

Intelligent Hearing

Introduction:

Listener is a device which captures ambient sound and categorizes the sound and perform the appropriate action assigned to it. The purpose of this device to implement it as a security measure in the streets so that in case of any mishappening the device would act as per the scenario.

By using advanced machine learning techniques enabling the device to be capable of recognizing sound similar to the complexity of that the human ear. The final result of the project would to manufacture a device which is capable of comprehending sounds and probably having the super hearing quality and can be used for the benefit of human race.

This device has cross-overs in the following fields:

1. Biology - Functioning of Human Ear
2. Machine learning
3. Data mining
4. Digital Signal Processing
5. Embedded Systems
6. Data Analysis
7. Algorithm Analysis

Specifics:

1. Python and its libraries for Machine learning and Data mining
2. CPython / Embedded Python for Configuration of Raspberry Pi / Gumstix / MSP430
3. Hadoop - Spark for Data-Mining and Data analysis (Python / Scala implementation)
4. Cocktail-Problem

Map

1. Functioning Of human Ear - Detailed Study
2. Digital Signal Processing
3. Machine Learning
4. Data Mining
5. Embedded Systems
6. Algorithm Analysis

Implementation Details:

The project is in its partial stage of implementation considering the scale of the project and its domain specific understanding requires time to gain expertise over it and manufacture a device or built a prototype which fulfills the desired level of clearance for calling it to be the completion of the project.

Implementation steps include:

1. Information gathering
2. Understanding the scope of the thesis-project
3. Classifying and identifying the project problem
4. Detail study of the domains required.
5. Defining the scope of the domains required to be studied.
6. Understanding the software requirements
7. Understanding the hardware requirements
8. Identifying the various phases required for project implementation
9. Preparing schedule of the project implementation phase
10. Setup the environment and Start the project phase by phase.

1.Information Gathering: This step includes the process of gathering information over the subject of matter. Collect all the information which will be required for completion of the project. This step is crucial in the sense that if it fails to gather the various information required or misses out a particular section or fail to identify the smaller or intertwined or complex connective domains it will be hard for in the future to access the necessary tools required for the successful implementation of the components or the phase in which the hardware device can be made in much simpler and efficient way but actually ended up implementing in much more complex and time consuming way. So lack of information only makes the gaps in experimental procedures leading to lack of optimization.

2.Understanding the scope of the thesis-project: This step includes the process of understanding the scope of the project. This step might seem not necessary but in fact it is the most wanted and most important step in the implementation procedure. By determining the scope of the project it will only help to reduce time by not wasting time in implementing the sections or the sectors which are not necessary for successful

execution of the project. This step not only help in making the project completion in time but also defines the concrete steps for the completion of it.

3.Classifying and identifying the project problem: It is necessary to understand the problem that you are solving. It is necessary to identify and exactly know what is the problem that is to be solved. Without the clear understanding of the problem is not possible to get a device exactly get to work we want. Classification of the problem is absolutely necessary for understanding the working and defining the steps necessary for the execution of the project.

4.Detail study of the domains required: The detail study of the domains only help in making the execution better. As it is correctly said in the second step of the implementation that understanding of the scope is necessary as it helps with efficiency and better optimization of time not directly but indirectly it speeds up the process. The various domains of the projects are : Machine learning, Data mining, Digital Signal Processing, Embedded Systems, Data Analysis, Algorithm Analysis. Most important of them all is the biological functioning of the ‘Human Ear’.

5.Defining the scope of the domains required to be studied: In the previous two steps where the scope of the project was defined and the domains of the project was defined. It is not enough to just define the domains of the project but also the scope of the domains. Every domain is as vast as sea considering the fact that in the modern world research and development keep coming up in the domains that is mentioned. Thus, it is necessary to define the scope of the domains required to be studied.

6.Understanding the software requirements: It is important to understand the software support which provides the successful execution of the project. Software details include:

1. Operating system: Suse Linux/ CentOS 7 server
2. Architecture: 64 bit system
3. Core Programming Language: Python 3.5
4. Environment tool: Anaconda Distribution 4.3.1
5. CPython / Embedded Python for Configuration of Raspberry Pi / Gumstix/ MSP430

7. Understanding the hardware requirements: It is necessary to understand and determine the hardware requirements. Following are the hardware requirements necessary for the project

1. ChipSet: Intel greater than i3 versions
2. RAM: 4gb minimum
3. Disk Space: 500 GB
4. TI MSP 430 motherboard / RaspBerry Pi
5. GPU 8GB Nvidia K40 / Tesla K40 series Iaas

8. Identifying the various phases required for project implementation: It is very important to identify the various phases required for the project implementation.

Phases include:

1. Problem Recognition Phase
2. Information Gathering Phase
3. Domain Specification Phase
4. Scope designing Phase
5. Implementation Phase
6. Modules designing Phase
7. Modules development phase
8. Modules Code Optimization phase
9. Modules Testing Phase
10. Modules Integration phase
11. Modules Post Integration Testing Phase
12. Integrated Modules Deployment phase

9. Preparing schedule of the project implementation phase: Time is an important factor for development of any project. The question is of how fast a project can be developed i.e. most reduced time for deployment of a project. If a project takes too long for successful completion of the project it is not feasible to produce furthermore any kind of financial support towards the project. It is well advised to fund the project based on completion of modules. The scheduled should be produced for the same matter.

10. Setup the environment and Start the project phase by phase: Once the scope of the project is decided and the modularization of the project is completed it is feasible and possible to make a schedule based on the completion of the modules thereby enabling the making it possible for funding the project to. Once the client is in terms of agreement with the cost of planning and the schedule given starting then the phases of the project can be established. The phases of project mentioned in the step 8 is

carried out in this phase. The software requirements that is mentioned in the step 6 is finally brought into action and the environment is set up and the project modules development plan in executed.

Experimental Details and Setup:

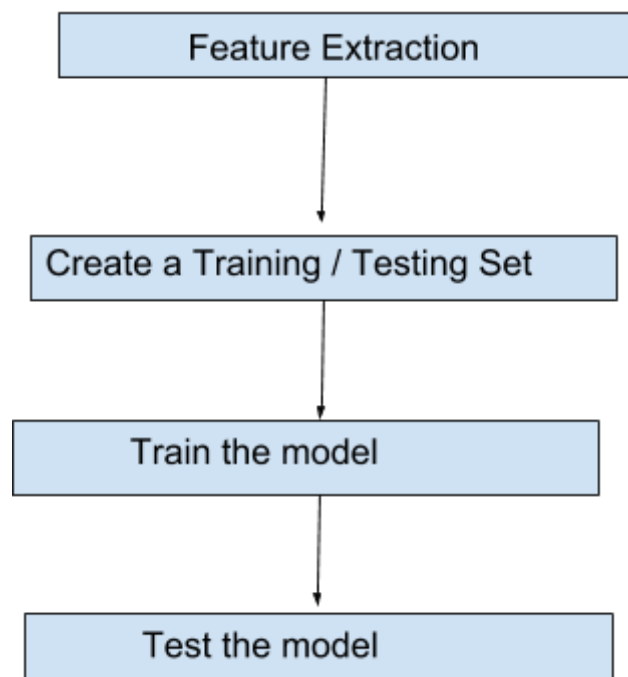
Following are the steps included in the implementation procedure.

1.Environment SetUp:

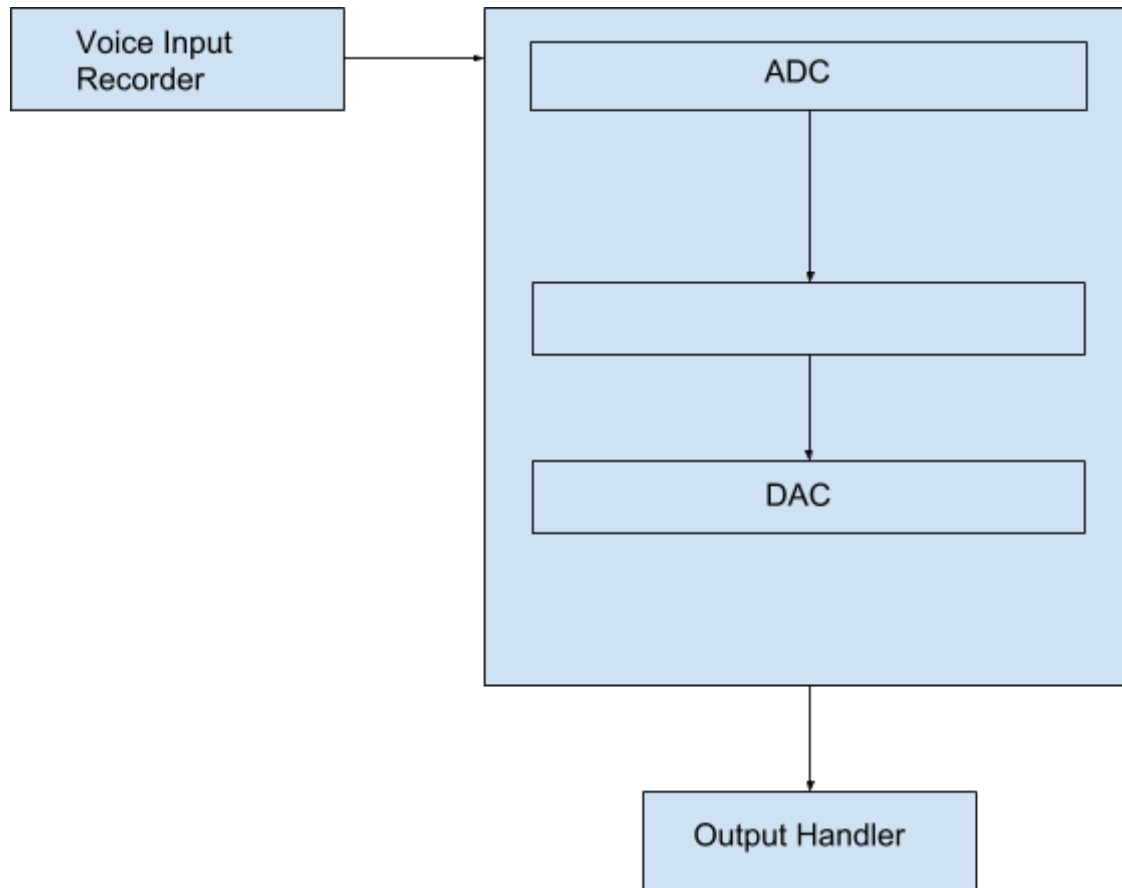
1. Install Linux x86_64 bit
2. Download and install Anaconda Distribution
3. Create a virtual conda environment using the commands:
`conda create -n <env_name>`
4. Install the packages necessary for it either using the spectrogram analysis and neural networks analysis

2. Running Sample Code: A sample code for testing the environment was run and everything was running okay. It is noticed as and when the model is developed the packages required for the same is bound to happen. So the development of the project while introducing and identifying the models of the project the dependencies of the project is changed as and when the the models changed. So it is utmost important to first understand the models included in the project.

