ABSTRACT

Nowadays, there are many robberies and thefts being done inside Automated Teller Machine (ATM) and in most cases, there is a threat to peoples lives in such situations.The main idea of the project is to identify an ongoing theft/robbery inside ATM.There are more than 3 million ATM machines all over the world, thus the security of ATM should always be considered. One of the common method is to rob the customer amid transaction or while the staff is filling the machine with cash. The bank uses encryption techniques to ensure secrecy of transactions and other sensitive data also in some cases banks have installed biometric identification such as fingerprint, iris, palm vein patterns etc., but these methods does not ensure customer safety thus here we purpose detection using both audio and video processing, making intrusion detection systems more accurate and reliable. Here we will try to identify abnormal sounds such as glass-breaking, scream, threat dialogue identification etc. using Machine Learning models such as scattering transform and the Principal Component Analysis (PCA) with Gaussian kernel and assist the detection with image classification using You Look Only Once (YOLO)9000 and video captioning with Long Short Term Memory (LSTM) and neuro-evolution techniques.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my guide, Mr. S.INIYAN for his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to me for completing this research work.

Ravi shankar kumawat

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

LSTM	Long Short Term Memory
RNNLM	Recurrent Neural Networks Language Models
PCA	Principal Component Analysis
ATM	Automated Teller Machine
SVM	Support Vector Machine
MFCC	Mel Frequency Cepstral Coefficients
FPR	False Positive Rate
SBV	Spectral Basis Vectors
DNN	Deep Neural Network
NMF	Non-negative Matrix Factorization
RBM	Restricted Boltzman Machine
GMM	Gaussian Mixture Models
HMM	Hidden Markov Models
YOLO	You Look Only Once
RNN	Recurrent Neural Networks
PESQ	Perceptual Evaluation of Speech Quality
NLP	Natural Language Processing

CHAPTER 1

INTRODUCTION

ATM accepts cash request from user, verifies the authenticity of the user to access, ensures user has sufficient amount and dispenses the money. There are numerous cases of ATM robbery amid the ATM transactions due to absence of security in ATM. ATM has various security features to identify fraud detection, transactional errors or hacking but it has very few features to ensure public safety in events of a loot or robbery. This paper reviews the techniques based on Natural Language Processing (NLP), machine learning and video processing that can be applied in order to stop a physical loot/robbery from ATM systems ensuring public safety along with safe guard of physical and capital assets. There are various methods involving physical sensors, audio and video processing to identify any suspicious activities and alarm the authorities in any such scenario.

CHAPTER 2

LITERATURE SURVEY

2.1 Audio sounds classification using scattering features

Author:Souli, S., & Lachiri, Z

Souli and Lachiri (2018) proposes to recognize environmental sounds.Support Vector Machine (SVM) classifier is used as algorithm.GTZAN dataset ,Mel Frequency Cepstral Coefficients (MFCC), scatter wavelet are the dataset used.The performance measured used is confusion matrix. It retrieves lost high frequencies via consecutive wavelet convolutions and thus high accuracy.

2.2 Reliable detection of audio events

Author:Foggia, P., Petkov, N., Saggese, A., Strisciuglio, N., & Vento, M.

Foggia et al. (2015) proposes detection of audio events for surveillance applications. It also use SVM Classifier as algorithm. Mivia audio events dataset is the dataset.Recognition rate , False Positive Rate (FPR) are the performance measure used.It is robust w.r.t.to background noise and real environments.

2.3 Feature extraction based on the high-pass filtering

Author:Ludena-Choez, J., & Gallardo-Antolin, A

The objective of LudeÃśa-Choez and Gallardo-AntolÃ■n (2015) is Acoustic event classification.The algorithm used for this is Non-negative Matrix Factorization (NMF), Spectral Basis Vectors (SBV). Datasets and parameters used are FBK-Irst, UPC-TALP. Performance and measure used here is Confusion Matrix.It is New front-end for AEC(high pass filtering),better accuracy.

2.4 Robust acoustic event classification using DNN

Author:Sharan, R. V., & Moir, T. J.

Sharan and Moir (2017) proposed Robust acoustic event classifier using DNN. Algorithm used here is Restricted Boltzman Machine (RBM), DNN. Performance and measures used are Positive classification rate Datasets and parameters are RWCP sound scene dataset.More robust and better classification.

2.5 Surveillance system

Author:J. Kotus & K. Lopatka & A. Czyzewski & G. Bogdanis

The objective of the paper Kotus et al. (2016) is Recognize environmental sounds by processing of acoustical data in a multimodal bank operating room . Gaussian Mixture Models (GMM)/Hidden Markov Models (HMM) this the algorithm contains following datasets Handcrafted real scenario performance measures are confusion matrix.Efficient but slightly old.

2.6 Video Captioning

Author:] Gao, L., Guo, Z., Zhang, H., Xu, X., & Shen, H. T.

Gao et al. (2017) proposes video captioning using attention-based LSTM model with semantic consistency trained on MSR-VTT, MSVD datasets, evaluated with BLEU, METEOR.This proposed a new framework of aLSTMs which can be extended to domain specific datasets.

2.7 Image Classification:YOLO9000

Author:Redmon, Joseph, and Ali Farhadi.

Redmon and Farhadi (2016) proposes to detect over 9000 objects with state-of-the-art performance and accuracy using YOLO algorithm on Imagenet1000, COCO datasets.This is a real-time framework for detection 9000 object categories, jointly optimizes detection and classification.

2.8 Automatic Speech Recognition

Author:Xu, Hainan, et al.

Xu et al. (2018) proposes automatic speech recognition using Recurrent Neural Networks (RNN) language models Recurrent Neural Networks Language Models (RNNLM) using 3-gram model and interpolation weights as features and evaluated with WERs of different rescoring. This is a low heuristics and pruned n-gram model which allows for faster, better and accurate results.

2.9 Dual-channel noise reduction

Author:Nabi, W., Ben Nasr, M., Aloui, N., & Cherif, A

Nabi et al. (2018) proposes noise reduction from sound signals using coherence function and bionic wavelet on Google sound dataset with Perceptual Evaluation of Speech Quality (PESQ) scores to evaluate.Thus the model better reduces noisy signals exploiting two closely spaced microphones.

2.10 Inference from survey

It has been inferred from the survey to use **YOLO9000!** (**YOLO9000!**) Redmon and Farhadi (2016) for real time object detection, adaptive filtering for noise reduction in audio, RNN for Video captioning and gesture detection, GMM-HMM model for sound classification. The MIVIA dataset can be used with transcription for audio analysis and COCO dataset for image analysis.

CHAPTER 3

MODULES

The system is based on three modules namely audio, video and sensors module in order to retrieve data and then process it to identify any suspicious activities at the site and alarm the authorities in any such case. Figure 3.1 shows the architecture of the system :

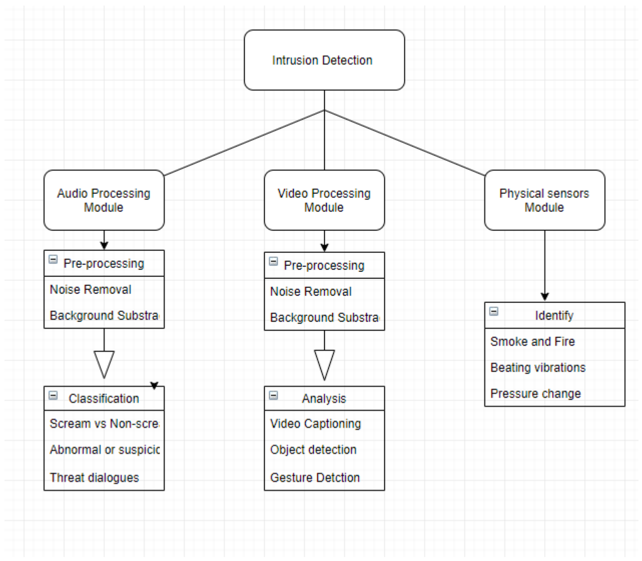


Figure 3.1: Proposed System Architecture

After a suspicious activity is identified: The door will be locked and a silent alarm will be sent to both Police and Bank authorities.

3.1 Audio Module

Audio module analyze sounds and classifies them into suspicious vs non-suspicious sound using microphones. Identifies sounds such as gunfire, screams, threats, glass-breaking etc.

3.1.1 Audio Preprocessing

Pre-processing such as pre-emphasis, creating frames and windows, applying hamming windows, noise reduction using adaptive filtering etc is done.The audio files are transcribed as well.

3.1.2 Feature Extraction

Extract features such as string edit distance, WER, SER, MFCC, mel-filter bank, FFT etc.

3.1.3 Classification

The sounds are then classified as suspicious vs non-suspicious using SVM and GMM-HMM model.

3.2 Video Module

Video module uses video camera for identification of weapons, face-masks, punches, pose detection etc

3.2.1 Object Detection

The system implements YOLO9000 for realtime detection of objects.

3.2.2 Video captioning and Gesture Recognition

The system recognizes the gestures of the customers and the intruder to determine if there is any suspicious activity going on and finally generates a captioning for the suspicious clip and sends the transcription in the alert message.

3.3 Physical Sensor Module

This module implements physical sensors to determine the physical status of the room and checks for the physical parameters given below:

3.3.1 Temperature

This identifies very high increase in the room temperature due to actions such as welding torch to break ATM, fire etc.

3.3.2 Smoke

Detects smoke inside the room from situations such as fire etc.

3.3.3 Vibration

Detects vibrations inside the room due to activities such as cutting or making holes in the machine.

The full ER diagram of the system is given below

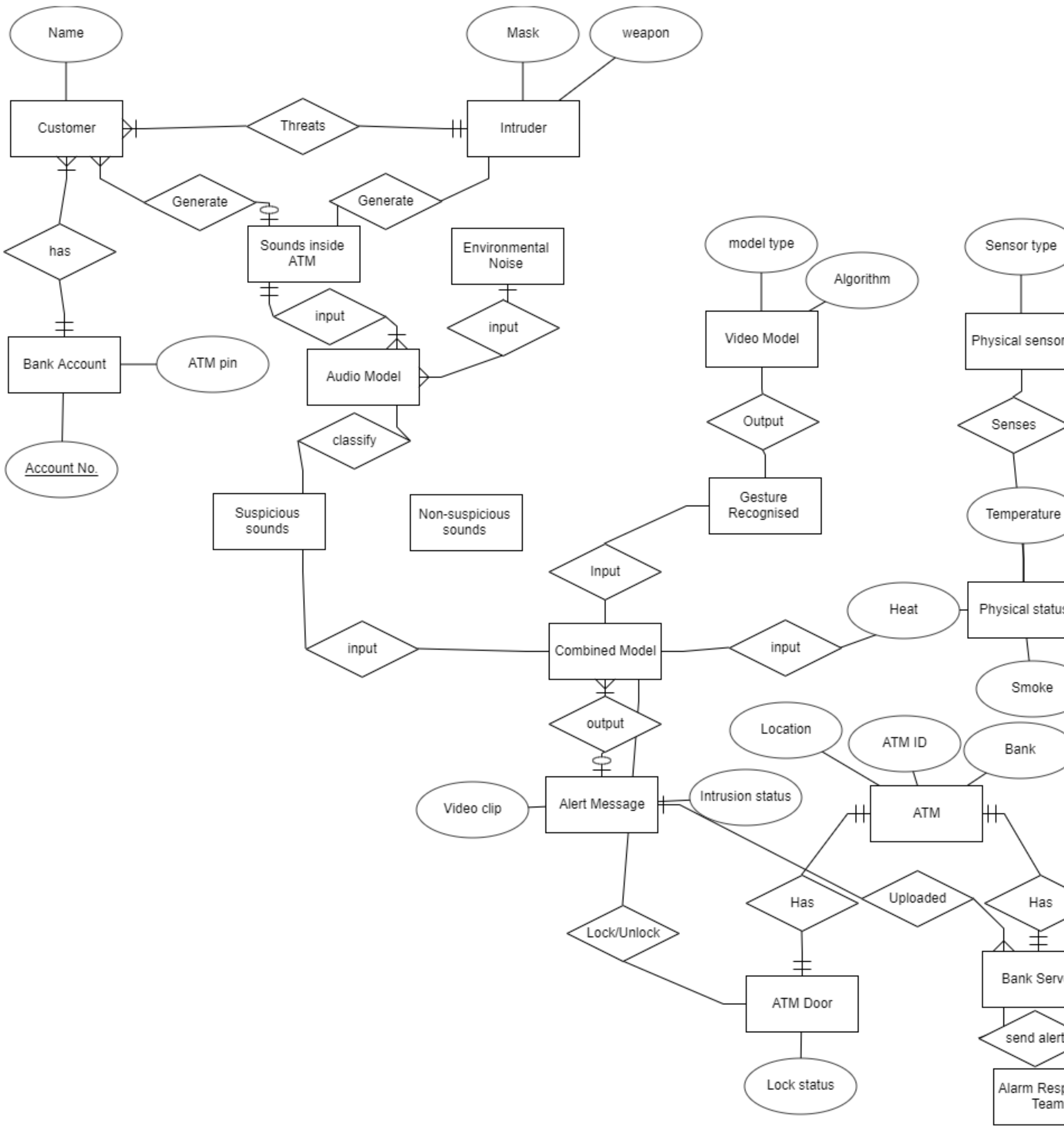


Figure 3.2: ER Diagram

CHAPTER 4

CONCLUSION

In this paper, we presented various techniques to enhance customer safety in ATMs by detecting suspicious criminal activities at the site. The above discussed audio, video and physical sensors can be deployed together to get better accuracy.Our system Identifies and classifies the any suspicious activities, locks the criminal inside and informs the authorities.Thus we hope to reduce criminal activities and increasing property and public lives safety inside ATMs.

