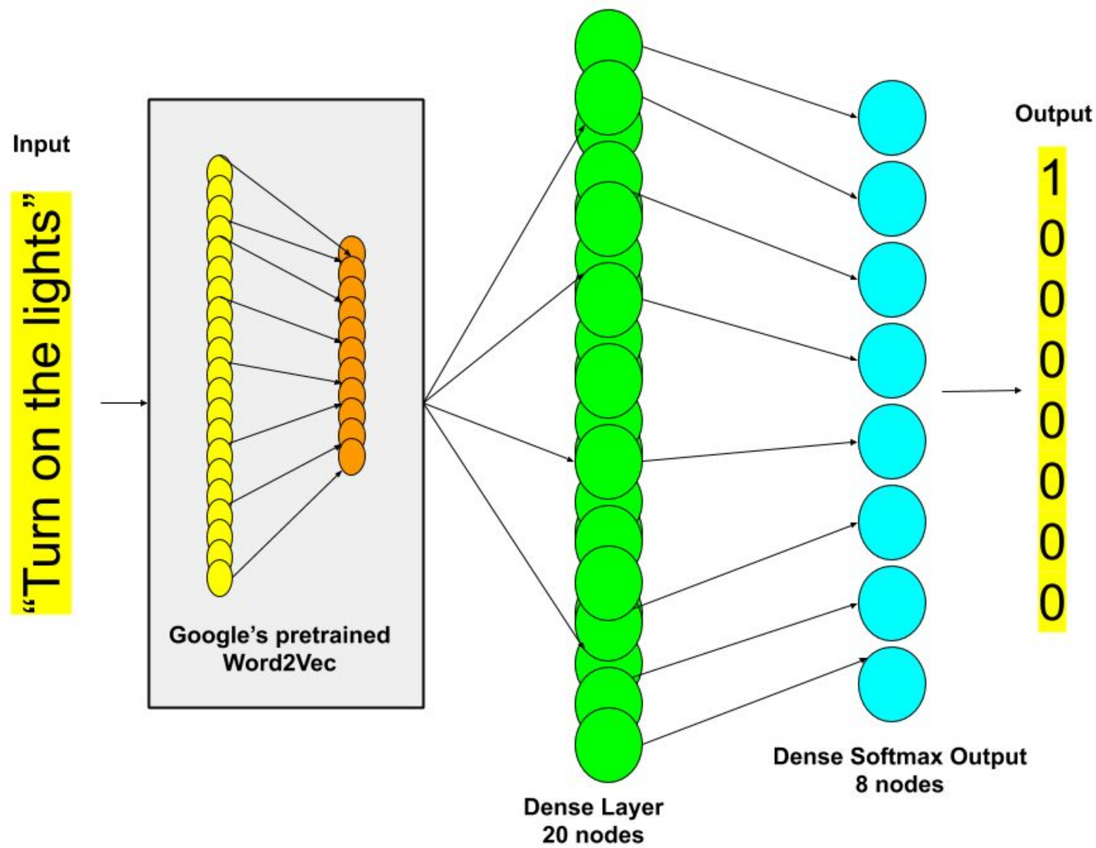


Figure 6: Deep learning model

d) DOS :