3. Understanding the modules: It is important to understand the modules included in the development of the project. Following are the main software components that is supposed to be considered as a module and then develop it based on test-optimize based principle.



4. Understanding the components of the Device: The Device that is desired to be build has the following components:



- 1. Voice input:** This component consists of a mic that captures voice from the surrounding and sends the input stream into the MCU- Main Computation Unit. Microphone is an crucial element because the denoising capacity or the quality of the mic with which it can record a sound from the environment is important as the input data we get are what we use for training the algorithm.
- 2. ADC:** In electronics, an analog-to-digital converter (ADC, A/D, A–D, or A-to-D) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current.
- 3. MAU - Main Analysis Unit:** It consists of the model or the algorithm which is trained upon and the model is used for inferencing. It takes an input from the ADC in the desired format and processes it and take the output and converts it

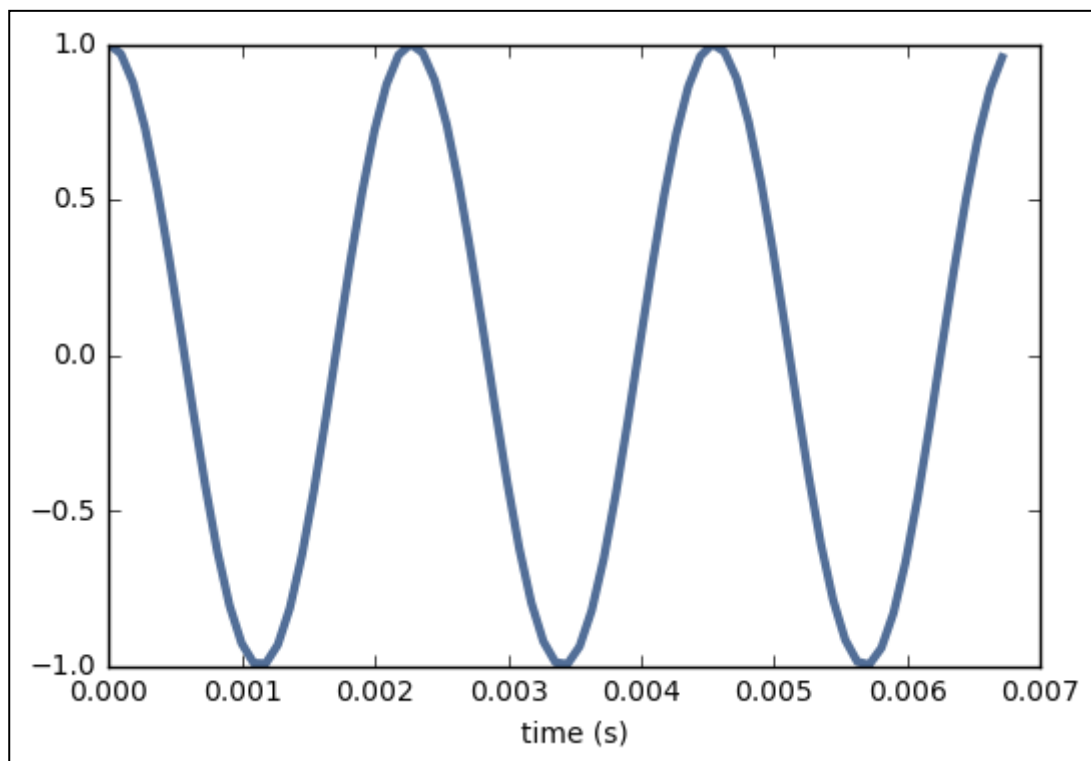
into the desired output. (output may be a voice alarm system / any kind of alarm system)

- 4. Output Handler:** The output handler takes the output the estimated value from the MAU- Main Analysis Unit and puts the output to the respective desired ones. For example, Suppose there is a shrieking sound of a woman in an alley. Then the post processing of the MAU, it is supposed to output the woman's shriek and then based on switch casing, the concerned authority is contacted with the S.O.S message.

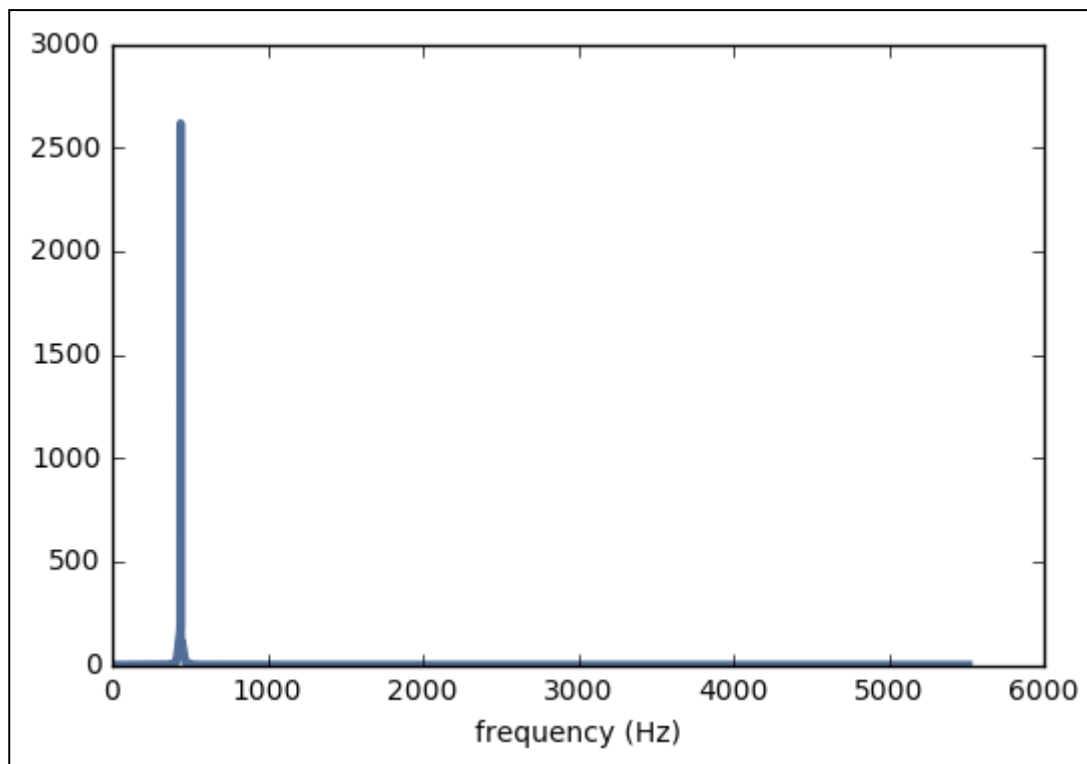
Analysis of a sample Spectrograms:

Following are some of the graphs and its understanding

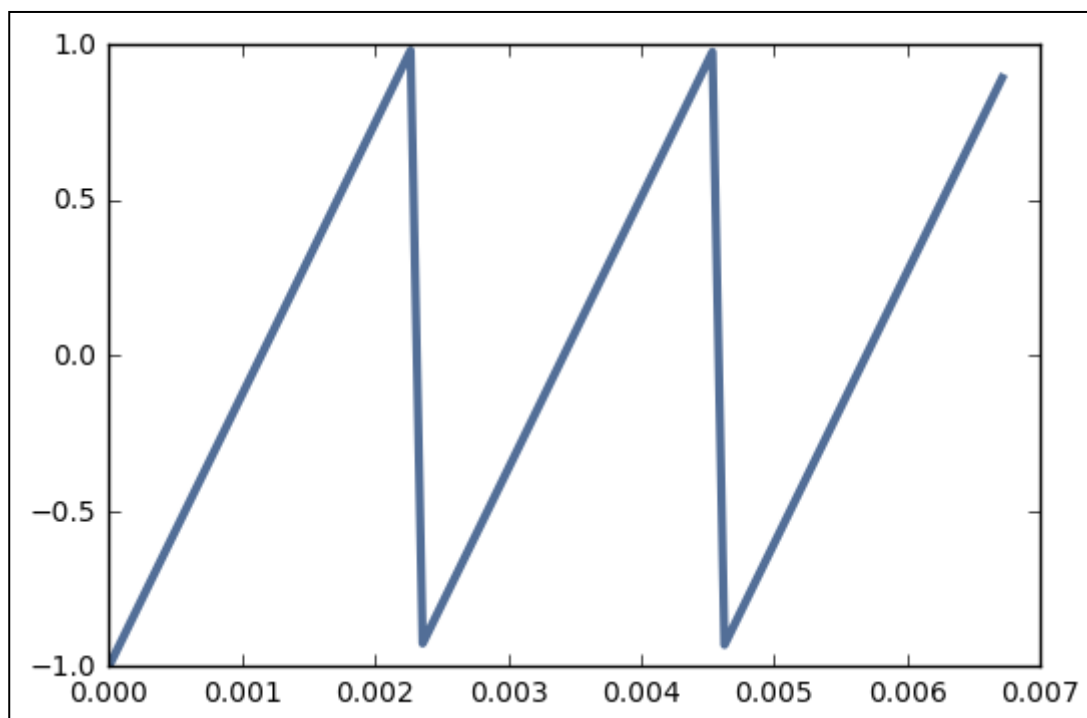
1. Cosine Signal: It is one of the most basic things to be known that of a cosine signal. Following is the example of a cosine signal. As the time moves on the wave moves in the dimension of -1 to +1. It is observed that the wave starts from the positive domain and ends in the positive domain.