CHAPTER 5

FUTURE ENHANCEMENT

In future we may use the recent Neuro-evolution and Metalearning techniques can be applied to find the best model for processing each module utilizing genetic algorithms.Also techniques for faster video captioning with less training can be seen along with video compression and GUI application for monitoring can be provided to authorities.

APPENDIX A

AUDIO PRE-PROCESSING

```
#Audio Pre-Processing

import numpy as np
import wave
import librosa
import scipy.io.wavfile
from scipy.fftpack import dct
from pydub import AudioSegment
from pydub.silence import split_on_silence
import matplotlib.pyplot as plt
import madmom
import scipy.signal
import os
from scipy.fftpack import fft, ifft,fftshift,fftfreq
from scipy.signal import butter, lfilter, freqz
import IPython.display
get_ipython().run_line_magic('matplotlib', 'inline')


def resize_audio(signal,sample_rate,time=500):
#convert it into 5 sec i.e. if short:pad else trim
if((len(signal)/sample_rate) < time):
no_pads=time *sample_rate -len(signal)
pads=np.zeros(no_pads)
signal=np.append(signal,pads)

else:
signal=signal[0:int(time*sample_rate)]
```

```
return signal
#signal, sample_rate =
   madmom.audio.signal.load_wave_file(audio_file_name)
def emphasize_signal(signal,pre_emphasis = 0.97):
return np.append(signal[0], signal[1:] - pre_emphasis *
   signal[:-1])

#signal=emphasize_signal(signal)
#audio_file_name="1.wav"
#signal, sample_rate = librosa.load(audio_file_name,
   res_type='kaiser_fast')

def
   create_frames(emphasized_signal,size=2048,sample_rate=8000):
#Framing the audio signal
frame_size=size/sample_rate #24ms
   128=sample_rate*time thus time=128/sample_rate
frame_overlap=frame_size/2 #15ms overlap( ~50%)
frame_length=int(round(frame_size*sample_rate))
stride_length=int(round((frame_overlap)*sample_rate))
# Pad the emphasized_signal with zeros in end corresponding to
   frame size
signal_length=len(emphasized_signal)
print("signal_length:"+str(signal_length)+"
   frame_length:"+str(frame_length)+"
   stride_length:"+str(stride_length))

no_strides=np.ceil((signal_length/stride_length))
print("no_strides:"+str(no_strides))
no_pads=int(abs(signal_length-no_strides*stride_length))
print("no_pads:"+str(no_pads))
z=np.zeros(no_pads)
print(np.shape(z))
emphasized_signal=np.append(emphasized_signal,z)
```

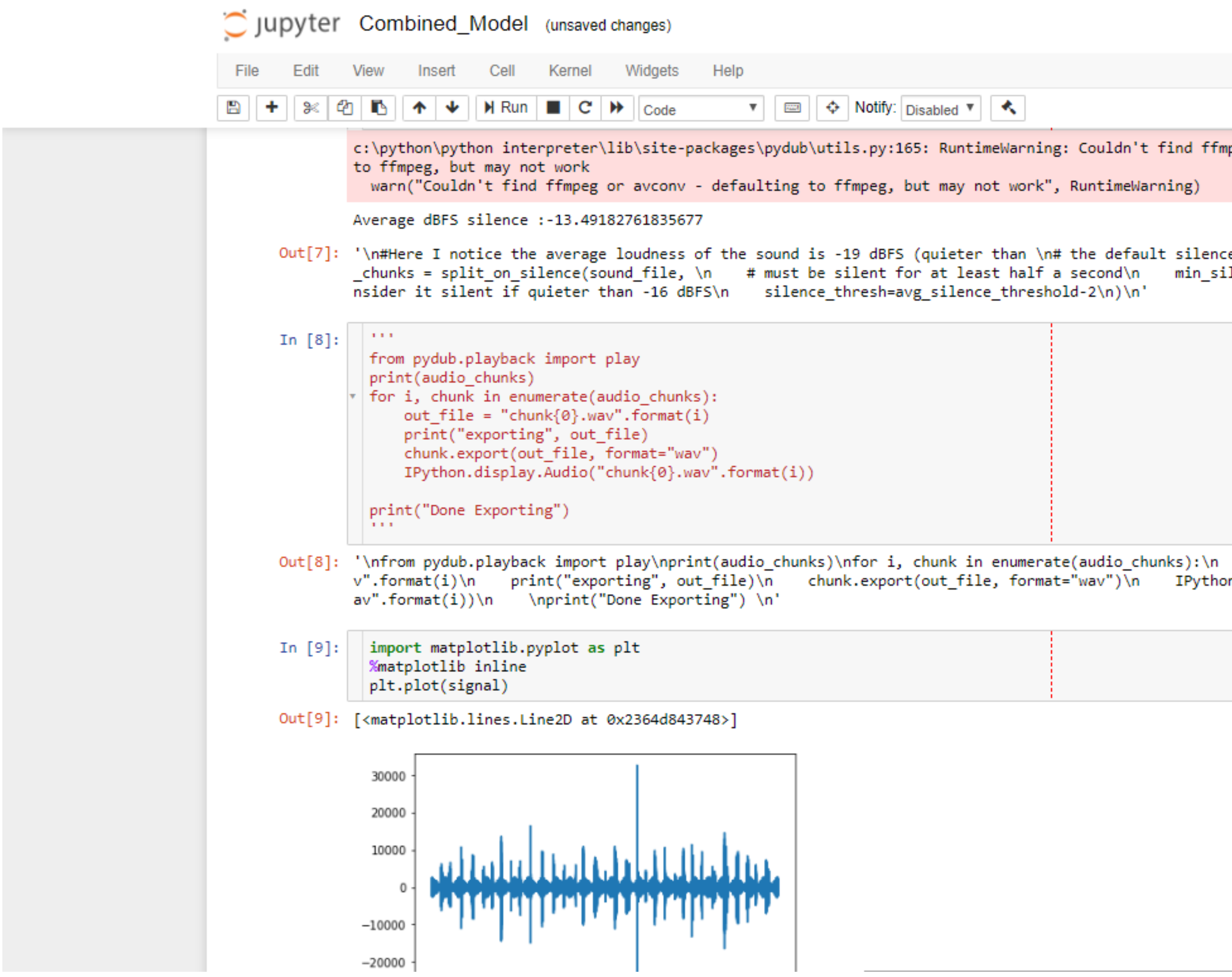
```
no_frames=int(len(emphasized_signal))/(stride_length)
print("no_frames:"+str(no_frames))
print(np.shape(emphasized_signal))

frames=[]
counter=0
for i in range(0,len(signal)):
#frames[counter]=signal[i:(i+frame_length)]
frames.insert(counter, signal[i:(i+frame_length)])
i+=stride_length
counter+=1

return frames



path=''
def create_chunks(sound):
#for i in sounds:
sound_file = AudioSegment.from_wav(sound)
print("Average dBFS silence :"+str(sound_file.dBFS))
avg_silence_threshold=sound_file.dBFS
audio_chunks = split_on_silence(sound_file,
   min_silence_len=500,silence_thresh=avg_silence_threshold-2)
for j, chunk in enumerate(audio_chunks):
out_file = "chunk{0}.wav".format(j)
print("exporting", out_file)
chunk.export(path+out_file, format="wav")
print("Done Exporting")



intrusion_sounds=os.listdir("Dataset/1_intrusion")

#get the audio
```

```
audio_file_name="atm.wav"
signal, sample_rate = librosa.load(audio_file_name,
   res_type='kaiser_fast')
print("Length of signal before resizing:
   "+str(len(signal)/sample_rate))

signal=resize_audio(signal,sample_rate)
print("Length of signal after resizing:
   "+str(len(signal)/sample_rate))

#Signal pre-emphasis
#signal, sample_rate = librosa.load(audio_file_name,
   res_type='kaiser_fast')
plt.subplot(2,2,1)
plt.plot(signal)


emphasized_signal=emphasize_signal(signal)
plt.subplot(2,2,2)
plt.plot(signal)
print("Length of signal after emphasis:
   "+str(len(emphasized_signal)/sample_rate))


frames=create_frames(emphasized_signal,2048,8000)
plt.subplot(2,2,3)
plt.plot(frames[550])


# Steps:-
#
# 0)Resample,attenuate,normalize etc the signal to
   sample_rate=48000Hz
#
```

```
# 1)Divide the audio into frames with han windows and
#
# 2)then perform noise reduction ,
#
# 3)then combine the signal again and
#
# 4)divide it into sound chunks for further classification

frames=np.vstack(frames).astype(None)


print(len(emphasized_signal)/sample_rate)
scipy.io.wavfile.write("resized_signal.wav",sample_rate,signal.astype('int1
IPython.display.Audio("resized_signal.wav")


def butter_lowpass(cutoff, fs, order=5):
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='low', analog=False)
return b, a

def butter_lowpass_filter(frame, cutoff, fs, order=5):
b, a = butter_lowpass(cutoff, fs, order=order)
y = lfilter(b, a, frame)
return y

def butter_highpass(cutoff, fs, order=5):
nyq = 0.5 * fs
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='high', analog=False)
return b, a

def butter_highpass_filter(frame, cutoff, fs, order=5):
```

```
b, a = butter_highpass(cutoff, fs, order=order)
y = lfilter(b, a, frame)
return y

def
   remove_noise(frames,sample_rate=48000,low_cutoff=3000,high_cutoff=800):
denoised_frames=[]
for i in range(0,len(frames)):
#b,a=butter(5, normal_cutoff, btype='low', analog=False)
#fft=scipy.fftpack.fft(frames[i])
filtered_signal=butter_lowpass_filter(frames[i],low_cutoff,sample_rate,5)
   #select frequencies below 3000Hz
filtered_signal=butter_highpass_filter(filtered_signal,high_cutoff,sample_
   #select frequencies above 800Hz
frames[i]=filtered_signal

remove_noise(frames)




#create_chunks(signal)



#feature extraction for each audio, get the mfcc,onsets(abrupt
   changes),zero_cross_rate,roll-off,energy,energy_entropy,
#Spectral centroid ,spectral flux,spectral entropy,spectral
   rolloff, harmonic ratio and pitch,MFCC filterbanks and MFC,
#Chroma features,spectograms

def extract_features(audio_file):
combined_feature=[]
signal, sample_rate =
   madmom.audio.signal.load_wave_file(audio_file)
```