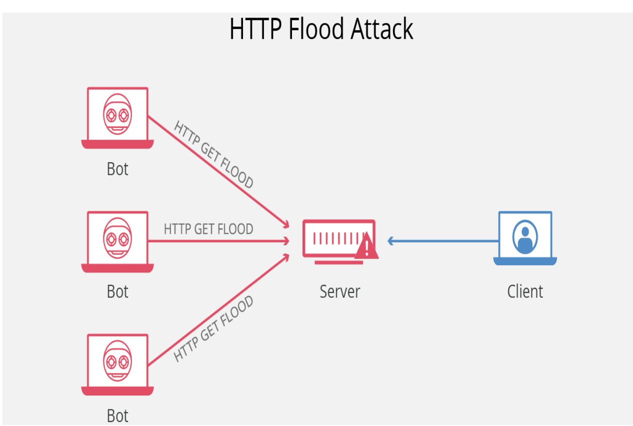


Figure 7: Http Flood Attack

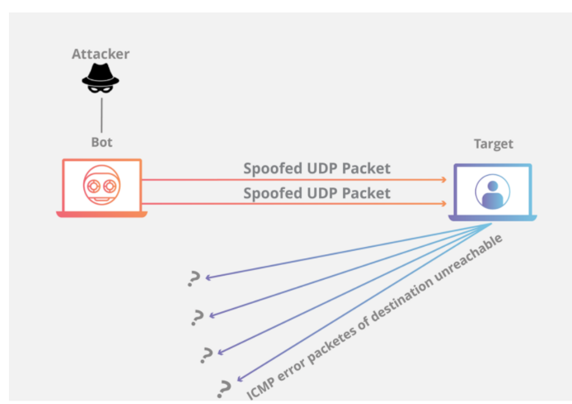


Figure 8: UDP Flood attack

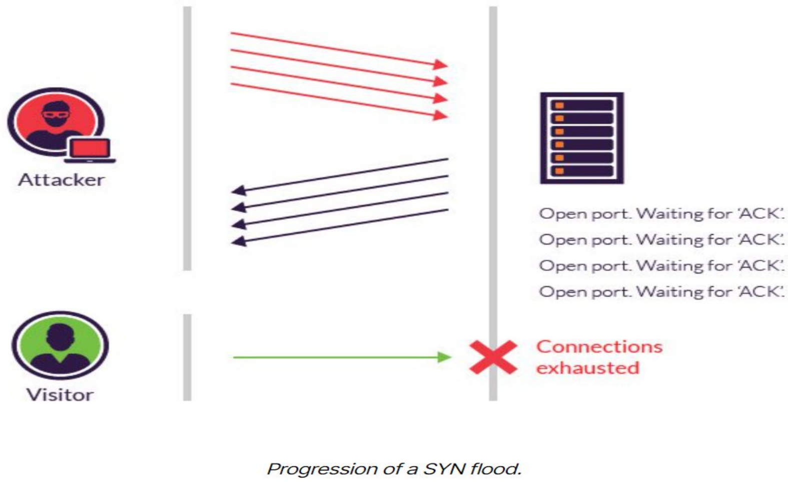


Figure 9: SYN flood attack

DESCRIPTION:

1. Language Brain:

- The user feeds in his voice in the model.
- Preprocessing is done on the voice input in two steps :
    1. Resampling
    2. Removing shorter commands of less than 1 second
- Oversampling of data is done to get 70 percent balanced data
- Training of CNN model is done with different hyper tuning parameters and evaluated many times with different parameters to get better accuracy.
- After that prediction is done with many different parameter and best model is selected
- Last the model is tested and generation of confusion matrics are done to evaluate the model.
- The converted text from the voice input is obtained as the output.

2.Encryption and Task brain:

- The text is then then preprocessed at the client side to convert it to a vector of length 186 by:
    - Decapitalizing each sentence
    - Removing Adjectives and punctuations
    - Removing unnecessary words

- This vector of integers is then encrypted using public key and sent to the server where the task brain resides.
- After this the pre-processed encrypted dataset is oversampled to to get minimum of 70% balanced dataset.
- Then 10-Fold cross validation is used to get max accuracy with a fixed parameter.
- Also parameters are changed (Several times)and 10 fold cross validation is done again.
- At last the best model among all the models is taken and testing is done in that model.
- Finally, to evaluate the model F1-score, Precision, Recall and Accuracy along with confusion matrix is used.
- The trained model processes the input encrypted vector and returns an array of length 8.
- This new array is sent back to the client side.
- It is decrypted using private key and then the values are compared with a predefined dictionary of commands.
- The command with the highest probability is executed in the client side.

3) DOS attack prevention:

- We will be preventing some of the DOS attack on the server where our task brain is located so that the channel between the client and the server is secure.