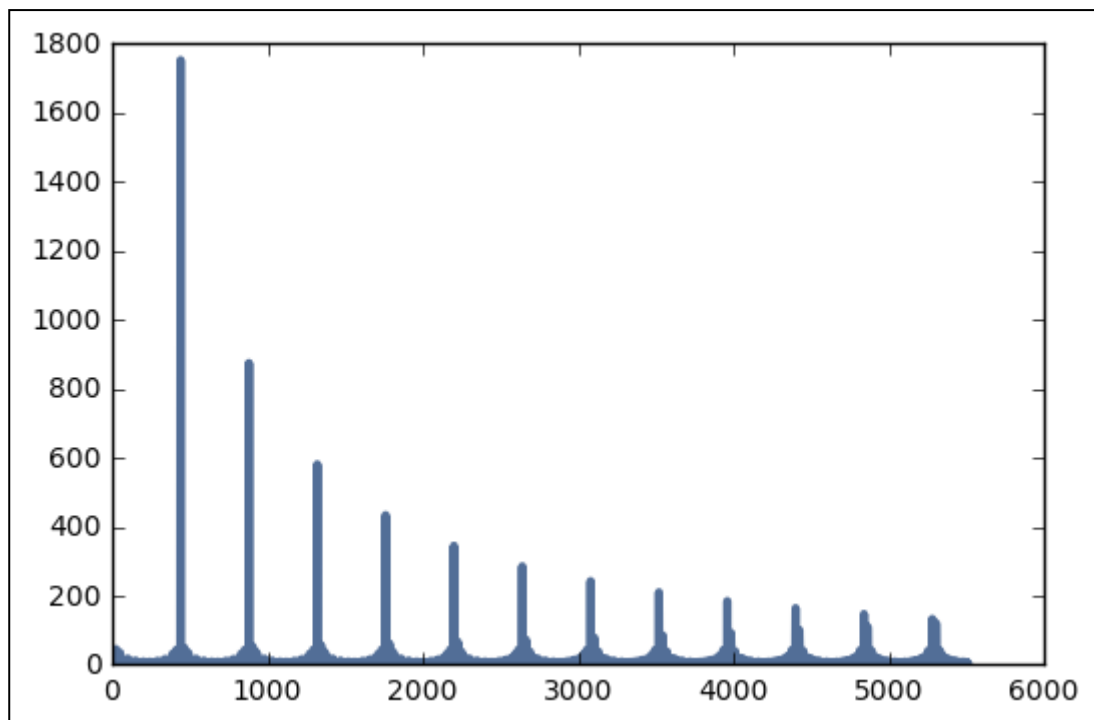
2. Spectrum plot of the cosine signal: It is the simple plotting of a cosine signal using the python library. It shows that the frequency of the waveform in the cosine signal. Since the cosine signal has no harmonics the graph has such a representation.



3. SawTooth Wave plot: The Sawtooth spectrogram as the name suggests is the spectrogram that is formed from the signal of the sawtooth signal. It suggests that the sharp peaks of the graph which resembles the tooth of the saw runs in parallel to the time with the dimension of -1 to +1.

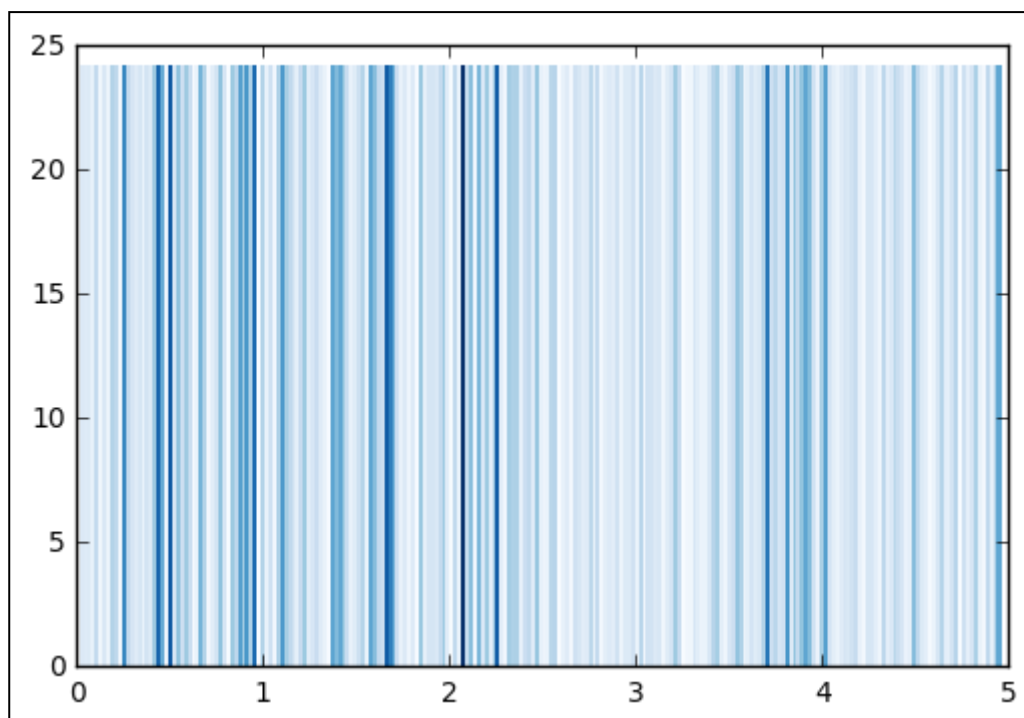


4. Sawtooth Spectrum: As we saw in the above point about the sawtooth signal and its characteristics so is the following spectrum of the sawtooth signal

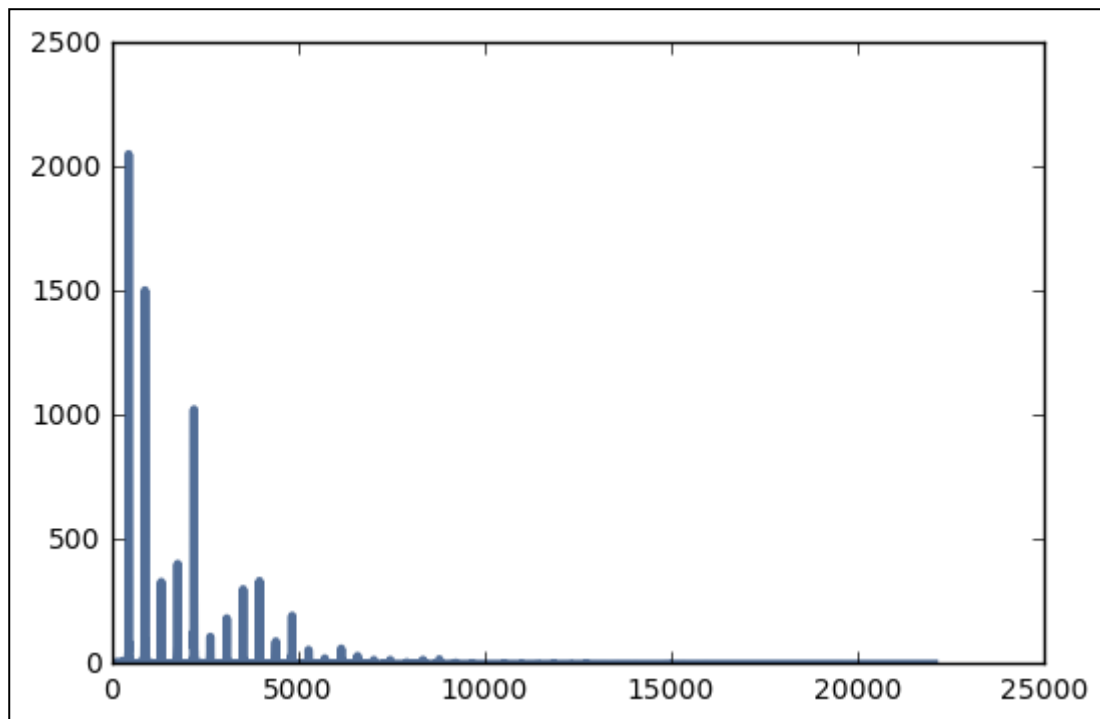


. As you can see the spectrum falls down as in time with the peak value decreasing exponentially to the form.