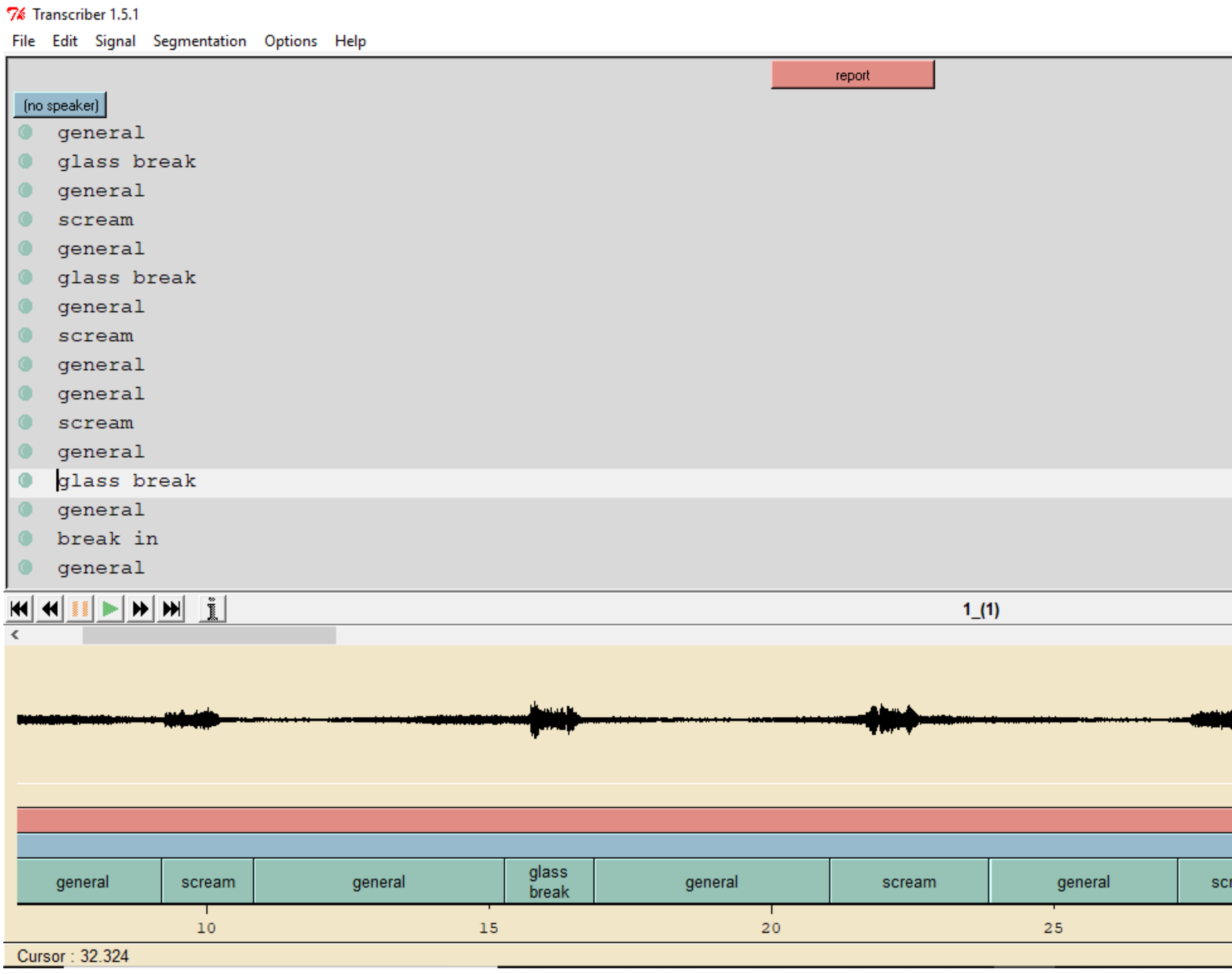
```
signal=madmom.audio.signal.normalize(signal)#normalize the
    signal
y=signal.astype('float')
sr=sample_rate
```



jupyter Combined_Model (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled

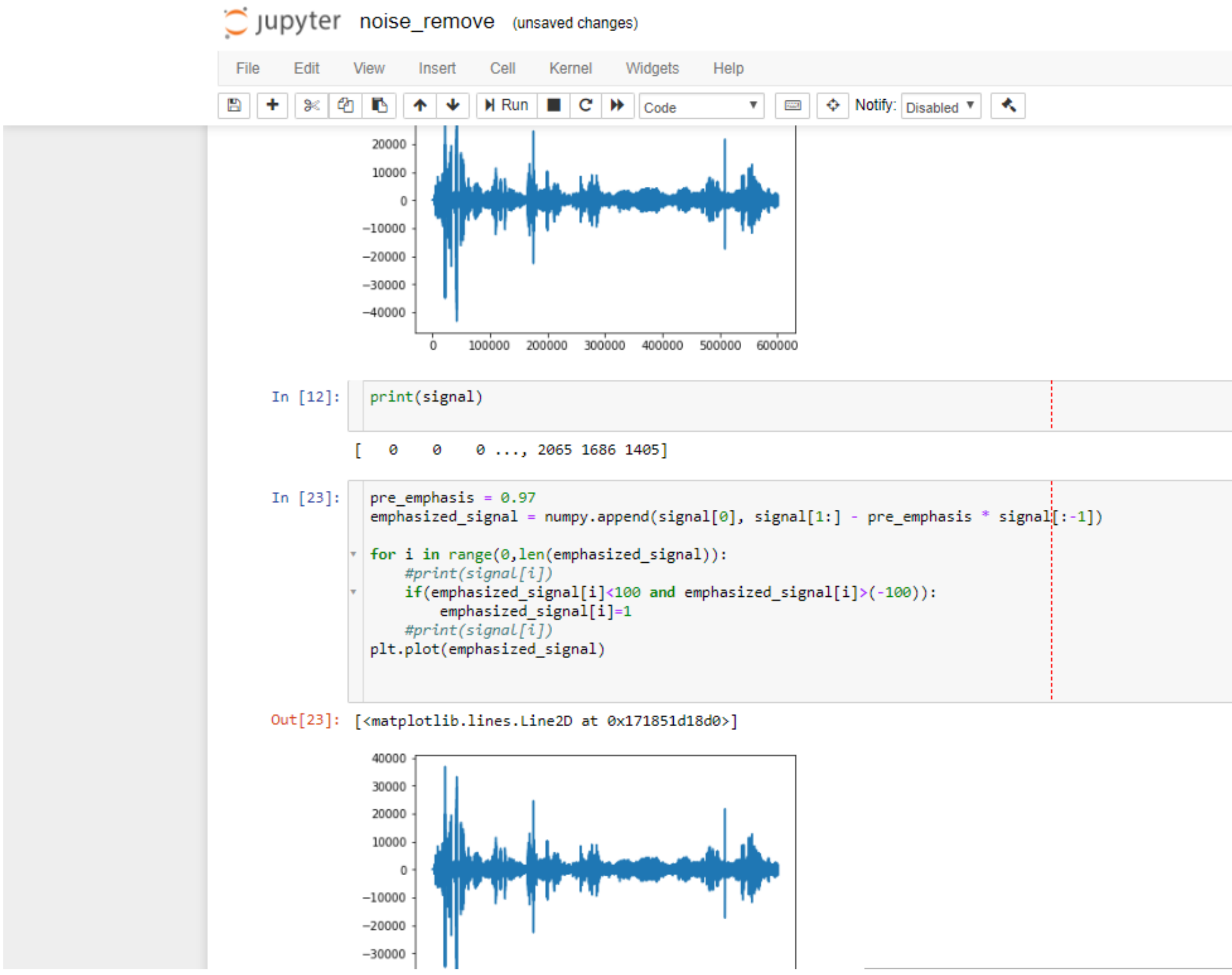
```
c:\python\python interpreter\lib\site-packages\pydub\utils.py:165: RuntimeWarning: Couldn't find ffmp
to ffmpeg, but may not work
  warn("Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work", RuntimeWarning)

Average dBFS silence :-13.49182761835677

Out[7]: '\n#Here I notice the average loudness of the sound is -19 dBFS (quieter than \n# the default silence
        _chunks = split_on_silence(sound_file, \n    # must be silent for at least half a second\n    min_sil
        nsider it silent if quieter than -16 dBFS\n    silence_thresh=avg_silence_threshold-2\n)\n'

In [8]: '''
        from pydub.playback import play
        print(audio_chunks)
        for i, chunk in enumerate(audio_chunks):
            out_file = "chunk{0}.wav".format(i)
            print("exporting", out_file)
            chunk.export(out_file, format="wav")
            IPython.display.Audio("chunk{0}.wav".format(i))

        print("Done Exporting")
        '''

Out[8]: '\nfrom pydub.playback import play\nprint(audio_chunks)\nfor i, chunk in enumerate(audio_chunks):\n
        v".format(i)\n    print("exporting", out_file)\n    chunk.export(out_file, format="wav")\n    IPythor
        av".format(i))\n    \nprint("Done Exporting") \n'

In [9]: import matplotlib.pyplot as plt
        %matplotlib inline
        plt.plot(signal)

Out[9]: [<matplotlib.lines.Line2D at 0x2364d843748>]
```

Figure A.1: Audio signal



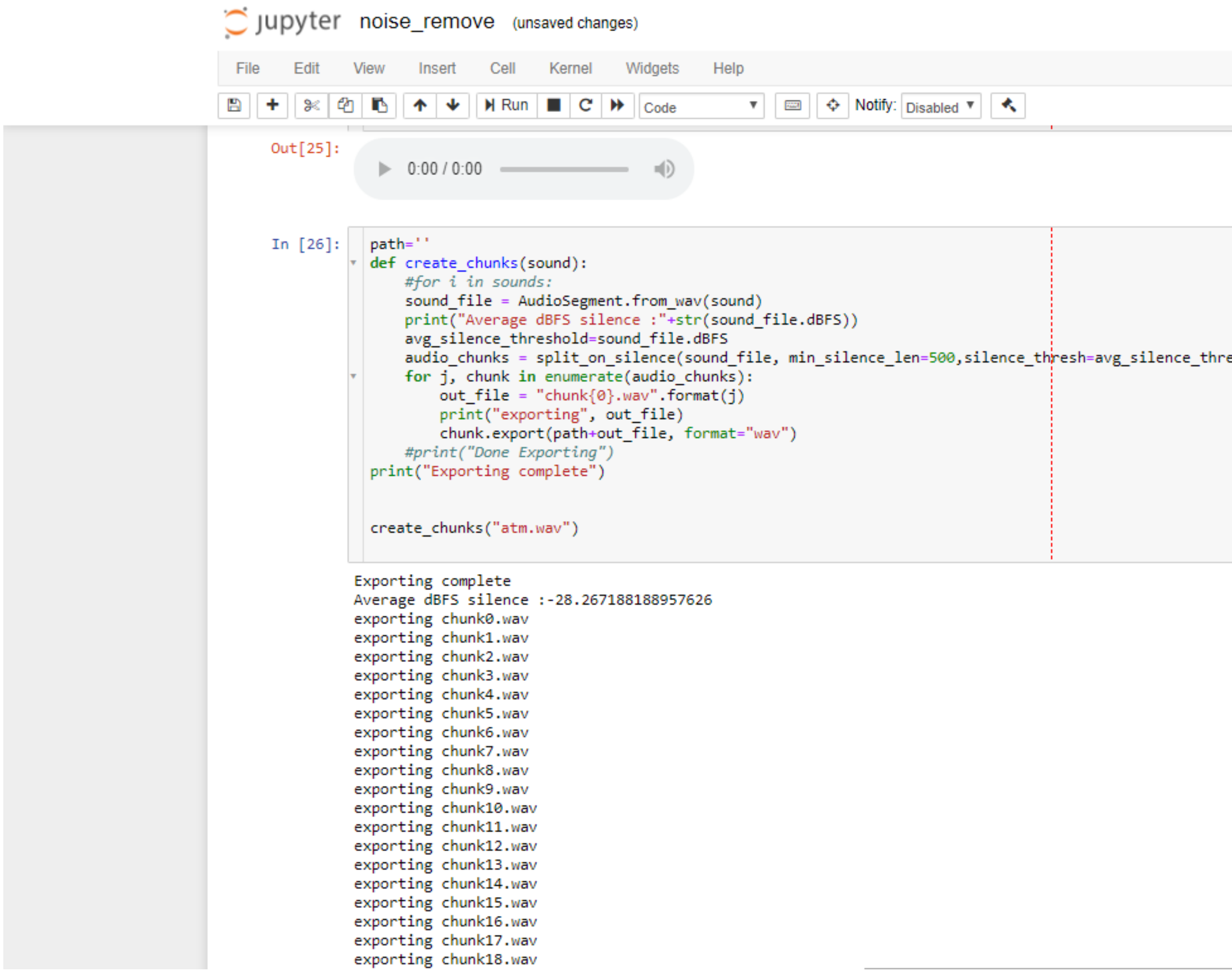
general
glass break
general
scream
general
glass break
general
scream
general
general
scream
general
glass break
general
break in
general