Import the libraries

First, import all the necessary libraries into our notebook. LibROSA and SciPy are the Python libraries used for processing audio signals.

```
In [1]: import os
        import librosa
        import IPython.display as ipd
        import matplotlib.pyplot as plt
        import numpy as np
        from scipy.io import wavfile
        import warnings

        warnings.filterwarnings("ignore")
```

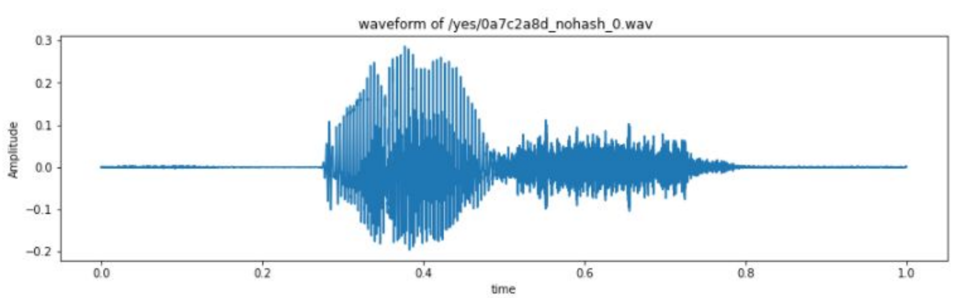
Data Exploration and Visualization

Data Exploration and Visualization helps us to understand the data as well as pre-processing steps in a better way.

Visualization of Audio signal in time series domain

Now, we'll visualize the audio signal in the time series domain:

```
In [3]: train_audio_path = 'D:/train/audio/'
        samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = 16000)
        fig = plt.figure(figsize=(14, 8))
        ax1 = fig.add_subplot(211)
        ax1.set_title(' waveform of ' + '/yes/0a7c2a8d_nohash_0.wav')
        ax1.set_xlabel('time')
        ax1.set_ylabel('Amplitude')
        ax1.plot(np.linspace(0, sample_rate/len(samples), sample_rate), samples)

Out[3]: [<matplotlib.lines.Line2D at 0x1d4f6970>]
```

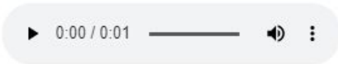


Sampling rate

Let us now look at the sampling rate of the audio signals:

```
In [4]: ipd.Audio(samples, rate=sample_rate)
```

Out[4]:




```
In [5]: print(sample_rate)
```

```
16000
```

Resampling

From the above, we can understand that the sampling rate of the signal is 16,000 Hz. Let us re-sample it to 8000 Hz since most of the speech-related frequencies are present at 8000 Hz:

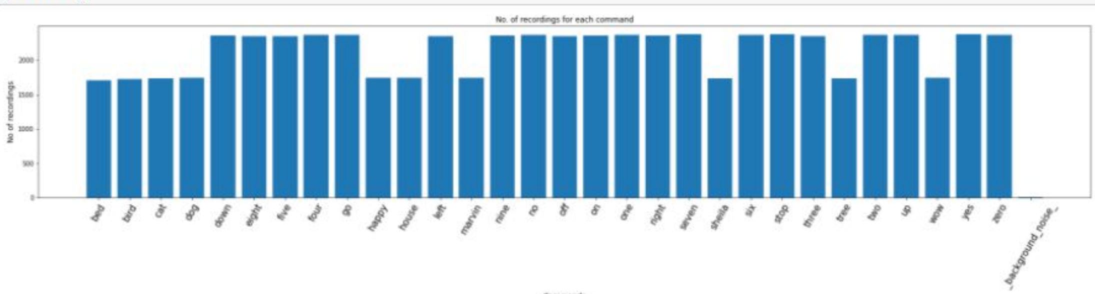
```
In [6]: samples = librosa.resample(samples, sample_rate, 8000)
        ipd.Audio(samples, rate=8000)
```

Out[6]:

Now,calculated the number of recordings for each voice command:

```
In [7]: labels=os.listdir(train_audio_path)
```

```
In [8]: #find count of each label and plot bar graph
        no_of_recordings=[]
        for label in labels:
            waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
            no_of_recordings.append(len(waves))

        #plot
        plt.figure(figsize=(30,5))
        index = np.arange(len(labels))
        plt.bar(index, no_of_recordings)
        plt.xlabel('Commands', fontsize=12)
        plt.ylabel('No of recordings', fontsize=12)
        plt.xticks(index, labels, fontsize=15, rotation=60)
        plt.title('No. of recordings for each command')
        plt.show()
```