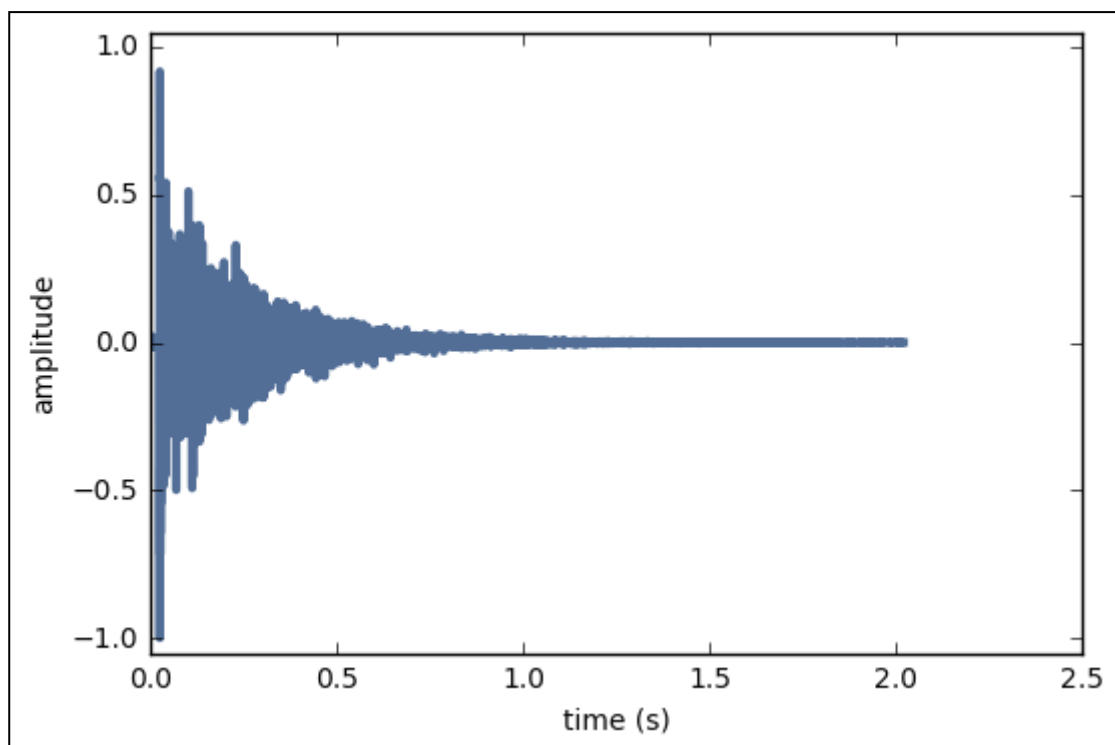
5. Spectrogram of a sample violin: As it is not enough to just get the single signal analysis we are turning up bit a notch to get the understanding of how to handle much complex situations. For mediocre level of analysis let's understand the spectrogram of a person playing violin. Following is the spectrogram of a person playing violin. The dark blue lines suggests the high notes or the points where the sound of the violin had high pitch.



6. Spectrum of a sample violin: Now, understanding the spectrogram is not enough. Learning what is the meaning of the spectrum of the same music or rather in the case of noise it is means to arrive at an inference.

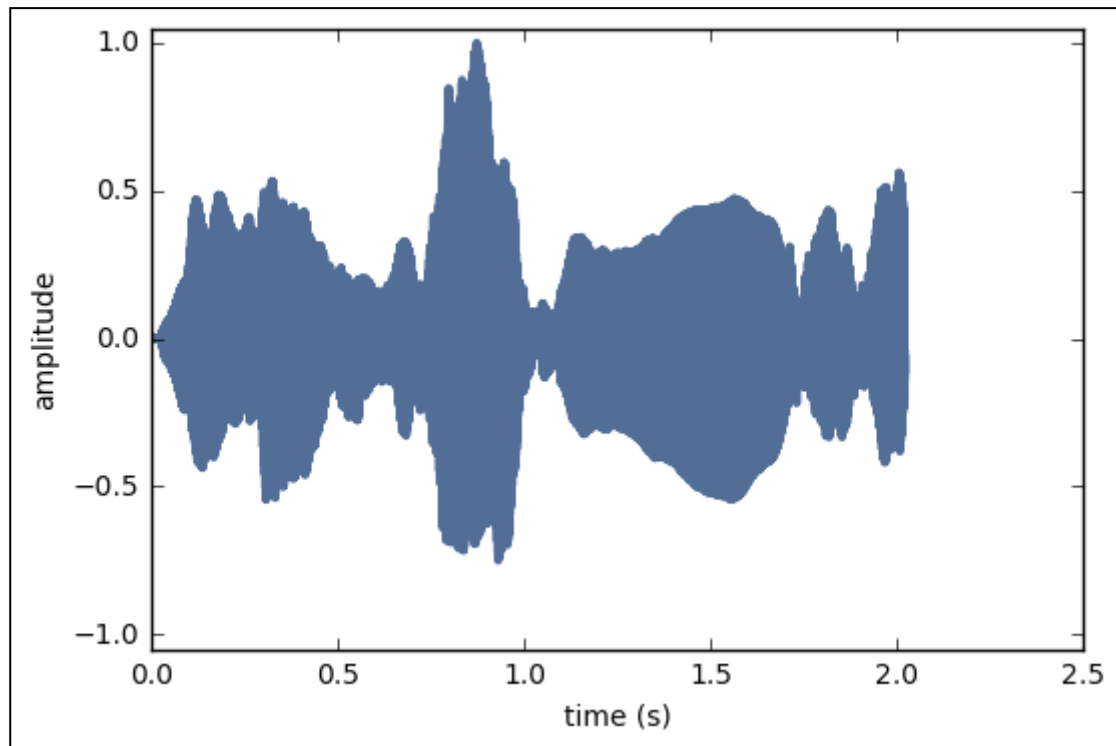


7. Mixed Sound analysis: So far we have only used to figure out the spectrum and the spectrogram of the sound or the audio clips that are singular in nature. Now to increase the complexity of the let's try mixing two sounds. Following is the sound of a gunshot

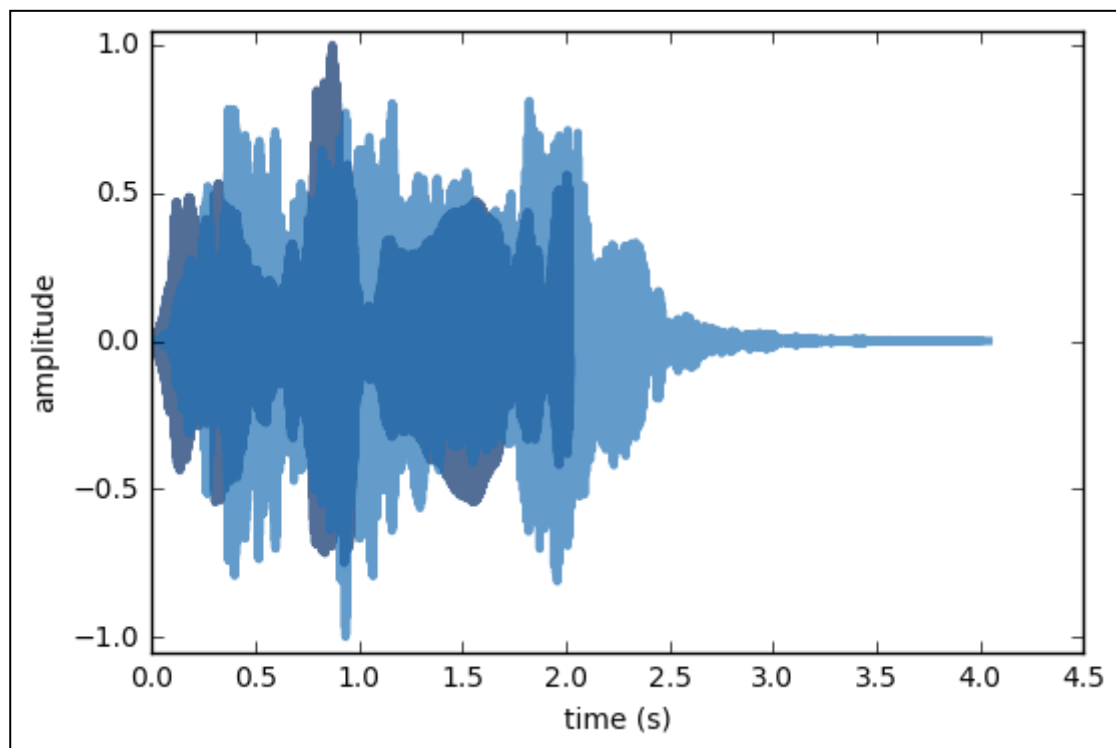


The plot states that with time the amplitude of the sound is reduced to normal

line 0. Following is the plot for the recording of the a person playing violin.