Figure A.2: Transcribed sound



```
In [12]: print(signal)

         [    0     0     0 ..., 2065 1686 1405]

In [23]: pre_emphasis = 0.97
         emphasized_signal = numpy.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])

         for i in range(0,len(emphasized_signal)):
             #print(signal[i])
             if(emphasized_signal[i]<100 and emphasized_signal[i]>(-100)):
                 emphasized_signal[i]=1
             #print(signal[i])
         plt.plot(emphasized_signal)

Out[23]: [<matplotlib.lines.Line2D at 0x171851d18d0>]
```

Figure A.3: Emphasized Signal


jupyter noise_remove (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled
Out[25]:
0:00 / 0:00
In [26]:

```
path=''
def create_chunks(sound):
    #for i in sounds:
    sound_file = AudioSegment.from_wav(sound)
    print("Average dBFS silence :"+str(sound_file.dBFS))
    avg_silence_threshold=sound_file.dBFS
    audio_chunks = split_on_silence(sound_file, min_silence_len=500,silence_thresh=avg_silence_thre
    for j, chunk in enumerate(audio_chunks):
        out_file = "chunk{0}.wav".format(j)
        print("exporting", out_file)
        chunk.export(path+out_file, format="wav")
    #print("Done Exporting")
print("Exporting complete")


create_chunks("atm.wav")
```

```
Exporting complete
Average dBFS silence :-28.267188188957626
exporting chunk0.wav
exporting chunk1.wav
exporting chunk2.wav
exporting chunk3.wav
exporting chunk4.wav
exporting chunk5.wav
exporting chunk6.wav
exporting chunk7.wav
exporting chunk8.wav
exporting chunk9.wav
exporting chunk10.wav
exporting chunk11.wav
exporting chunk12.wav
exporting chunk13.wav
exporting chunk14.wav
exporting chunk15.wav
exporting chunk16.wav
exporting chunk17.wav
exporting chunk18.wav
```

Figure A.4: creating chunks

APPENDIX B

FEATURE EXTRACTION

```
#fft_freqs=madmom.audio.stft.fft_frequencies(num_fft_bins,
   sample_rate) #num_fft_bins=len(fft)/2
mfcc=librosa.feature.mfcc(y=signal.astype('float'),
   sr=sample_rate, S=None, n_mfcc=30)
combined_feature.append(mfcc)

energy=madmom.audio.signal.energy(signal)
combined_feature.append(energy)

rms=madmom.audio.signal.root_mean_square(signal)
combined_feature.append(rms)

spl=madmom.audio.signal.sound_pressure_level(signal,
   p_ref=None) #sound pressure level
combined_feature.append(spl)

fs = madmom.audio.signal.FramedSignal(signal, frame_size=2048,
   hop_size=512)
combined_feature.append(fs)

stft = madmom.audio.stft.STFT(fs) #short-time_fourier transform

spec = madmom.audio.spectrogram.Spectrogram(stft) #magnitudes
   of stft are used for MIR tasks
combined_feature.append(spec)

sf = madmom.features.onsets.spectral_flux(spec) #spectral_flux
combined_feature.append(sf)
```

```
chroma_stft=librosa.feature.chroma_stft(y=signal.astype('float'),
    sr=sample_rate)
combined_feature.append(chroma_stft)



mel1 =
    librosa.feature.melspectrogram(y=signal.astype('float'),
    sr=sample_rate)
combined_feature.append(mel1)

rmse=librosa.feature.rmse(y=y)
combined_feature.append(rmse)

centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
combined_feature.append(centroid)

spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
combined_feature.append(spec_bw)

flatness = librosa.feature.spectral_flatness(y=y)
combined_feature.append(flatness)

rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
combined_feature.append(rolloff)

return combined_feature




extract_features("atm.wav")
```

```
#REMOVE NOISE

import numpy
#import winsound
import scipy.io.wavfile
from scipy.fftpack import dct
from pydub import AudioSegment
from pydub.silence import split_on_silence
import matplotlib.pyplot as plt
import os
intrusion_sounds=os.listdir("Dataset/1_intrusion")



sample_rate, signal = scipy.io.wavfile.read('atm.wav')
#signal = signal[0:int(3.5 * sample_rate)]
# Keep the first 3.5 seconds i.e. in 1 sec 8k movements thus
   in 3.5sec, 3.5*frequency
pre_emphasis = 0.97
emphasized_signal = numpy.append(signal[0], signal[1:] -
   pre_emphasis * signal[:-1])


sample_rate, signal = scipy.io.wavfile.read('atm.wav')
#signal = signal[0:int(7 * sample_rate)]
pre_emphasis = 0.97
emphasized_signal = numpy.append(signal[0], signal[1:] -
   pre_emphasis * signal[:-1])

plt.plot(emphasized_signal)
```

```
print(signal)



pre_emphasis = 0.97
emphasized_signal = numpy.append(signal[0], signal[1:] -
   pre_emphasis * signal[:-1])

for i in range(0,len(emphasized_signal)):
#print(signal[i])
if(emphasized_signal[i]<100 and emphasized_signal[i]>(-100)):
emphasized_signal[i]=1
#print(signal[i])
plt.plot(emphasized_signal)




import IPython

#IPython.display.Audio("1.wav")



scipy.io.wavfile.write("atm.wav", sample_rate,
   emphasized_signal.astype(signal.dtype))
IPython.display.Audio("atm.wav")

path=''
def create_chunks(sound):
#for i in sounds:
sound_file = AudioSegment.from_wav(sound)
print("Average dBFS silence :"+str(sound_file.dBFS))
avg_silence_threshold=sound_file.dBFS
audio_chunks = split_on_silence(sound_file,
   min_silence_len=500,silence_thresh=avg_silence_threshold-2)
for j, chunk in enumerate(audio_chunks):
```

```
out_file = "chunk{0}.wav".format(j)
print("exporting", out_file)
chunk.export(path+out_file, format="wav")
#print("Done Exporting")
print("Exporting complete")


create_chunks("atm.wav")


import audioread
from scipy.signal import lfilter

n = 15 # the larger n is, the smoother curve will be
b = [1.0 / n] * n
a = 1
yy = lfilter(b,a,y)




from scipy.signal import butter, lfilter, freqz



sample_rate, signal = scipy.io.wavfile.read('atm.wav')
#signal = signal[0:int(3.5 * sample_rate)]
# Keep the first 3.5 seconds i.e. in 1 sec 8k movements thus
   in 3.5sec, 3.5*frequency
pre_emphasis = 0.97
emphasized_signal = numpy.append(signal[0], signal[1:] -
   pre_emphasis * signal[:-1])

#Framing the audio signal
frame_size=0.012 #24ms
```

```
frame_overlap=0.006 #15ms overlap( ~50%)
frame_length=int(round(frame_size*sample_rate))
stride_length=int(round((frame_overlap)*sample_rate))
# Pad the emphasized_signal with zeros in end corresponding to
   frame size
signal_length=len(emphasized_signal)
print("signal_length:"+str(signal_length)+"
   frame_length:"+str(frame_length)+"
   stride_length:"+str(stride_length))

no_strides=numpy.ceil((signal_length/stride_length))
print("no_strides:"+str(no_strides))
no_pads=int(abs(signal_length-no_strides*stride_length))
print("no_pads:"+str(no_pads))
z=numpy.zeros(no_pads)
print(numpy.shape(z))
emphasized_signal=numpy.append(emphasized_signal,z)
no_frames=int(len(emphasized_signal))/(stride_length)
print("no_frames:"+str(no_frames))
print(numpy.shape(emphasized_signal))

def create_frames(signal):
frames=[]
counter=0
for i in range(0,len(signal)):
#frames[counter]=signal[i:(i+frame_length)]
frames.insert(counter, signal[i:(i+frame_length)])
i+=stride_length
counter+=1

return frames

frames=create_frames(emphasized_signal)
```

```
N=len(frames[50])
T=1.0/sample_rate
xf = numpy.linspace(0.0, N, T)
#fft_frame=fft(frames[50])
plt.plot(frames[4])

from scipy.fftpack import fft, ifft,fftshift,fftfreq
#plot.set_xlim(0,8000)
N=len(frames[4])
T=1.0/sample_rate
xf = numpy.linspace(0.0, 1.0/(2.0*T), N//2)
fft_frame=fft(frames[4])
plt.plot(xf,numpy.abs(fft_frame[0:N//2]))

IPython.display.Audio(frames[0])

sample_freq =fftfreq(frames[0].size, d =1/sample_rate)
sig_fft = fft(frames[0])
plt.plot(abs(sig_fft))

import scipy.signal
no_frames=int(no_frames)
NFFT=216
for i in range(0,2):
#frames[i] *= numpy.hamming(frame_length)
frames[i] *= scipy.signal.hamming(frame_length)

# frames *= 0.54 - 0.46 * numpy.cos((2 * numpy.pi * n) /
    (frame_length - 1)) # Explicit Implementation **
mag_frames = numpy.absolute(numpy.fft.rfft(frames[i], NFFT)) #
   Magnitude of the FFT
```

```
pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2)) # Power
    Spectrum



plt.plot(fft(frames[3]))
plt.axis([-20,200,-5,100])
plt.show()

bin=

plt.plot(numpy.fft.rfft(frames[3],512))

plt.plot(numpy.fft.fft(frames[3]))

from scipy.signal import butter, filtfilt
import numpy as np

def butter_highpass(cutoff, sample_rate, order=5):
nyq = 0.5 * sample_rate
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='high', analog=False)
return b, a

def butter_highpass_filter(data, cutoff, fs, order=5):
b, a = butter_highpass(cutoff, fs, order=order)
y = filtfilt(b, a, data)
return y

rawdata = np.loadtxt('sampleSignal.txt', skiprows=0)
signal = rawdata
fs = 100000.0

cutoff = 100
```

```
order = 6
conditioned_signal = butter_highpass_filter(signal, cutoff,
    fs, order)



#M1_SCORE
import argparse
import wer
import re

# create a function that calls wer.string_edit_distance() on
    every utterance
# and accumulates the errors for the corpus. Then, report the
    word error rate (WER)
# and the sentence error rate (SER). The WER should include
    the the total errors as well as the
# separately reporting the percentage of insertions, deletions
    and substitutions.
# The function signature is
# num_tokens, num_errors, num_deletions, num_insertions,
    num_substitutions =
    wer.string_edit_distance(ref=reference_string,
    hyp=hypothesis_string)
#
def score(ref_trn=None, hyp_trn=None):
if ref_trn==None or hyp_trn is None:
print("One of the string empty")
return
hyp_d={}
ref_d={}
#pp = pprint.PrettyPrinter(indent=4)
total_sentences=0
total_errors=0
```