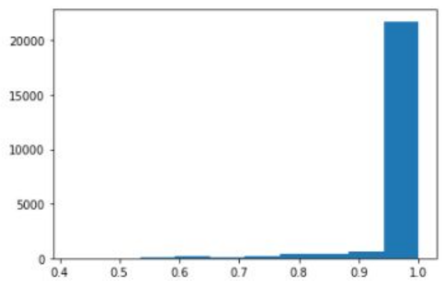
Duration of recordings

What’s next? A look at the distribution of the duration of recordings:

In [9]:
```
labels=["yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go"]
```

In [10]:
```
duration_of_recordings=[]
for label in labels:
    waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
    for wav in waves:
        sample_rate, samples = wavfile.read(train_audio_path + '/' + label + '/' + wav)
        duration_of_recordings.append(float(len(samples)/sample_rate))

plt.hist(np.array(duration_of_recordings))
```

Out[10]:
```
(array([1.5000e+01, 3.0000e+01, 4.4000e+01, 1.3800e+02, 1.3600e+02,
        1.7900e+02, 3.6600e+02, 4.3400e+02, 5.9300e+02, 2.1747e+04]),
 array([0.418 , 0.4762, 0.5344, 0.5926, 0.6508, 0.709 , 0.7672, 0.8254,
        0.8836, 0.9418, 1.    ]),
 <BarContainer object of 10 artists>)
```



Preprocessing the audio waves

In the data exploration part earlier, we have seen that the duration of a few recordings is less than 1 second and the sampling rate is too high.Here are the two steps:

- Resampling
- Removing shorter commands of less than 1 second

In [11]:
```
all_wave = []
all_label = []
for label in labels:
    print(label)
    waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
    for wav in waves:
        samples, sample_rate = librosa.load(train_audio_path + '/' + label + '/' + wav, sr = 16000)
        samples = librosa.resample(samples, sample_rate, 8000)
        if(len(samples)== 8000) :
            all_wave.append(samples)
            all_label.append(label)
```

```
yes
no
up
down
left
right
on
off
stop
go
```

Convert the output labels to integer encoded:

```
In [12]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         y=le.fit_transform(all_label)
         classes= list(le.classes_)
```

Now, convert the integer encoded labels to a one-hot vector since it is a **multi-classification problem**:

```
In [13]: from keras.utils import np_utils
         y=np_utils.to_categorical(y, num_classes=len(labels))

         Using TensorFlow backend.
```

Reshape the 2D array to 3D since the input to the conv1d must be a 3D array:

```
In [14]: all_wave = np.array(all_wave).reshape(-1,8000,1)
```

Split into train and validation set

Next, we will train the model on 80% of the data and validate on the remaining 20%:

```
In [15]: from sklearn.model_selection import train_test_split
         x_tr, x_val, y_tr, y_val = train_test_split(np.array(all_wave),np.array(y),stratify=y,test_size = 0.2,random_state=777,shuffle=Tr
```