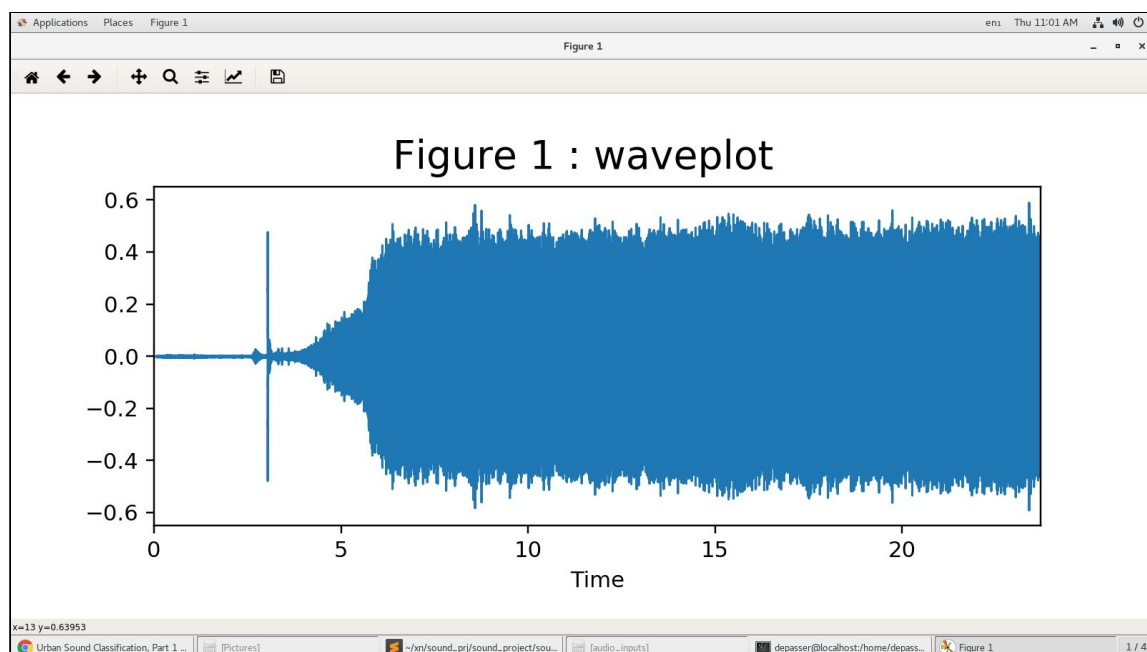


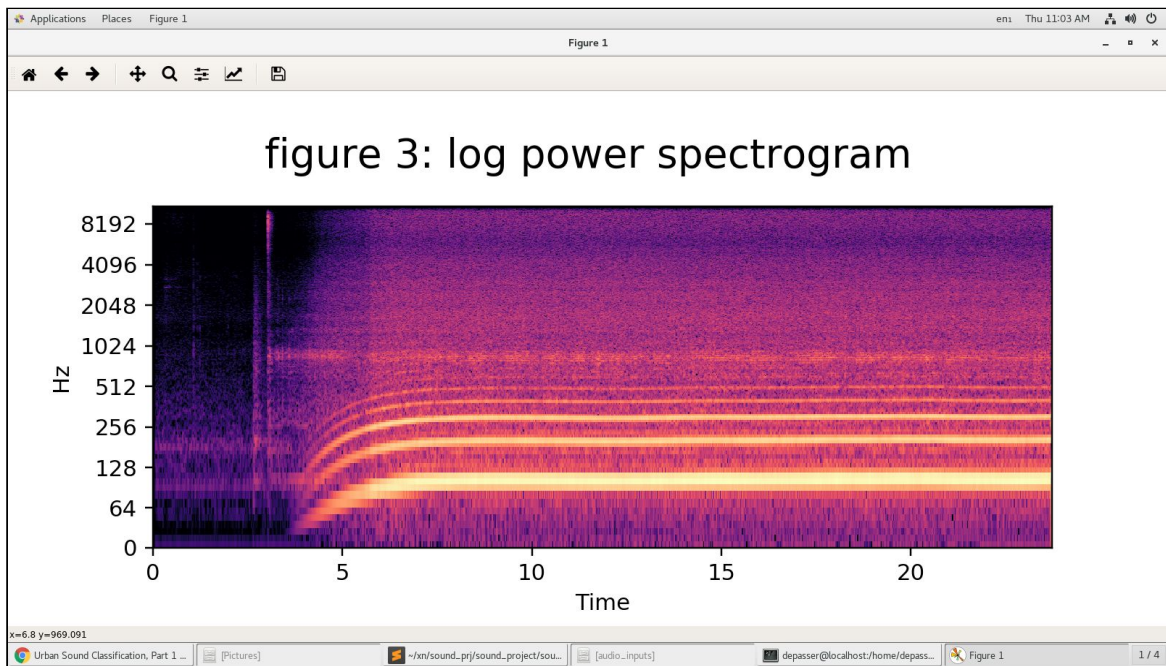
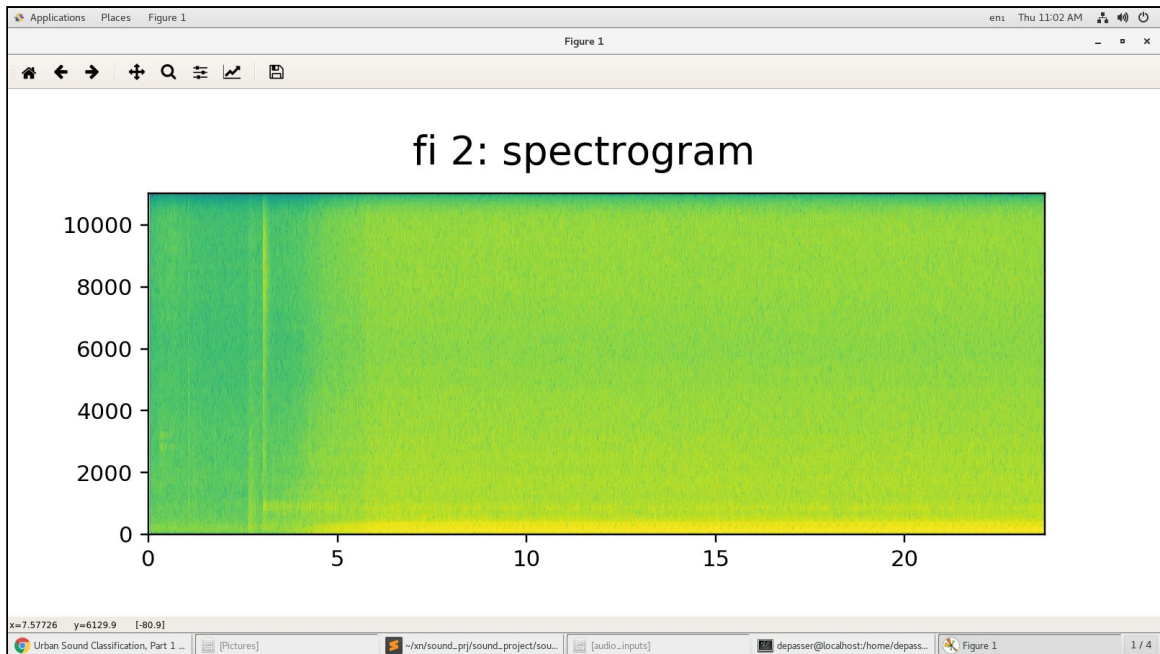
Now, if i mix both the sound. Following is the plot you get.

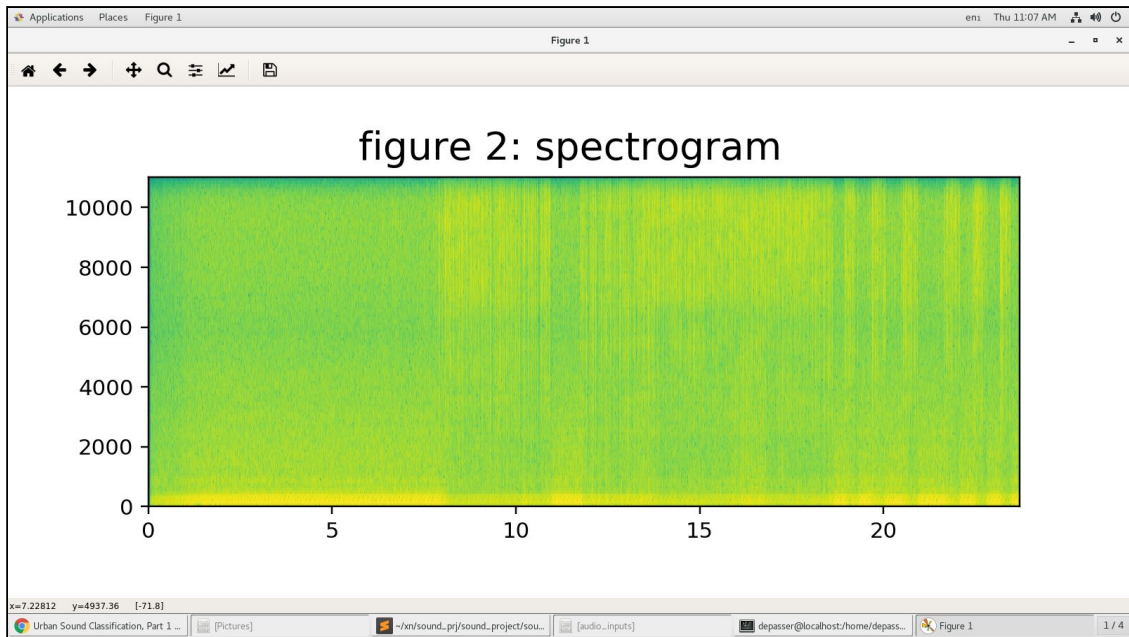


And this is the mixture of two sounds. It sounds like the echo of violin in a hall. Since these are only the experimental measures further we need to go and talk about convolutional neural network. The type of neural network that will be used for creating a training model such that on to this training model the program will be trained and relied upon.