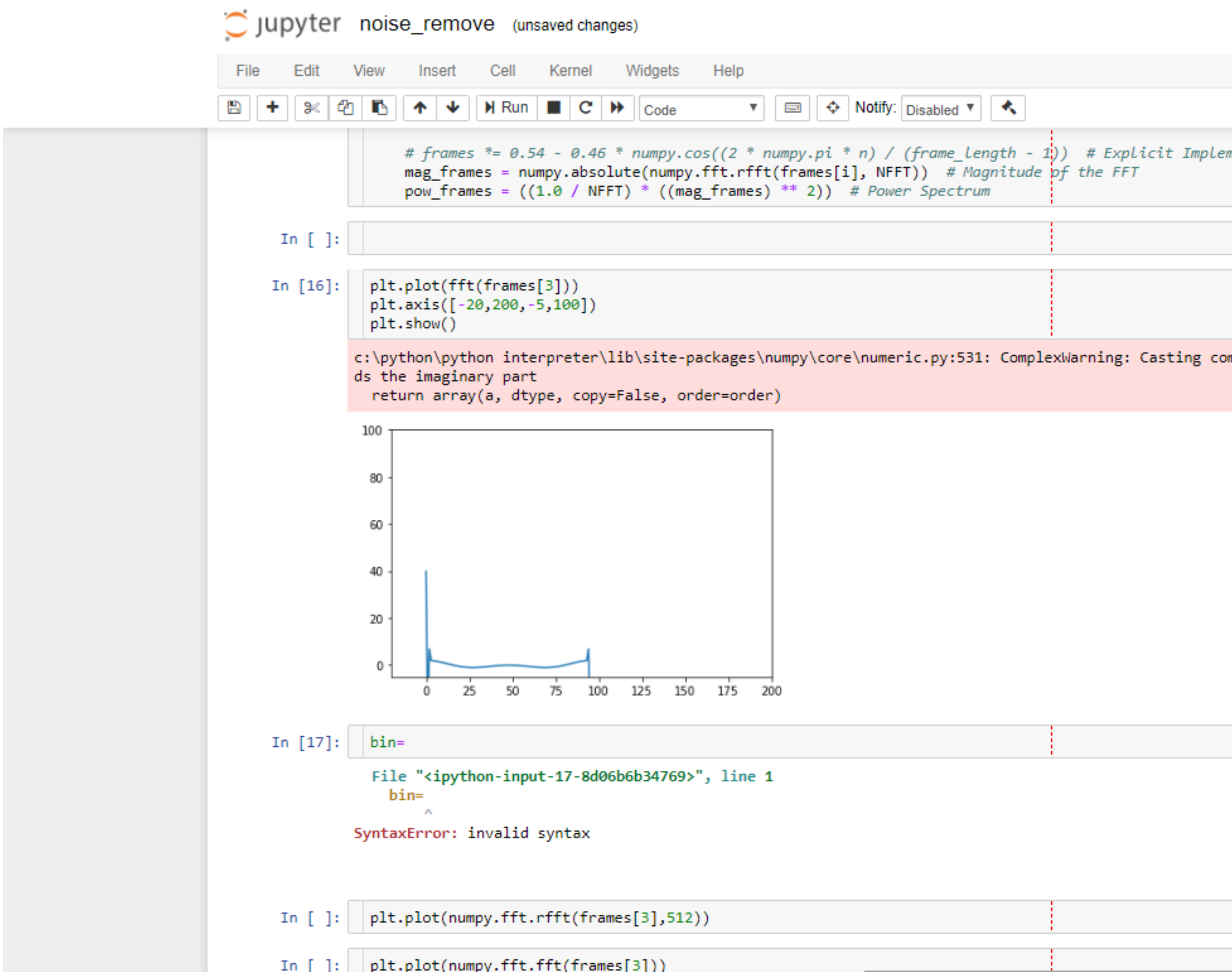
```
with open(hyp_trn,'r') as file:
for line in file:
total_sentences+=1
lst=re.split("[(](.*)[)]",line)
hyp_id=lst[1]
hyp_str=lst[0]
hyp_d[hyp_id]=hyp_str

with open(ref_trn,'r') as file:
for line in file:
lst=re.split("[(](.*)[)]",line)
ref_id=lst[1]
ref_str=lst[0]
ref_d[ref_id]=ref_str

#pp.pprint(ref_d)
#pp.pprint(hyp_d)
total_s=0
total_d=0
total_i=0
total_t=0
total_e=0
for hyp_id in hyp_d:
#print("Checking for Hypothesis ",hyp_d[hyp_id])
#print("Checking with Reference ",ref_d[hyp_id])
num_tokens, num_errors, num_deletions, num_insertions,
   num_substitutions =
   wer.string_edit_distance(ref_d[hyp_id].split(),hyp_d[hyp_id].split())
total_s+=num_substitutions
total_d+=num_deletions
total_i+=num_insertions
total_e+=num_errors
total_t+=num_tokens
if num_errors>0:
```

```
total_errors+=1
print("\n\nID:",hyp_id)
print("N={0} D={1} I={2} S={3}".format(num_tokens,
   num_deletions, num_insertions, num_substitutions))

wer_score=(total_d+total_i+total_s)/total_t
ser_score=total_sentences/total_errors
print("\n\n\nSentence Error Rate :\n")
print("Sum: N={0}
   Err={1}".format(total_sentences,total_errors))
print("Avg: N={0}
   Err={1}%\n\n\n".format(total_sentences,total_sentences/total_errors))

print("\n\n\nWord Error Rate :\n")
print("Sum: N={0} Err={1} Sub={2} Del={3}
   Ins={4}".format(total_t,total_e,total_s,total_d,total_i))
print("Avg: N={0} Err={1}% Sub={2}% Del={3}%
   Ins={4}%".format(total_t,total_e/total_t,total_s/total_t,total_d/total_t

return



if __name__=='__main__':
parser = argparse.ArgumentParser(description="Evaluate ASR
   results.\n"
"Computes Word Error Rate and Sentence Error Rate")
parser.add_argument('-ht', '--hyptrn', help='Hypothesized
   transcripts in TRN format', required=True, default=None)
parser.add_argument('-rt', '--reftrn', help='Reference
   transcripts in TRN format', required=True, default=None)
args = parser.parse_args()

if args.reftrn is None or args.hyptrn is None:
```

```
RuntimeError("Must specify reference trn and hypothesis trn
    files.")

score(ref_trn=args.reftrn, hyp_trn=args.hyptrn)



import os
import soundfile as sf
import numpy as np
import matplotlib.pyplot as plt
import htk_featio as htk
import speech_sigproc as sp

data_dir = '../Experiments'
wav_file='../LibriSpeech/dev-clean/1272/128104/1272-128104-0000.flac'
feat_file=os.path.join(data_dir,'feat/1272-128104-0000.feat')
plot_output=True

if not os.path.isfile(wav_file):
raise RuntimeError('input wav file is missing. Have you
    downloaded the LibriSpeech corpus?')

if not os.path.exists(os.path.join(data_dir,'feat')):
os.mkdir(os.path.join(data_dir,'feat'))

samp_rate = 16000

x, s = sf.read(wav_file)
if (s != samp_rate):
raise RuntimeError("LibriSpeech files are 16000 Hz, found
    {0}".format(s))

fe = sp.FrontEnd(samp_rate=samp_rate,mean_norm_feat=True)
```

```
feat = np.array(fe.process_utterance(x))

if (plot_output):
if not os.path.exists('fig'):
os.mkdir('fig')

# plot waveform
plt.plot(x)
plt.title('waveform')
plt.savefig('fig/waveform.png', bbox_inches='tight')
plt.close()

# plot mel filterbank
for i in range(0, fe.num_mel):
plt.plot(fe.mel_filterbank[i, :])
plt.title('mel filterbank')
plt.savefig('fig/mel_filterbank.png', bbox_inches='tight')
plt.close()

# plot log mel spectrum (fbank)
plt.imshow(feat, origin='lower', aspect=4) # flip the image so
    that vertical frequency axis goes from low to high
plt.title('log mel filterbank features (fbank)')
plt.savefig('fig/fbank.png', bbox_inches='tight')
plt.close()

htk.write_htk_user_feat(feat, feat_file)
print("Wrote {0} frames to {1}".format(feat.shape[1],
    feat_file))

# if you want to verify, that the file was written correctly:
#feat2 = htk.read_htk_user_feat(name=feat_file)
```

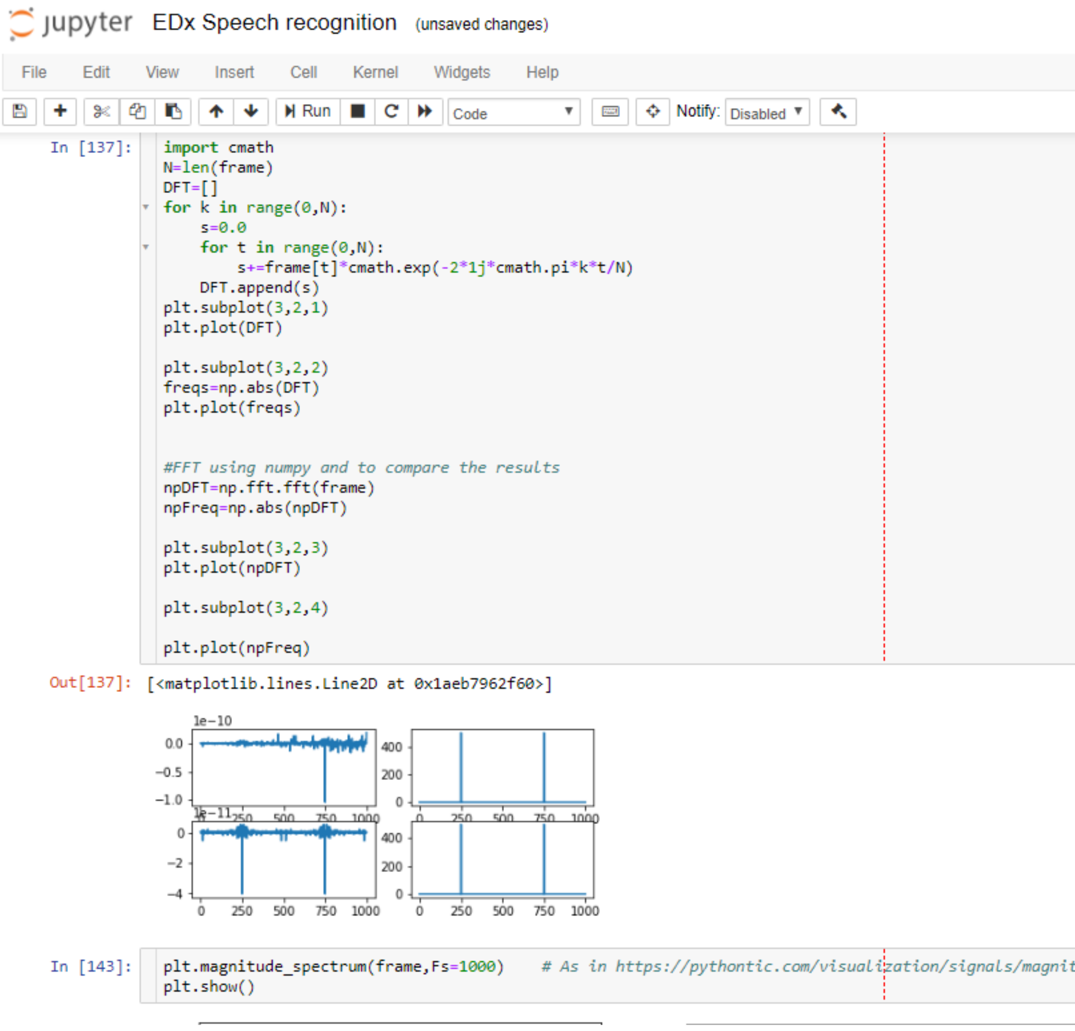
```
#print("Read {0} frames rom {1}".format(feat2.shape[1],
    feat_file))
#print("Per-element absolute error is
    {0}".format(np.linalg.norm(feat-feat2)/(feat2.shape[0]*feat2.shape[1]))
```



jupyter noise_remove (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled

```
    # frames *= 0.54 - 0.46 * numpy.cos((2 * numpy.pi * n) / (frame_length - 1))  # Explicit Implem
    mag_frames = numpy.absolute(numpy.fft.rfft(frames[i], NFFT))  # Magnitude of the FFT
    pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))  # Power Spectrum

In [ ]:

In [16]: plt.plot(fft(frames[3]))
         plt.axis([-20,200,-5,100])
         plt.show()

c:\python\python interpreter\lib\site-packages\numpy\core\numeric.py:531: ComplexWarning: Casting com
ds the imaginary part
  return array(a, dtype, copy=False, order=order)

In [17]: bin=

  File "<ipython-input-17-8d06b6b34769>", line 1
    bin=
       ^
SyntaxError: invalid syntax

In [ ]: plt.plot(numpy.fft.rfft(frames[3],512))

In [ ]: plt.plot(numpy.fft.fft(frames[3]))
```

Figure B.1: FFT of frame


jupyter
EDx Speech recognition (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled

```
In [137]:  import cmath
           N=len(frame)
           DFT=[]
           for k in range(0,N):
               s=0.0
               for t in range(0,N):
                   s+=frame[t]*cmath.exp(-2*1j*cmath.pi*k*t/N)
               DFT.append(s)
           plt.subplot(3,2,1)
           plt.plot(DFT)

           plt.subplot(3,2,2)
           freqs=np.abs(DFT)
           plt.plot(freqs)


           #FFT using numpy and to compare the results
           npDFT=np.fft.fft(frame)
           npFreq=np.abs(npDFT)

           plt.subplot(3,2,3)
           plt.plot(npDFT)

           plt.subplot(3,2,4)

           plt.plot(npFreq)

Out[137]: [<matplotlib.lines.Line2D at 0x1aeb7962f60>]
```

```
In [143]:  plt.magnitude_spectrum(frame,Fs=1000)    # As in https://pythontic.com/visualization/signals/magnit
           plt.show()
```