Model Architecture for this problem

We will build the speech-to-text model using conv1d. Conv1d is a convolutional neural network which performs the convolution along only one dimension.

```
In [16]: from keras.layers import Dense, Dropout, Flatten, Conv1D, Input, MaxPooling1D
         from keras.models import Model
         from keras.callbacks import EarlyStopping, ModelCheckpoint
         from keras import backend as K
         K.clear_session()

         inputs = Input(shape=(8000,1))

         #First Conv1D Layer
         conv = Conv1D(8,13, padding='valid', activation='relu', strides=1)(inputs)
         conv = MaxPooling1D(3)(conv)
         conv = Dropout(0.3)(conv)

         #Second Conv1D Layer
         conv = Conv1D(16, 11, padding='valid', activation='relu', strides=1)(conv)
         conv = MaxPooling1D(3)(conv)
         conv = Dropout(0.3)(conv)

         #Third Conv1D Layer
         conv = Conv1D(32, 9, padding='valid', activation='relu', strides=1)(conv)
         conv = MaxPooling1D(3)(conv)
         conv = Dropout(0.3)(conv)

         #Fourth Conv1D Layer
         conv = Conv1D(64, 7, padding='valid', activation='relu', strides=1)(conv)
         conv = MaxPooling1D(3)(conv)
         conv = Dropout(0.3)(conv)

         #Flatten Layer
         conv = Flatten()(conv)

         #Dense Layer 1
         conv = Dense(256, activation='relu')(conv)
         conv = Dropout(0.3)(conv)

         #Dense Layer 2
         conv = Dense(128, activation='relu')(conv)
         conv = Dropout(0.3)(conv)

         outputs = Dense(len(labels), activation='softmax')(conv)

         model = Model(inputs, outputs)
         model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 8000, 1)           0
_________________________________________________________________
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 8000, 1)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 7988, 8)           112
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 2662, 8)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2662, 8)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2652, 16)          1424
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 884, 16)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 884, 16)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 876, 32)           4640
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 292, 32)           0
_________________________________________________________________
dropout_3 (Dropout)          (None, 292, 32)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 286, 64)           14400
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 95, 64)            0
_________________________________________________________________
dropout_4 (Dropout)          (None, 95, 64)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6080)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1556736
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,611,498
Trainable params: 1,611,498
Non-trainable params: 0
_________________________________________________________________
```

Define the loss function to be categorical cross-entropy since it is a multi-classification problem:

```
In [17]: model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Early stopping and model checkpoints are the callbacks to stop training the neural network at the right time and to save the best model after every epoch:

```
In [18]: es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, min_delta=0.0001)
         mc = ModelCheckpoint('best_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

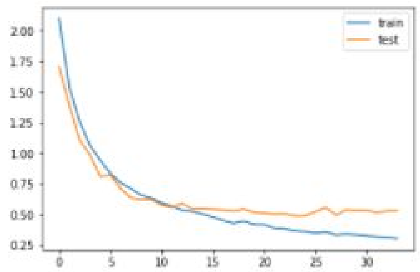
train the model on a batch size of 32 and evaluate the performance on the holdout set:

```
In [19]: history=model.fit(x_tr, y_tr ,epochs=100, callbacks=[es,mc], batch_size=32, validation_data=(x_val,y_val))

Train on 17049 samples, validate on 4263 samples
Epoch 1/100
17049/17049 [==============================] - 13s 752us/step - loss: 2.0998 - acc: 0.2008 - val_loss: 1.7085 - val_acc: 0.3
800

Epoch 00001: val_acc improved from -inf to 0.38001, saving model to best_model.hdf5
Epoch 2/100
17049/17049 [==============================] - 6s 335us/step - loss: 1.5393 - acc: 0.4221 - val_loss: 1.3905 - val_acc: 0.50
79

Epoch 00002: val_acc improved from 0.38001 to 0.50786, saving model to best_model.hdf5
Epoch 3/100
17049/17049 [==============================] - 6s 333us/step - loss: 1.2589 - acc: 0.5556 - val_loss: 1.1108 - val_acc: 0.63
71

Epoch 00003: val_acc improved from 0.50786 to 0.63711, saving model to best_model.hdf5
Epoch 4/100
17049/17049 [==============================] - 6s 335us/step - loss: 1.0668 - acc: 0.6292 - val_loss: 0.9898 - val_acc: 0.67
14
```

Diagnostic plot

I'm going to lean on visualization again to understand the performance of the model over a period of time:

In [20]:
```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



Loading the best model

In [21]:
```
from keras.models import load_model
model=load_model('best_model.hdf5')
```

Make predictions on the validation data:

In [23]:
```
import random
index=random.randint(0,len(x_val)-1)
samples=x_val[index].ravel()
print("Audio:",classes[np.argmax(y_val[index])])
ipd.Audio(samples, rate=8000)
```

```
Audio: no
```

Out[23]:

In [24]:
```
print("Text:",predict(samples))
```

```
Text: no
```