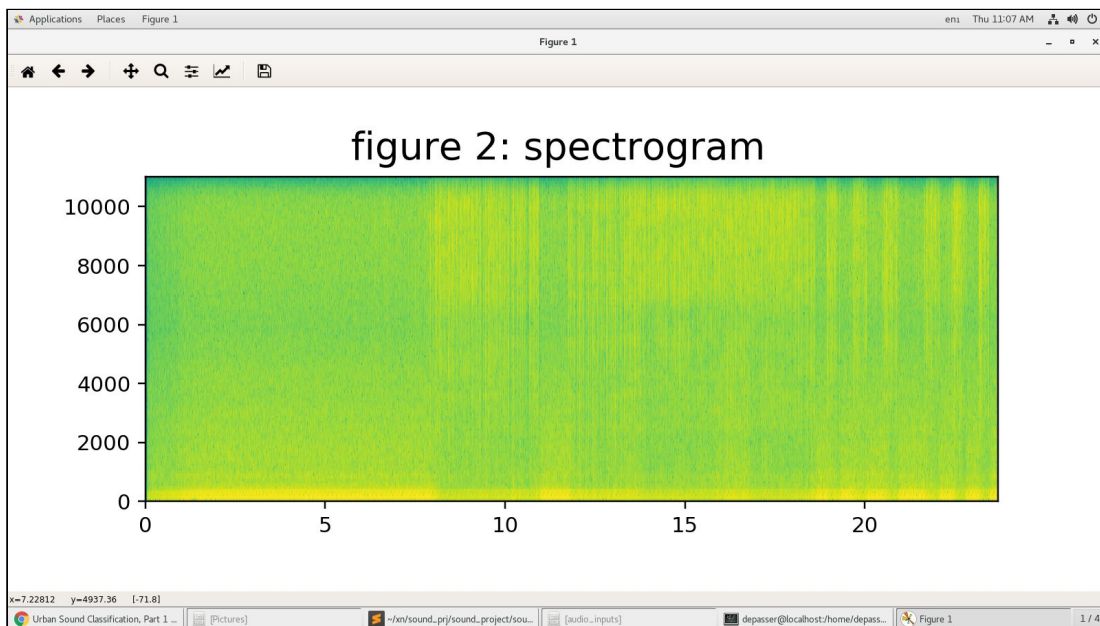
8. Convolutional neural Network: The convolutional neural network is given a sequence of raw input signal, split into frames, and outputs a score for each classes, for each frame. The network architecture is composed of several filter stages, followed by a classification stage. A filter stage involves a convolutional layer, followed by a temporal max-pooling layer and a nonlinearity ($\tanh()$). Our optimal architecture included three filter stages. Processed signals coming out of these stages are fed to a classification stage, which in our case is a multi-layer perceptron, with one hidden layer. It outputs the conditional probabilities $p(i|x)$ for each class i , for each frame x using a SoftMax layer. The network is trained under the cross-entropy criterion, maximized using the stochastic gradient ascent algorithm.
9. Databases: The Urban Sound 8k dataset used for model training, can be downloaded from the following. Out of which for initial experimental purposes the sound of AC vs Car Horn is taken. Another related dataset that can be used is Google's AudioSet. And for testing purposes actual sound of AC compressor is captured and then converted into the machine usable code.
10. Filter Analysis: Spectrograms of the actual datasets are as follows:

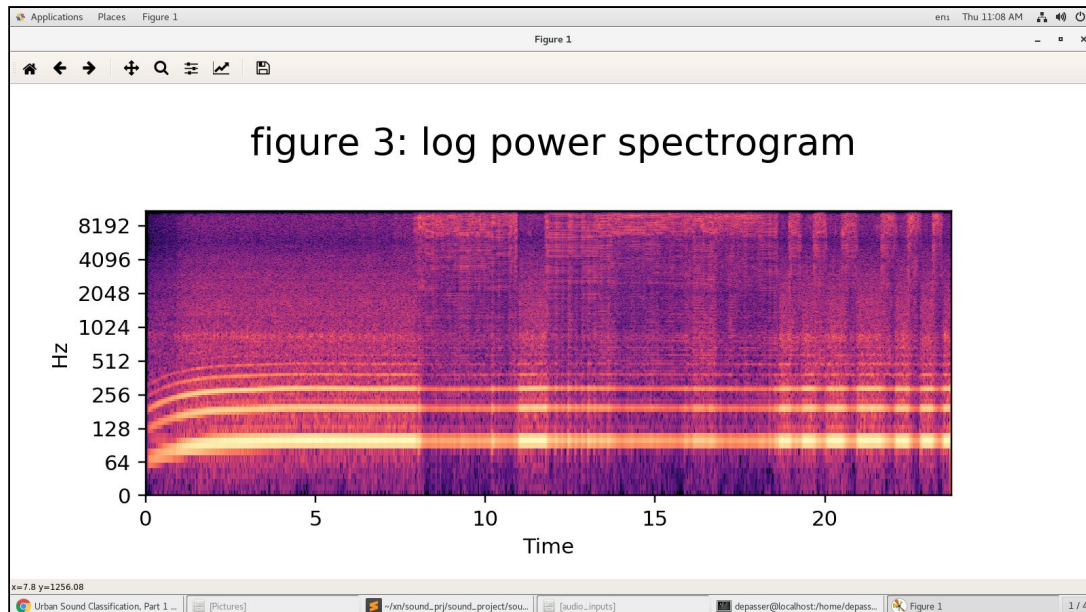






The above spectrogram are of the healthy sounds. Healthy sounds means how the actual sound can be visualized as.





The above two spectrograms are those of the faulty sound, which means how the machine will sound when it is faulty.

Implementation Methodologies:

I have used three types of methods to implement and carry out a sample experiment. As mentioned in the semester-2 and semester - 3, I have been researching which of the one ANN or CNN should be used. Well, it turns out I ended up using both of them. Along with that one more method I seeked is of Anomaly Detection which will be explained later on.

Method 1: Using ANN.

1. Since there exists only two classes the sound of AC compressor and Sound of car horn, this turns out to be a two class classification problem.
2. Feature Extraction: First, using the gathered data, the feature extraction process is carried out.

The Tools I used for feature extraction are :

1. Python libraries : Librosa, Numpy, Pandas

Librosa is one of the famous library for audio analysis in python. Using librosa first the short time fourier transform (STFT) is obtained. Then mel-frequency cepstral coefficients are obtained (MFCCs). Third, chroma is obtained or extracted from the audio source. Then mel-frequency, spectral_contrast, tonnetz (harmonics) are extracted. All the feature extraction tools are available in the librosa library.

The extracted features are then stored in npy files so that i can be loaded while

the training process.

Following is the code snippet for the feature extraction:

```
def extract_feature(filename):
    X, sample_rate = librosa.load(filename)
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,
axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,
axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T, axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft,
sr=sample_rate).T, axis=0)
    tonnetz =
np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),sr=sample_rate).T,
axis=0)
    return mfccs, chroma, mel, contrast, tonnetz
```

3. Parsing the Audio files from the source is the next step.

4. Building a neural network in tensor flow.

5. The network constitutes of layers :

1. Input layer: Weights-> 128×4 , Bias -> 128×4 , shape of the input -> $[n_dim, 128 \times 4]$, activation function used is sigmoid
2. Hidden layer: Weights -> 128×6 , Bias -> 128×6 , shape of the hidden layer -> $[128 \times 4, 128 \times 6]$, activation function used is sigmoid
3. Output layer: Weights -> $n_classes$, Bias -> $n_classes$, shape of the output layer -> $[128 \times 6, n_classes]$, activation function used is softmax.
4. Complete network code is as follows:

```
#input layer
Weights_1 = tf.Variable(tf.random_normal([n_dim, n_hidden_units_one], mean
=0, stddev = sd), name="Weights_1")
bias_1 = tf.Variable(tf.random_normal([n_hidden_units_one], mean=0,
stddev=sd), name="bias_1")
activation_1 = tf.nn.sigmoid(tf.matmul(X,Weights_1) + bias_1,
name="activation_1")

#hidden layer
Weights_2 =
tf.Variable(tf.random_normal([n_hidden_units_one,n_hidden_units_two],
mean=0, stddev=sd), name="Weights_2")
```

```

bias_2 = tf.Variable(tf.random_normal([n_hidden_units_two], mean=0,
stddev=sd), name="bias_2")
activation_2 = tf.nn.sigmoid(tf.matmul(activation_1,Weights_2) + bias_2,
name="activation_2")

#output layer - 2 layers
Weights_ = tf.Variable(tf.random_normal([n_hidden_units_two, n_classes],
mean=0, stddev = sd), name="Weights_")
bias_ = tf.Variable(tf.random_normal([n_classes], mean=0, stddev=sd),
name="bias_")
y_ = tf.nn.softmax(tf.matmul(activation_2,Weights_) + bias_, name="y_")

```

6. Training Process: Following are the intricacies of the training process

1. training_epochs -> 15000
2. The cost function used is mean_squared_error
3. The optimizer used is GradientDescent with learning rate of 0.001
4. The entire training set is passed to the network

Method 2: Using CNN

1. Since there exists two classes we will be using two class classification building the CNN network

2. Feature Extraction:

The feature extraction method used in this method is different from what was used in the method 1

The tools required for making this feature extraction method are:

1. Python Libraries : Librosa, Numpy

Here, we make use of windows / kernels of predefined size and divide the dataset. Using the windows we clip the audio data and then the features are extracted.

The extracted features are bit different from what we saw in the earlier method.

The features are : melspectrogram, logamplitude.