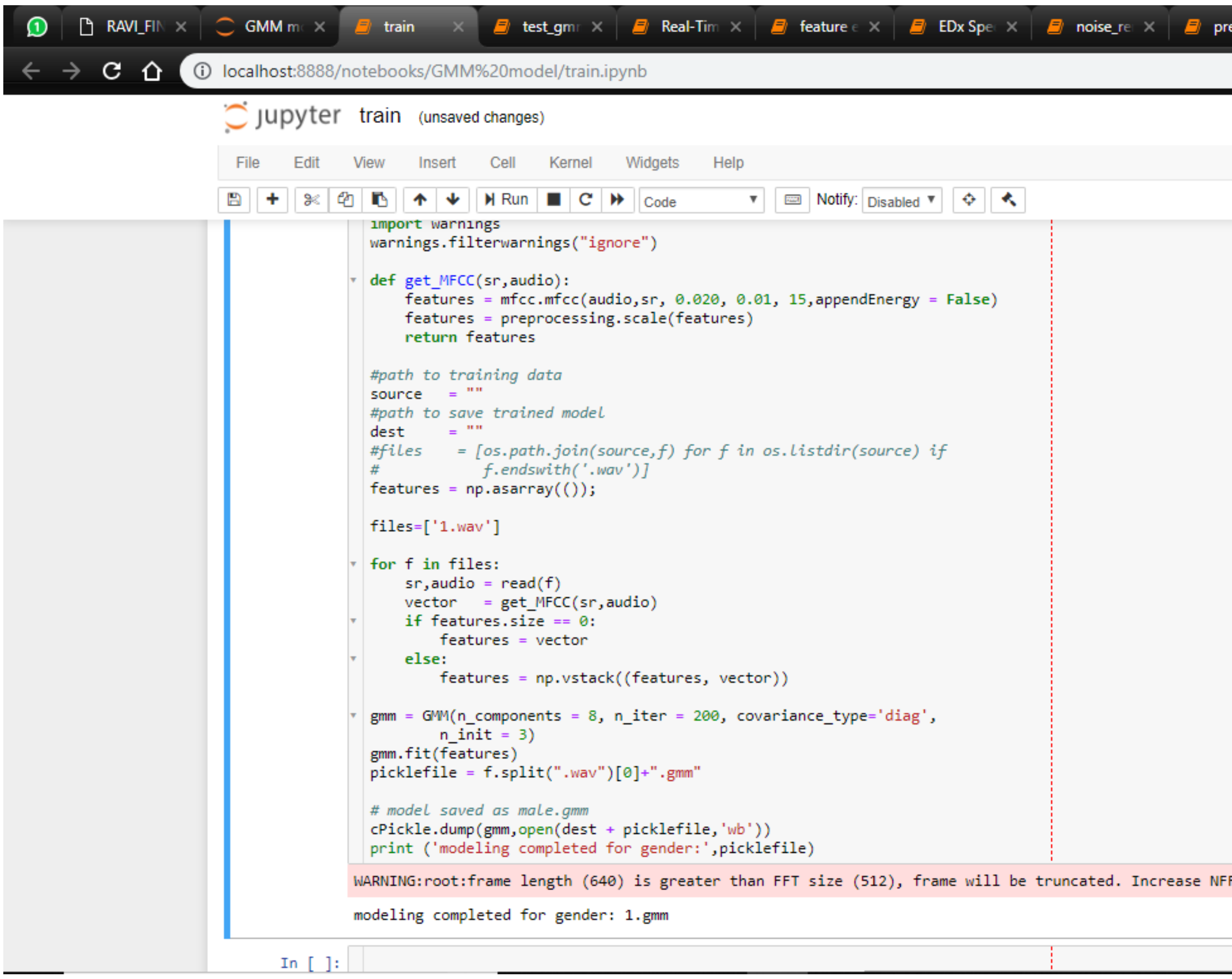
Figure B.2: DFT,frequecies, DFT using numpy, Frequencies using numpy

APPENDIX C

MODEL TRAINING

```
#MODEL TRAINING
#train_models.py

import os
import _pickle as cPickle
import numpy as np
from scipy.io.wavfile import read
from sklearn.mixture import GMM
import python_speech_features as mfcc
from sklearn import preprocessing
import warnings
warnings.filterwarnings("ignore")

def get_MFCC(sr,audio):
features = mfcc.mfcc(audio,sr, 0.020, 0.01, 15,appendEnergy =
   False)
features = preprocessing.scale(features)
return features

#path to training data
source = ""
#path to save trained model
dest   = ""
#files  = [os.path.join(source,f) for f in os.listdir(source)
   if
#          f.endswith('.wav')]
features = np.asarray(());

files=['1.wav']
```

```
for f in files:
sr,audio = read(f)
vector = get_MFCC(sr,audio)
if features.size == 0:
features = vector
else:
features = np.vstack((features, vector))

gmm = GMM(n_components = 8, n_iter = 200,
   covariance_type='diag',
n_init = 3)
gmm.fit(features)
picklefile = f.split(".wav")[0]+".gmm"

# model saved as male.gmm
cPickle.dump(gmm,open(dest + picklefile,'wb'))
print ('modeling completed for gender:',picklefile)
```


RAVI_FIN
GMM m
train
test_gmr
Real-Tim
feature
EDx Spe
noise_re
pre
localhost:8888/notebooks/GMM%20model/train.ipynb
jupyter train (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled
In [ ]:

```
import warnings
warnings.filterwarnings("ignore")

def get_MFCC(sr,audio):
    features = mfcc.mfcc(audio,sr, 0.020, 0.01, 15,appendEnergy = False)
    features = preprocessing.scale(features)
    return features

#path to training data
source   = ""
#path to save trained model
dest     = ""
#files     = [os.path.join(source,f) for f in os.listdir(source) if
#             f.endswith('.wav')]
features = np.asarray(());

files=['1.wav']

for f in files:
    sr,audio = read(f)
    vector   = get_MFCC(sr,audio)
    if features.size == 0:
        features = vector
    else:
        features = np.vstack((features, vector))

gmm = GMM(n_components = 8, n_iter = 200, covariance_type='diag',
        n_init = 3)
gmm.fit(features)
picklefile = f.split(".wav")[0]+".gmm"

# model saved as male.gmm
cPickle.dump(gmm,open(dest + picklefile,'wb'))
print ('modeling completed for gender:',picklefile)
```

```
WARNING:root:frame length (640) is greater than FFT size (512), frame will be truncated. Increase NFF
```

```
modeling completed for gender: 1.gmm
```

Figure C.1: GMM training

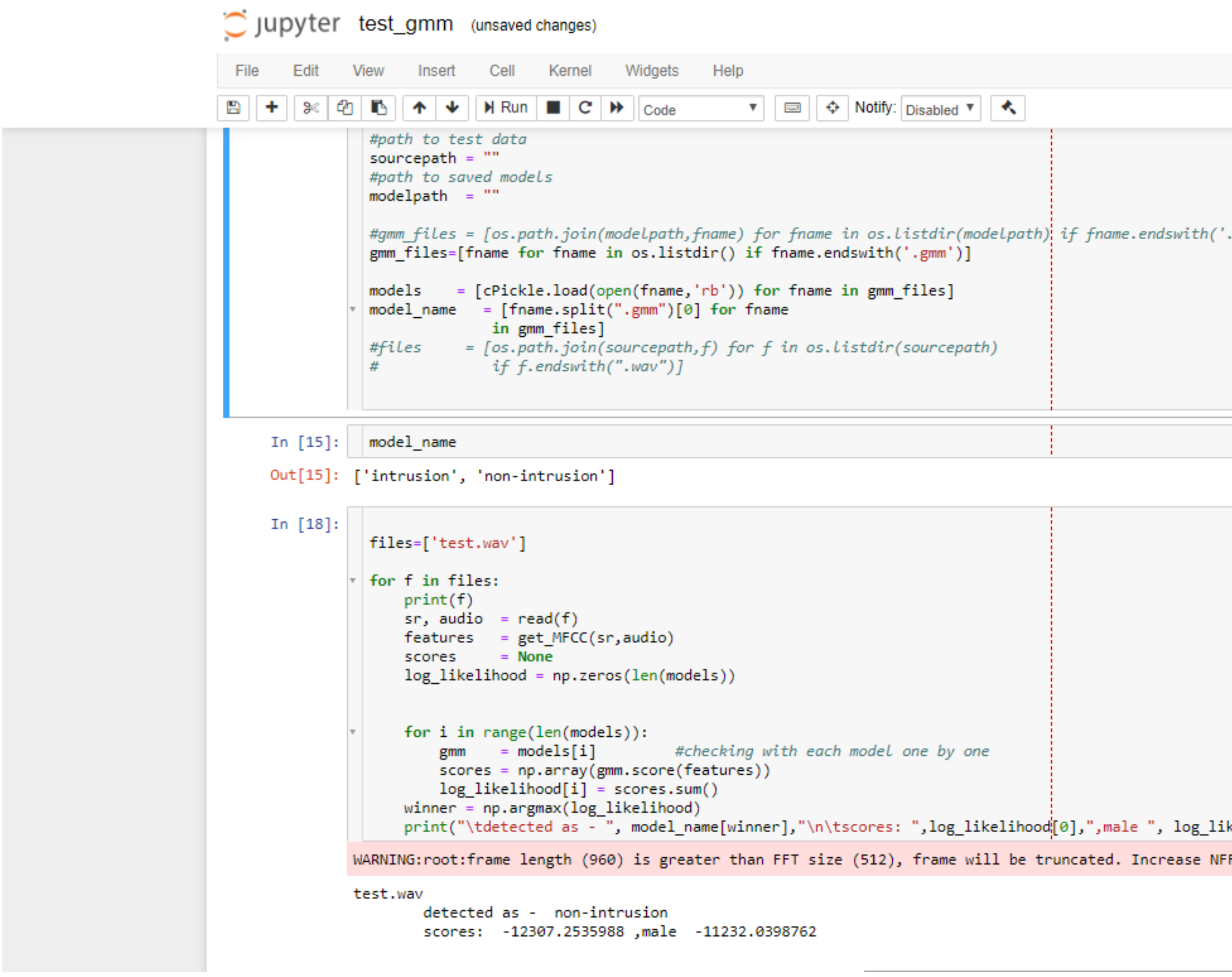
APPENDIX D

TESTING

```
#TESTING

#test_intrusion.py
import os
import pickle as cPickle
import numpy as np
from scipy.io.wavfile import read
import python_speech_features as mfcc
from sklearn import preprocessing
import warnings
warnings.filterwarnings("ignore")
def get_MFCC(sr,audio):
features = mfcc.mfcc(audio,sr, 0.020, 0.01, 15,appendEnergy =
   False)
feat   = np.asarray(())
for i in range(features.shape[0]):
temp = features[i,:]
if np.isnan(np.min(temp)):
continue
else:
if feat.size == 0:
feat = temp
else:
feat = np.vstack((feat, temp))
features = feat;
features = preprocessing.scale(features)
return features

#path to test data
```

```
sourcepath = ""
#path to saved models
modelpath = ""

#gmm_files = [os.path.join(modelpath,fname) for fname in
    os.listdir(modelpath) if fname.endswith('.gmm')]
gmm_files=[fname for fname in os.listdir() if
    fname.endswith('.gmm')]

models  = [cPickle.load(open(fname,'rb')) for fname in
    gmm_files]
model_name = [fname.split(".gmm")[0] for fname
in gmm_files]
#files   = [os.path.join(sourcepath,f) for f in
    os.listdir(sourcepath)
#          if f.endswith(".wav")]


model_name


files=['test.wav']

for f in files:
print(f)
sr, audio = read(f)
features = get_MFCC(sr,audio)
scores   = None
log_likelihood = np.zeros(len(models))


for i in range(len(models)):
gmm   = models[i]     #checking with each model one by one
scores = np.array(gmm.score(features))
```

```
log_likelihood[i] = scores.sum()
winner = np.argmax(log_likelihood)
print("\tdetected as - ", model_name[winner],"\n\tscores:
    ",log_likelihood[0],",male ", log_likelihood[1],"\n")
```


jupyter test_gmm (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Notify: Disabled

```
#path to test data
sourcepath = ""
#path to saved models
modelpath  = ""

#gmm_files = [os.path.join(modelpath,fname) for fname in os.listdir(modelpath) if fname.endswith('.
gmm_files=[fname for fname in os.listdir() if fname.endswith('.gmm')]

models     = [cPickle.load(open(fname,'rb')) for fname in gmm_files]
model_name    = [fname.split(".gmm")[0] for fname
              in gmm_files]
#files      = [os.path.join(sourcepath,f) for f in os.listdir(sourcepath)
#              if f.endswith(".wav")]
```

```
In [15]: model_name

Out[15]: ['intrusion', 'non-intrusion']
```

```
In [18]:
files=['test.wav']

for f in files:
    print(f)
    sr, audio  = read(f)
    features   = get_MFCC(sr,audio)
    scores     = None
    log_likelihood = np.zeros(len(models))


    for i in range(len(models)):
        gmm    = models[i]          #checking with each model one by one
        scores = np.array(gmm.score(features))
        log_likelihood[i] = scores.sum()
    winner = np.argmax(log_likelihood)
    print("\tdetected as - ", model_name[winner],"\n\tscores: ",log_likelihood[0],",male ", log_lik
```

```
WARNING:root:frame length (960) is greater than FFT size (512), frame will be truncated. Increase NFF

test.wav
        detected as -  non-intrusion
        scores:  -12307.2535988 ,male  -11232.0398762
```