Following is the code snippet on feature extraction

```

def windows(data, window_size):
    start = 0
    while start < len(data):
        yield int(start), int(start + window_size)
        start += (window_size/2)
def extract_features(file_paths, frames = 41, bands = 60):
    window_size = 512 * (frames-1)
    log_specgrams = []
    counter = 0
    for f in file_paths:
        path =
'/home/depasser/xn/sound_prj/sound_project/audio_inputs/'+f
        for files in os.listdir(path):
            path2 = path + '/' + files
            try:
                counter += 1
                print("extracting file {}:{}".format(f,counter))
                sound_clip , s = librosa.load(path2)
                for i in windows(sound_clip, window_size):
                    if(len(sound_clip[i[0]:i[1]]) == window_size):
                        signal = sound_clip[i[0]:i[1]]
                        melspec =
librosa.feature.melspectrogram(signal, n_mels=bands)
                        logspec = librosa.logamplitude(melspec)
                        logspec = logspec.T.flatten()[:,
np.newaxis].T
                        log_specgrams.append(logspec)
            except Exception as e:
                print("Exception:{}, for {}".format(e,f))
        log_specgrams =
np.asarray(log_specgrams).reshape(len(log_specgrams), bands,
frames,1)
        features = np.concatenate((log_specgrams,
np.zeros(np.shape(log_specgrams))), axis= 3)
        for i in range(len(features)):
            features[i,:,: , 1] = librosa.feature.delta(features[i,:,: ,0])
    return np.array(features)

```

3. Then the extracted feature is saved into npy files.

4. Then the CNN is built

5. Following are the intricacies of the network:

Network consists of following layers

1. Conv layer1 -> weights -> [kernel_size, kernel_size, num_channels, 32] , bias -> 32, activation used is relu,
2. Conv layer2 -> weights -> [kernel_size, kernel_size, 32, 64], bias -> 64 , activation used is relu
3. Fully connected -> weights -> 1024, bias -> 64, activation used is relu
4. Drop out -> keep_rate is 80%
5. Output layer -> weights -> [1024, num_labels], bias -> num_labels
6. Following is the code snippet for the same

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,2,2,1], padding='SAME')
def cnn(x):
    weights = {
        'W_conv1' : tf.Variable(tf.random_normal([kernel_size, kernel_size,
num_channels, 32])),
        'W_conv2' : tf.Variable(tf.random_normal([kernel_size, kernel_size, 32,
64])),
        'W_fc' : tf.Variable(tf.random_normal([1024])),
        'out' : tf.Variable(tf.random_normal([1024, num_labels]))
    }
    biases = {
        'b_conv1' : tf.Variable(tf.random_normal([32])),
        'b_conv2' : tf.Variable(tf.random_normal([64])),
        'b_fc' : tf.Variable(tf.random_normal([1024])),
        'out' : tf.Variable(tf.random_normal([num_labels]))
    }
    conv1 = tf.nn.relu(conv2d(x, weights['W_conv1']) + biases['b_conv1'])
    conv2 = tf.nn.relu(conv2d(conv1, weights['W_conv2']) +
biases['b_conv2'])
    shape = conv2.get_shape().as_list()
    fc = tf.reshape(conv2, [-1, shape[1]*shape[2]*shape[3]])
    fc = tf.nn.relu(tf.matmul(fc, weights['W_fc']) + biases['b_fc'])
    fc = tf.nn.dropout(fc, keep_rate)
    output = tf.matmul(fc, weights['out']) + biases['out']
    return output
```

Note : Max-pooling is not used for experimental purposes.

6. Training process: Following are the details of the training process

1. Training_epochs -> 10
2. Cost function used is softmax_cross_entropy_with_logits
3. Optimizer used is AdamOptimizer with learning rate 0.01
4. The training process is carried out in batches

Method 3: Using Anomaly Detection Approach

1. In this method, I used an approach by training the algorithm only one of the data and if the results produces a value which is far greater value say a threshold to be of 5, anything beyond that it means the audio is not what it listened to.
2. The feature extraction process is same as that of the ann.
3. The model is built in keras framework and a simple neural network is built
4. Details of the network are :
 1. Sequential model Input layer -> weights -> 64, activation used is relu
 2. Dense output layer-> weights -> 193
 3. Code snippet details are as follows:

```
def build_model():  
    model = Sequential()  
    model.add(Dense(64, input_dim=193, activation='relu'))  
    model.add(Dense(193))  
    return model
```

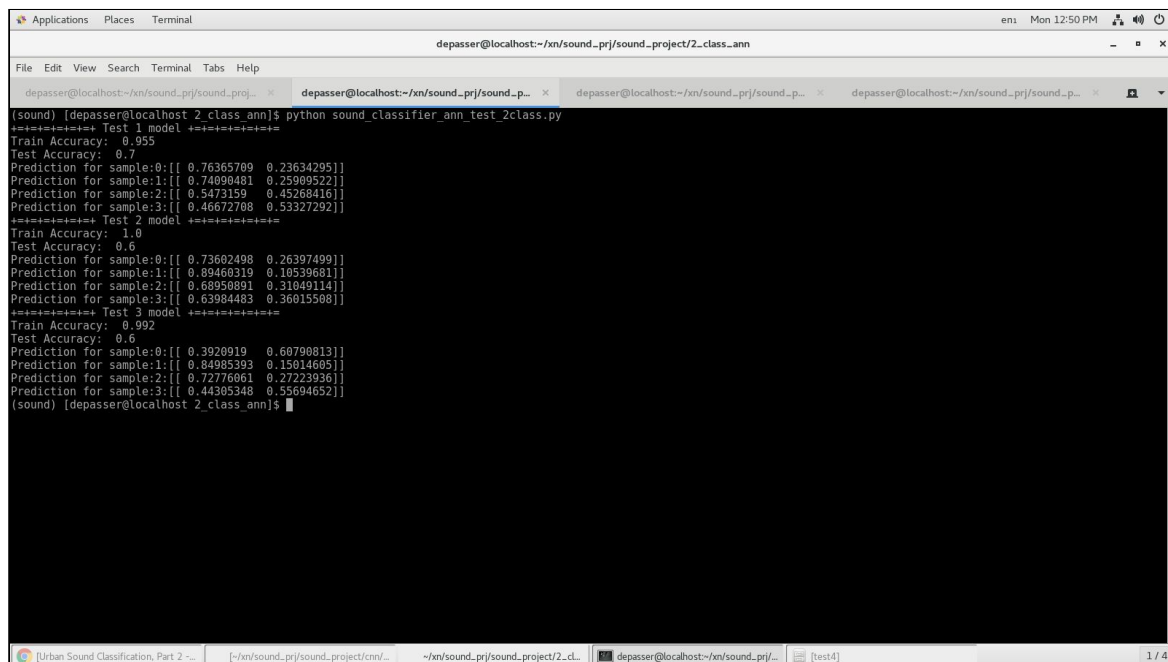
5. Details of the training are :

1. Training Epochs -> 2500
2. Cost Function used is Mean Squared Error
3. Optimizer -> Adam Optimizer
4. Batch Size -> 1
5. Check points -> For every variation in the val loss best model is saved respective to the epoch

Analysis of the Result:

1. It is Observed that **Feature extraction plays a major role** in the concept of intelligent hearing. Since human ears are so developed over the course of thousands of years, it is only fair to use a high-end microphone
2. **Noise Cancellation or De noise algorithm** is required to be implemented as despite the quality of the microphone it is highly required such that noise in the inputs are minimal
3. It is also observed that humans have a special capability of **selective hearing**, which needs to be implemented here too so as to focus on the required audio to process.
4. In method 1 the output turns out to be quite good despite being much a simpler network than the CNN. Here the feature extraction methods bit vary from that used in the CNN. But nonetheless the concept is same.

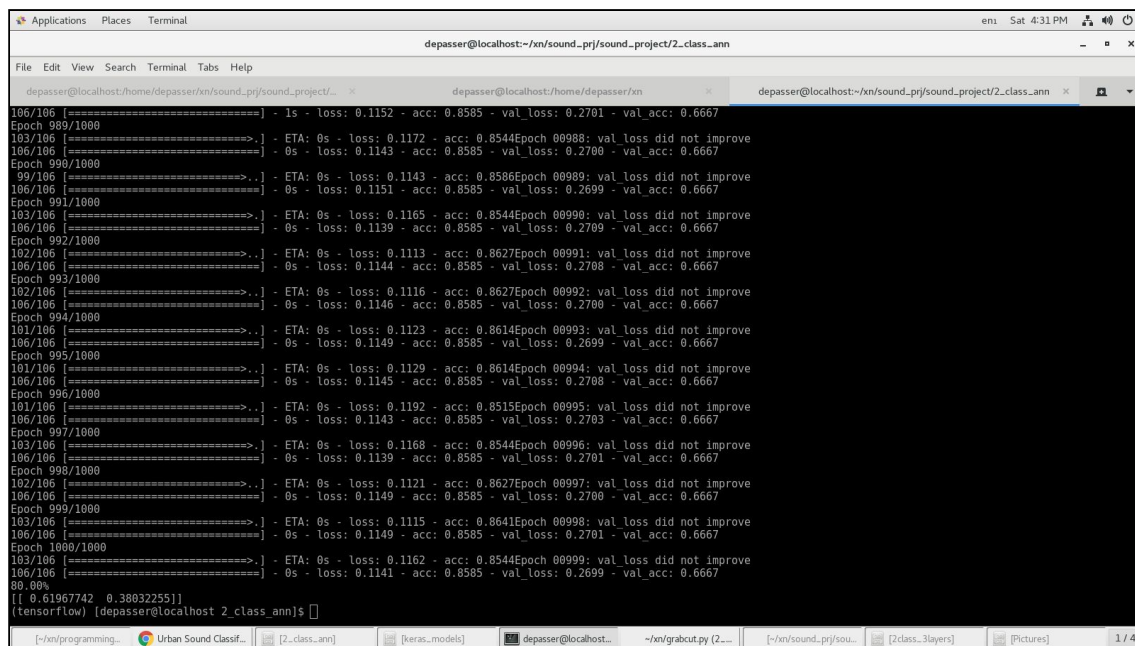
In method 1, an accuracy of over 95% is achieved, i.e. the network is capable of differentiating between the sound of an AC vs a Car Horn.



```
(sound) [depasser@localhost 2_class_ann]$ python sound_classifier_ann_test_2class.py
+++++++ Test 1 model ++++++
Train Accuracy: 0.955
Test Accuracy: 0.7
Prediction for sample:0:[[ 0.76365789  0.23634295]]
Prediction for sample:1:[[ 0.74090481  0.25909522]]
Prediction for sample:2:[[ 0.5473159  0.45268416]]
Prediction for sample:3:[[ 0.46672788  0.53327292]]
+++++++ Test 2 model ++++++
Train Accuracy: 1.0
Test Accuracy: 0.6
Prediction for sample:0:[[ 0.73602498  0.26397499]]
Prediction for sample:1:[[ 0.89460319  0.10539681]]
Prediction for sample:2:[[ 0.68950891  0.31049114]]
Prediction for sample:3:[[ 0.63984483  0.36015508]]
+++++++ Test 3 model ++++++
Train Accuracy: 0.992
Test Accuracy: 0.6
Prediction for sample:0:[[ 0.3920919  0.60790813]]
Prediction for sample:1:[[ 0.84985393  0.15014605]]
Prediction for sample:2:[[ 0.72776061  0.27223936]]
Prediction for sample:3:[[ 0.44305348  0.55694652]]
(sound) [depasser@localhost 2_class_ann]$
```

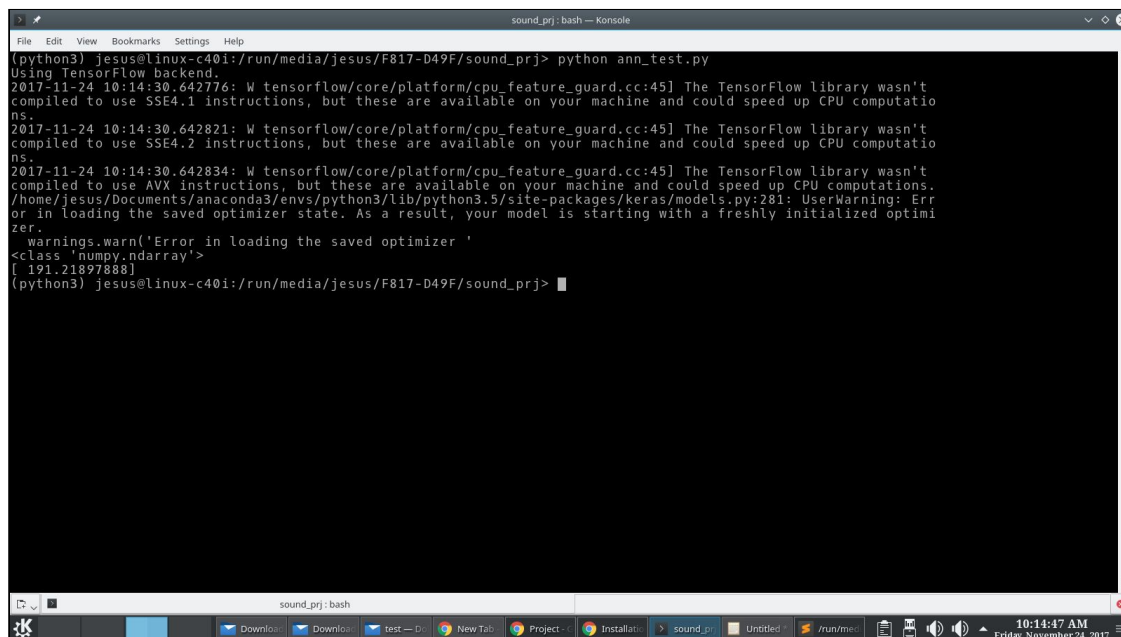
5. In method 2, the CNN model, there are chances that due to not using max pooling techniques and bit variations in the feature extraction techniques, the model is only able to achieve over 80% of accuracy. There are chances that

optimized hyperparameters are yet to be found.



```
depasser@localhost:~/xn/sound_prj/sound_project/2_class_ann
File Edit View Search Terminal Tabs Help
depasser@localhost:~/xn/sound_prj/sound_project/2_class_ann
Epoch 989/1000 - 1s - loss: 0.1152 - acc: 0.8585 - val_loss: 0.2701 - val_acc: 0.6667
Epoch 990/1000 - ETA: 0s - loss: 0.1172 - acc: 0.8544Epoch 00988: val_loss did not improve
Epoch 991/1000 - 0s - loss: 0.1143 - acc: 0.8585 - val_loss: 0.2700 - val_acc: 0.6667
Epoch 992/1000 - ETA: 0s - loss: 0.1143 - acc: 0.8586Epoch 00989: val_loss did not improve
Epoch 993/1000 - 0s - loss: 0.1151 - acc: 0.8585 - val_loss: 0.2699 - val_acc: 0.6667
Epoch 994/1000 - ETA: 0s - loss: 0.1165 - acc: 0.8544Epoch 00990: val_loss did not improve
Epoch 995/1000 - 0s - loss: 0.1139 - acc: 0.8585 - val_loss: 0.2709 - val_acc: 0.6667
Epoch 996/1000 - ETA: 0s - loss: 0.1113 - acc: 0.8627Epoch 00991: val_loss did not improve
Epoch 997/1000 - 0s - loss: 0.1144 - acc: 0.8585 - val_loss: 0.2708 - val_acc: 0.6667
Epoch 998/1000 - ETA: 0s - loss: 0.1116 - acc: 0.8627Epoch 00992: val_loss did not improve
Epoch 999/1000 - 0s - loss: 0.1146 - acc: 0.8585 - val_loss: 0.2700 - val_acc: 0.6667
Epoch 1000/1000 - ETA: 0s - loss: 0.1123 - acc: 0.8614Epoch 00993: val_loss did not improve
Epoch 1001/1000 - 0s - loss: 0.1149 - acc: 0.8585 - val_loss: 0.2699 - val_acc: 0.6667
Epoch 1002/1000 - ETA: 0s - loss: 0.1129 - acc: 0.8614Epoch 00994: val_loss did not improve
Epoch 1003/1000 - 0s - loss: 0.1145 - acc: 0.8585 - val_loss: 0.2708 - val_acc: 0.6667
Epoch 1004/1000 - ETA: 0s - loss: 0.1192 - acc: 0.8515Epoch 00995: val_loss did not improve
Epoch 1005/1000 - 0s - loss: 0.1143 - acc: 0.8585 - val_loss: 0.2703 - val_acc: 0.6667
Epoch 1006/1000 - ETA: 0s - loss: 0.1168 - acc: 0.8544Epoch 00996: val_loss did not improve
Epoch 1007/1000 - 0s - loss: 0.1139 - acc: 0.8585 - val_loss: 0.2701 - val_acc: 0.6667
Epoch 1008/1000 - ETA: 0s - loss: 0.1121 - acc: 0.8627Epoch 00997: val_loss did not improve
Epoch 1009/1000 - 0s - loss: 0.1149 - acc: 0.8585 - val_loss: 0.2700 - val_acc: 0.6667
Epoch 1010/1000 - ETA: 0s - loss: 0.1115 - acc: 0.8641Epoch 00998: val_loss did not improve
Epoch 1011/1000 - 0s - loss: 0.1149 - acc: 0.8585 - val_loss: 0.2701 - val_acc: 0.6667
Epoch 1012/1000 - ETA: 0s - loss: 0.1162 - acc: 0.8544Epoch 00999: val_loss did not improve
Epoch 1013/1000 - 0s - loss: 0.1141 - acc: 0.8585 - val_loss: 0.2699 - val_acc: 0.6667
80.00%
[[ 0.61967742 0.38032255]]
(tensorflow) [depasser@localhost 2_class_ann]$
```

6. In method 3, which is totally a different approach, has no comparison with respective to the earlier two mentioned algorithms. Nonetheless the results of third method showed that it detects the sound of an AC compressor. Now, the most challenging part is to generalize the threshold that is the decision maker in this case



```
sound_prj: bash — Konsole
File Edit View Bookmarks Settings Help
(python3) jesu@linux-c40i:/run/media/jesu/F817-D49F/sound_prj> python ann_test.py
Using TensorFlow backend.
2017-11-24 10:14:30.642776: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computatio
ns.
2017-11-24 10:14:30.642821: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computatio
ns.
2017-11-24 10:14:30.642834: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't
compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.
/home/jesu/Documents/anaconda3/envs/python3/lib/python3.5/site-packages/keras/models.py:281: UserWarning: Err
or in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimi
zer.
  warnings.warn('Error in loading the saved optimizer '
<class 'numpy.ndarray'>
[ 191.21897888]
(python3) jesu@linux-c40i:/run/media/jesu/F817-D49F/sound_prj>
```

Conclusion:

1. It is clear that a new Feature Extraction method should be defined found or researched for.
2. The Audio input instrument also plays a vital role in the sound classification / sound recognition.
3. There should be a Denoise / Noise Cancellation technique should be implemented whilst the input is recorded.
4. There are chances that using the CNN max pooling technique might result in better results than the ANN.
5. New methods and approaches like those of Anomaly detection needs to be researched more.

Further Enhancement:

1. For Future enhancements, there is need to make a application using ensemble methods for Denoising, Filtering, Recognizing and classifying the algorithm.
2. An actual hardware is to be made as mentioned in the research report using Raspberry PI any of the development boards available.
3. Good Quality input microphone is to be purchased / looked for.
4. The device / gadget is to be implemented in machine fault diagnosis by including more perspectives like rotation speed, vibrations.

References:

<http://karol.piczak.com/papers/Piczak2015-ESC-ConvNet.pdf>

<https://librosa.github.io/librosa/>

<http://www.sciencedirect.com/science/article/pii/S0007850607631928>