Figure D.1: GMM output

APPENDIX E

OBJECT DETECTION

```
# USAGE
# python deep_learning_object_detection.py --image
    images/example_01.jpg \
# --prototxt MobileNetSSD_deploy.prototxt.txt --model
    MobileNetSSD_deploy.caffemodel

# import the necessary packages
import numpy as np
import argparse
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
help="path to input image")
ap.add_argument("-p", "--prototxt", required=True,
help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was
    trained to
# detect, then generate a set of bounding box colors for each
    class
CLASSES = ["background", "aeroplane", "bicycle", "bird",
    "boat",
"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
"dog", "horse", "motorbike", "person", "pottedplant", "sheep",
"sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# load the input image and construct an input blob for the
   image
# by resizing to a fixed 300x300 pixels and then normalizing it
# (note: normalization is done via the authors of the
   MobileNet SSD
# implementation)
image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)),
   0.007843, (300, 300), 127.5)

# pass the blob through the network and obtain the detections
   and
# predictions
print("[INFO] computing object detections...")
net.setInput(blob)
detections = net.forward()

# loop over the detections
for i in np.arange(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
   the
# prediction
confidence = detections[0, 0, i, 2]
```

```
# filter out weak detections by ensuring the `confidence` is
# greater than the minimum confidence
if confidence > args["confidence"]:
# extract the index of the class label from the `detections`,
# then compute the (x, y)-coordinates of the bounding box for
# the object
idx = int(detections[0, 0, i, 1])
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")

# display the prediction
label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
print("[INFO] {}".format(label))
cv2.rectangle(image, (startX, startY), (endX, endY),
COLORS[idx], 2)
y = startY - 15 if startY - 15 > 15 else startY + 15
cv2.putText(image, label, (startX, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

# show the output image
cv2.imshow("Output", image)
cv2.waitKey(0)
```

APPENDIX F

FACE DETECTION AND TRACKING

```
#[Navin_Kumar_Manaswi]_Deep_Learning_with_Applicati(z-lib.org).pdf
# Multi face tracking :
   https://www.guidodiepen.nl/2017/02/tracking-multiple-faces/

import cv2
import dlib
import os

saved_faces=[]



def extract_save_faces(img_,trackers):
#global trackers
global saved_faces
save_path=os.path.join(os.getcwd(),"Extracted Faces")
print("save_path: ",save_path)
for fid in trackers.keys():
if fid not in saved_faces:
tracked_position=trackers[fid].get_position()

t_x = int(tracked_position.left())
t_y = int(tracked_position.top())
t_w = int(tracked_position.width())
t_h = int(tracked_position.height())
print("Saving Face")
#print(img_[t_x:t_x+t_w][t_y:t_y+t_h])
#print("Saving to:
   ",os.path.join(save_path,'{0}.jpg'.format(len(saved_faces)+1)))
```

```
cv2.imwrite(os.path.join(save_path,'{0}.jpg'.format(len(os.listdir('Extract
   Faces'))+1)),img_[ t_y:t_y+t_h,t_x:t_x+t_w ])
saved_faces.append(fid)



def tracker_exist(x,y,w,h,trackers):
#global trackers
for fid in trackers.keys():
tracked_position=trackers[fid].get_position()

t_x = int(tracked_position.left())
t_y = int(tracked_position.top())
t_w = int(tracked_position.width())
t_h = int(tracked_position.height())

t_center_x= t_x + 0.5*t_w
t_center_y= t_y+ 0.5*t_h

#check if the centerpoint of the face is within the
#rectangleof a tracker region. Also, the centerpoint
#of the tracker region must be within the region
#detected as a face. If both of these conditions hold
#we have a match
center_x= x+ 0.5*w
center_y= y+ 0.5*h

if (x<=t_center_x <= (x+w)) and (y <=t_center_y <=(y+h)) and
   (t_x <=center_x <= (t_x+t_w)) and (t_y <= center_y
   <=(t_y+t_h)) :
return True

return False
```

```
def delete_trackers(img,face_count,trackers):
#global trackers
#global face_count
fidsToDelete=[]
for fid in trackers.keys():
track_quality=trackers[fid].update(img)

if track_quality<9:
fidsToDelete.append(fid)

for fid in fidsToDelete:
print("Removing tracker " + str(fid) + " from list of
   trackers")
#trackers.pop(fid,None)
del trackers[fid]
#face_count-=1 # as decrease the count and then there may be
   duplicate entries
facecascade=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eyecascade=cv2.CascadeClassifier('haarcascade_eye.xml')

for video_name in os.listdir('Videos'):
print("Video Name: ",video_name)
vs=cv2.VideoCapture('Videos/'+video_name)
#vs=cv2.VideoCapture(0)
face_count=0
eye_count=0
trackers={}

while True:
rc,img=vs.read()
#img = cv2.resize(img, (0,0), fx=0.5, fy=0.5)
#print(img)
```

```
orig_img=img.copy()
'''for i in range(0,3):
rc,img=vs.read()
continue'''
#img = cv2.rotateImage(img, 90)
#print(img)
#cv2.imshow('Detector',img)
gray_img=cv2.cvtColor(img.copy(),cv2.COLOR_BGR2GRAY)

delete_trackers(img,face_count,trackers)
for fid in trackers.keys():
tracked_position = trackers[fid].get_position()

t_x = int(tracked_position.left())
t_y = int(tracked_position.top())
t_w = int(tracked_position.width())
t_h = int(tracked_position.height())

cv2.rectangle(img, (t_x, t_y),
(t_x + t_w , t_y + t_h),
(255,0,0) ,1)
cv2.putText(img,"Tracking
   {0}".format(fid),(t_x+5,t_y+5),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),1


faces=facecascade.detectMultiScale(gray_img,1.3,5)
eyes=eyecascade.detectMultiScale(gray_img,1.3,5)

x=0
y=0
w=0
h=0
max_area=0
```

```
for (_x,_y,_w,_h) in faces:
if _w*_h>max_area:
x=_x
y=_y
h=_h
w=_w
max_area=_w*_h
if not tracker_exist(x,y,h,w,trackers):
t=dlib.correlation_tracker()
t.start_track(img,dlib.rectangle(int(x-10),int(y-20),int(x+w+10),int(y+h+2
cv2.putText(img,"Detecting
    {0}".format(face_count),(x+5,y+5),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255
cv2.rectangle(img, (x-10,y-20), (x+w+10, y+h+20), (0,0,255),2)
trackers[face_count]=t
face_count+=1

x1=0
y1=0
w1=0
h1=0
max_area1=0

for (_x,_y,_w,_h) in eyes:
if _w*_h>max_area1:
x1=_x
y1=_y
h1=_h
w1=_w
max_area1=_w*_h

cv2.rectangle(img, (x1-5,y1-10), (x1+w1+5, y1+h1+10),
    (0,255,0),1)
```

```
cv2.imshow('Detector',img)
extract_save_faces(orig_img,trackers)
key=cv2.waitKey(1) & 0xFF

if key==ord('q'):
break


vs.release()
cv2.destroyAllWindows()
```

APPENDIX G

OBJECT DETECTION AND MOTION TRACKING

```
content...from keras.applications.mobilenet_v2 import
   MobileNetV2
from keras.preprocessing import image
from keras.applications.mobilenet_v2 import
   preprocess_input, decode_predictions
import numpy as np
import argparse
import cv2
from keras.models import model_from_json

arg=argparse.ArgumentParser()
arg.add_argument('-i','--image',help='Path to image to be
   classified',nargs='+')
args=vars(arg.parse_args())



#model = MobileNetV2(weights='imagenet')

# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("savedWeightsMobileNet.h5")
print("Loaded model from disk")
model=loaded_model
```

```
img_path = args.get('image',None)
img = image.load_img(img_path[0], target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class,
   description, probability)
# (one such list for each sample in the batch)
preds=decode_predictions(preds, top=3)[0]
print('Predicted:', preds[0][1])
# Predicted: [(u'n02504013', u'Indian_elephant',
   0.82658225), (u'n01871265', u'tusker', 0.1122357),
   (u'n02504458', u'African_elephant', 0.061040461)]
'''
model.save('MobilenetModel.h5')
jsonmodel=model.to_json()
model.save_weights('savedWeightsMobileNet.h5')

# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
json_file.write(model_json)
# serialize weights to HDF5
print("Saved model to disk")'''

test_img=cv2.imread(img_path[0])
cv2.putText(test_img, "{0} / {1} /
   {2}".format(preds[0][1],preds[1][1],preds[2][1]) ,
   (20,20) ,cv2.FONT_HERSHEY_SIMPLEX, 1 , (0,255,0) , 2)
cv2.imshow("Classification",test_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

APPENDIX H

ENHANCE RESOLUTION

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:



from keras.preprocessing import image
img=image.load_img('Input_resolution.jpg')
x=image.img_to_array(img)
x.shape
import numpy as np
x = np.expand_dims(x, axis=0)
x.shape





# In[ ]:





# In[ ]:
```

```
# In[ ]:




# In[20]:


#Reference Book:
   [Navin_Kumar_Manaswi]_Deep_Learning_with_Applicati(z-lib.org).pdf

from os import path
import cv2
import os
import random
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D,MaxPool2D,Flatten
from keras.layers.core import Dense,Dropout,Activation
from keras.optimizers import adam
from keras.utils import np_utils
from keras import backend as K
import matplotlib.pyplot as plt


original_data_path='D:\dataset\Image\General-100'



def load_small_img_dataset(path=os.getcwd()):
data=[]
for image in os.listdir(path):
if image.endswith(('.jpg','.jpeg','.png','bmp'),0,len(image)):
```

```
pixels=cv2.imread(os.path.join(path,image))
pixels=pixels[0:100,0:90] #first make all images of same size
   using crop
data.append(pixels)

return data

imgs=load_small_img_dataset(original_data_path)

# OR import keras dataset from keras.datasets import cifar10
#(x_train, y_train), (x_test, y_test) =
   cifar100.load_data(label_mode='fine')






#generate a random number between 0 and 1 and if it is less
   than 0.7 than the current image is in train set else fi
   >0.9 then test else dev set

#or use sklearn.model_selection import train_test_split

def test_train_dev_split(dataset,train=0.7,dev=0.2,test=0.1):
#make seed for exact results everything
#random.sort(dataset)
random.seed(2)
random.shuffle(dataset)
split1=int(train*len(dataset))
split2=int((train+dev)*len(dataset))

train_set=dataset[:split1]
dev_set=dataset[split1:split2]
test_set=dataset[split2:]
```

```
return train_set,dev_set,test_set


def create_input_out_sets(train_output,dev_output,test_output):
train_input=[]
dev_input=[]
test_input=[]
for img in train_output:
low_pixels=cv2.resize(img,None,fx=0.5,fy=0.5,interpolation=cv2.INTER_AREA)
train_input.append(cv2.resize(low_pixels,None,fx=2,fy=2,interpolation=cv2.I

for img in dev_output:
low_pixels=cv2.resize(img,None,fx=0.5,fy=0.5,interpolation=cv2.INTER_AREA)
dev_input.append(cv2.resize(low_pixels,None,fx=2,fy=2,interpolation=cv2.INT

for img in test_output:
low_pixels=cv2.resize(img,None,fx=0.5,fy=0.5,interpolation=cv2.INTER_AREA)
test_input.append(cv2.resize(low_pixels,None,fx=2,fy=2,interpolation=cv2.IN



return
   np.array(train_input),np.array(train_output),np.array(dev_input),np.arra

#check wheather the imgs were correctly formed
train,dev,test=test_train_dev_split(imgs)
print("[INFO] Splitting done....")

train_in,train_out,dev_in,dev_out,test_in,test_out=create_input_out_sets(tr
print("[INFO] Input and output images created and sorted in
   datasets")



cv2.imshow("Original Img",train_out[1])
```

```
cv2.imshow("Low resolution",train_in[1])
cv2.waitKey(0)
cv2.destroyAllWindows()

print("[INFO] Shape of image",test_in[1].shape)


# In[5]:


#pre-Process Images :

'''
Both TensorFlow and Theano expects a 4 dimensional tensor as
    input.
But where TensorFlow expects the 'channels' dimension as the
    last dimension
(index 3, where the first is index 0) of the tensor âĂŞ i.e.
    tensor with shape (samples, rows, cols, channels) âĂŞ
Theano will expect 'channels' at the second dimension (index
    1) âĂŞ
i.e. tensor with shape (samples, channels, rows, cols). '''

# Keras Format:: [samples][width][height][channels]
# OpenCV format:: rows, columns and channels i.e.
    [height][width][channels]

# Current format:: [samples][height][width][channels] ->>>>>>
    [samples][width][height][channels]


'''# Reshape input data.
train_in=train_in.reshape(train_in.shape[0],90,100,3)
train_out=train_out.reshape(train_out.shape[0],90,100,3)
```

```
dev_in=dev_in.reshape(dev_in.shape[0],90,100,3)
dev_out=dev_out.reshape(dev_out.shape[0],90,100,3)
test_in=test_in.reshape(test_in.shape[0],90,100,3)
test_out=test_out.reshape(test_out.shape[0],90,100,3)
'''

# to convert our data type to float32 and normalize our
   database
train_in=train_in.astype('float32')
dev_in=dev_in.astype('float32')
test_in=test_in.astype('float32')
print(train_in.shape)
print(test_in.shape)


# Z-scoring or Gaussian Normalization
train_in=train_in - np.mean(train_in) / train_in.std()
dev_in=dev_in - np.mean(dev_in) / dev_in.std()
test_in=test_in - np.mean(test_in) / test_in.std()


# In[6]:


'''
Image Super-Resolution Using Deep
Convolutional Networks
Chao Dong[2015]
'''

# Define the keras DNN model
model =Sequential()
model.add(Conv2D(64,(9,9),input_shape=(100,90,3),activation='relu',padding=
#model.add(MaxPool2D(pool_size=(2,2)))
```

```
model.add(Dropout(0.3))
model.add(Conv2D(32,(1,1),activation='relu',padding='same'))
#model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))
model.add(Conv2D(3,(5,5),activation='relu',padding='same'))
#model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))


print("Models' output Shape: ",model.output_shape)


# In[ ]:




# In[12]:



def psnr(y_true, y_pred):
'''assert y_true.shape == y_pred.shape, "Cannot calculate
   PSNR. Input shapes not same."          " y_true shape = %s,
   y_pred shape = %s" % (str(y_true.shape),
str(y_pred.shape))
'''
"""
PSNR is Peek Signal to Noise Ratio, which is similar to mean
   squared error.
It can be calculated as
PSNR = 20 * log10(MAXp) - 10 * log10(MSE)
```

```
When providing an unscaled input, MAXp = 255. Therefore 20 *
   log10(255)== 48.1308036087.
However, since we are scaling our input, MAXp = 1. Therefore
   20 * log10(1) = 0.
Thus we remove that component completely and only compute the
   remaining MSE component.
"""
return -10. * K.log(K.mean(K.square(y_pred - y_true))) /
   K.log(10.)



#compile the model algong with adam optimiser along with
   PSNR/SSIM loss metric
model.compile(optimizer=adam(0.01),metrics=[psnr],loss='mse')
model.fit(train_in,train_out,batch_size=10,nb_epoch=50,validation_data=(dev
'''#loading saved weights
modelWts=model.load_weights('savedWeightsCNN.h5')
'''

#evaluate the model
score=model.evaluate(test_in,test_out)
print("[INFO] MSE:{0} PSNRLoss:{1}".format(score[0],score[1]))



# In[19]:



#print("TEST image shape: ",test_in[0].shape)

# prediction

#unknown test data
#cv2.imshow("Original Img",test_original_resolution[0])
```

```
#cv2.imshow("Low resolution",test_imgs[0])
pred_image=model.predict(test_in[4:5])
'''cv2.imshow("Peredicted resolution",pred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()'''


print("Peredicted resolution shape :",pred_image[0])

'''#save img
plt.subplot(221)
plt.imshow(test_in[2])

plt.subplot(222)
plt.imshow(test_out[2])

plt.subplot(223)
'''
plt.imshow(pred_image[0])




cv2.imwrite("Original_Img4.jpg",test_out[4])
cv2.imwrite("Input_resolution4.jpg",test_in[4])
cv2.imwrite("Test_Output4.jpg",pred_image[0])



# In[49]:



'''for i in range(0,len(test_in)):
cv2.imwrite("{0}.png".format(i),test_in[i])'''
```

```
# In[14]:




#Save the model
model.save('resoluteitmodelCNN_F.h5')
jsonmodel=model.to_json()
model.save_weights('savedWeightsCNN_F.h5')


print(model.summary())

#loading saved weights
#modelWts=model.load_weights('savedWeightsCNN.h5')
#model.get_weights()
#model.get_config()


# In[ ]:
```

APPENDIX I

FACE RECOGNITION

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



#https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8

from keras import applications
import cv2
import re
import os
import random
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense,
   GlobalAveragePooling2D
from keras import backend as k
from keras.callbacks import ModelCheckpoint,
   LearningRateScheduler, TensorBoard, EarlyStopping
import numpy as np
from keras.utils import to_categorical



# In[2]:



img_width=256
```

```
img_height=256


def load_small_img_dataset(path=os.getcwd()):
data=[]
labels=[]
for image in os.listdir(path):
if image.endswith(('.jpg','.jpeg','.png','bmp'),0,len(image)):
label=re.findall('^(.*)_.*',image)[0]
labels.append(label)
pixels=cv2.imread(os.path.join(path,image))
pixels=cv2.resize(pixels,(256,256),interpolation=cv2.INTER_CUBIC)
#first make all images of same size using crop
data.append(pixels)

return data,labels

x,y=load_small_img_dataset('D:\projects\ATM public
   safety\second review\Face Recognition\Face_Dataset')




# In[3]:




persons=set(y)
num_persons=len(set(y))
categorical_mapping={}

#convert into one hot encoding
for i,name in enumerate(persons):
#print(i,name)
categorical_mapping[name]=i
```

```
output_d=[]

for i_ in y:
i_=categorical_mapping[i_]
output_d.append(i_)
output_d=to_categorical(output_d)



# In[4]:



def
   test_train_dev_split(input_data,output_data,train=0.7,dev=0.2,test=0.1)
#make seed for exact results everything
#random.sort(dataset)
random.seed(2)
random.shuffle(input_data)
random.shuffle(output_data)
split1=int(train*len(input_data))
split2=int((train+dev)*len(input_data))
train_input=input_data[:split1]
dev_input=input_data[split1:split2]
test_input=input_data[split2:]



train_output=output_data[:split1]
dev_output=output_data[split1:split2]
test_output=output_data[split2:]

return
   np.array(train_input),np.array(train_output),np.array(dev_input),np.arra
```

```
# In[5]:


train_input,train_output,dev_input,dev_output,test_input,test_output=test_t


# In[6]:


model = applications.VGG19(weights = "imagenet",
   include_top=False, input_shape = (img_width, img_height, 3))


# In[7]:


train_output.shape


# In[8]:


model.summary()


# In[10]:


for layer in model.layers[:5]:
layer.trainable = False


# In[16]:
```

```
#Adding custom Layers
x = model.output
x = Flatten()(x)

x = Dense(512, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(216, activation="relu")(x)
x = Dropout(0.3)(x)
predictions = Dense(num_persons, activation="softmax")(x)

# creating the final model
model_final = Model(inputs = model.input, outputs =
   predictions)

# compile the model
model_final.compile(optimizer = optimizers.SGD(lr=0.001,
   momentum=0.9),loss = "categorical_crossentropy",
   metrics=["accuracy"])



# In[17]:



model_final.output_shape



# In[18]:



train_output.shape



# In[19]:
```

```
# Save the model according to the conditions
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc',
    verbose=1, save_best_only=True, save_weights_only=False,
    mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0,
    patience=10, verbose=1, mode='auto')

'''
# Train the model
model_final.fit_generator(
train_generator,
samples_per_epoch = nb_train_samples,
epochs = epochs,
validation_data = validation_generator,
nb_val_samples = nb_validation_samples,
callbacks = [checkpoint, early])
'''

model_final.fit(train_input,train_output,batch_size=5,nb_epoch=10,validatic
    = [checkpoint, early])


# In[20]:


score=model_final.evaluate(test_input,test_output)
#print(score)
print("[INFO] Loss:{0} Accuracy:{1}".format(score[0],score[1]))


# In[21]:
```

```
test_predictions=model_final.predict(test_input)


# In[22]:


oneHot2Name={}

for i in categorical_mapping.keys():
oneHot2Name[categorical_mapping[i]]=i
oneHot2Name


# In[26]:


c=0
for i in test_predictions:
cv2.imshow(str(oneHot2Name[np.argmax(i)])+str(c)+'.jpg',test_input[c])
c+=1

cv2.waitKey(0)
cv2.destroyAllWindows()


# In[144]:


get_ipython().run_line_magic('reset', '')


# In[162]:
```

```
1/14


# In[ ]:
```

REFERENCES

1. Foggia, P., Petkov, N., Saggese, A., Strisciuglio, N., and Vento, M. (2015). “Reliable detection of audio events in highly noisy environments.” *Pattern Recognition Letters*, 65, 22 – 28.

2. Gao, L., Guo, Z., Zhang, H., Xu, X., and Shen, H. T. (2017). “Video captioning with attention-based lstm and semantic consistency.” *IEEE Transactions on Multimedia*, 19(9), 2045–2055.

3. Kotus, J., Łopatka, K., Czyżewski, A., and Bogdanis, G. (2016). “Processing of acoustical data in a multimodal bank operating room surveillance system.” *Multimedia Tools and Applications*, 75(17), 10787–10805.

4. LudeÃśa-Choez, J. and Gallardo-AntolÃ■n, A. (2015). “Feature extraction based on the high-pass filtering of audio signals for acoustic event classification.” *Computer Speech and Language*, 30(1), 32 – 42.

5. Nabi, W., Nasr, M. B., Aloui, N., and Cherif, A. (2018). “A dual-channel noise reduction algorithm based on the coherence function and the bionic wavelet.” *Applied Acoustics*, 131, 186 – 191.

6. Redmon, J. and Farhadi, A. (2016). “YOLO9000: better, faster, stronger.” *CoRR*, abs/1612.08242.

7. Sharan, R. V. and Moir, T. J. (2017). “Robust acoustic event classification using deep neural networks.” *Information Sciences*, 396, 24 – 32.

8. Souli, S. and Lachiri, Z. (2018). “Audio sounds classification using scattering features and support vectors machines for medical surveillance.” *Applied Acoustics*, 130, 270 – 282.

9. Xu, H., Chen, T., Gao, D., Wang, Y., Li, K., Goel, N., Carmiel, Y., Povey, D., and Khudanpur, S. (2018). “A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition.” *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5929–5933.