TOSHKENT SHAHRIDAGI INHA UNIVERSITETI
INHA UNIVERSITY IN TASHKENT

Signals & Systems (ICE-3110),

Spring 2019

Term-Project Report

Voice recognition via Fourier Transform

Abbosjon Kudratov	Amalbek Jalilov	Umarjon Rajabov	Azizbek Samatov
u1610001 Section 001	u1610023 Section 001	u1610247 Section 001	u1610042 Section 003

08/05/2019

Contents

1 Introduction

Voice recognition is becoming more and more important in everyday life of people. In real world there are many ways of implementing those algorithms but as we learn Fourier Transform why not to try. So our team's main goal is to implement simple algorithm based on Fourier Transform to recognize "YES" or "NO" words.For signals, this is typically done via the cross-correlation function (of two signals) which is very similar to convolution. As such, it can be mathematically done via the FFT, which is specifically designed to be efficient. Once you take the correlation function, you can decide what threshold you want to be a "match".

In mathematics, the discrete Fourier transform (DFT) is a specific kind of Fourier transform, used in Fourier analysis. It transforms one function into another, which is called the frequency domain representation, or simply the DFT, of the original function (which is often a function in the time domain). But the DFT requires an input function that is discrete and whose non-zero values have a limited (finite) duration. Such inputs are often created by sampling a continuous function, like a person's voice. And unlike the discrete-time Fourier transform (DTFT), it only evaluates enough frequency components to reconstruct the finite segment that was analyzed. Its inverse transform cannot reproduce the entire time domain, unless the input happens to be periodic (forever) [1].

Therefore it is often said that the DFT is a transform for Fourier analysis of finite-domain discrete-time functions. The sinusoidal basis functions of the decomposition have the same properties.

Since FFT algorithms are so commonly employed to compute the DFT, the two terms are often used interchangeably in colloquial settings, although there is a clear distinction: "DFT" refers to a mathematical transformation, regardless of how it is computed, while "FFT" refers to any one of several efficient algorithms for the DFT.

2 Fast Fourier Transform and Voice Recognition

Fourier Analysis is the technique that produces something called the Fourier Transform, which contains the information in the spectrum [2]. But how does it work? – By using the inner product!

For speech signals analysis the ordinary transform used called the Fast Fourier transform (FFT). FFT provides the standard representation of a speech signal in the frequency domain. While Short Fourier transform be able to hold time frequency changes. The drawback of FFT that it is not appropriate for the signals whose frequencies are time varying hence in case of FFT is assumes that the signals are stationary in nature [3]. The FFT allows working in frequency domain and therefore using the frequency spectrum of the speech signal as a substitute of waveform. Frequency domain provides more information about the speech signal and hence can be more efficient to distinguish between speakers. For speaker recognition some techniques use the vice signal acquire directly by the sampling phase and some techniques use transformed form of the speech signal [3]. When a speech signal represented in time domain then it gives a little information regarding speech signal properties hence suitable transformation of speech signal is an essential problem. For this purpose generally Fourier or wavelet transform are used. Speech signal/audio processing techniques begin by converting the raw speech into a sequence of acoustic feature vectors carrying features of the signal. And this is known as pre-processing i.e. feature extraction is completed here and is also called front-end processing.

–THE DISCRETE FOURIER TRANSFORM

The FFT is a fast, O[NlogN] algorithm to compute the Discrete Fourier Transform (DFT), which naively is an O[N2] computation [4]. The DFT, like the more familiar continuous version of the Fourier transform, has a forward and inverse form which are defined as follows:

Forward Discrete Fourier Transform (DFT):

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\, 2\pi\, k\, n\, /\, N}$$

Inverse Discrete Fourier Transform (IDFT):

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i\, 2\pi\, k\, n\, /\, N}$$

–WHY FFT?

FFT algorithms are faster ways of doing DFT. It is a family of algorithms and not a single algorithm. How it becomes faster can be explained based on the heart of the algorithm: Divide And Conquer [5]. So rather than working with big size Signals, we divide our signal into smaller ones, and perform DFT of these smaller signals. At the end we add all the smaller DFT to get actual DFT of the big signal. This gives great benefit asymptotically. So for large values of N, we save a lot!

When N is a power of 2, the Cooley-Tukey FFT divides the N-DFT problem to two N2 DFT problems. Using the same idea, one may further divide the two N2 DFT problems to four separate N4-DFT problems, and so on... At the end, you have log2N steps with O(N) operations to be performed at each step.

–COMPUTING THE DISCRETE FOURIER TRANSFORM

For simplicity, we'll concern ourself only with the forward transform, as the inverse transform can be implemented in a very similar manner. Taking a look at the DFT expression above, we see that it is nothing more than a straightforward linear operation: a matrix-vector multiplication of $\vec{x}$,

$$\vec{X} = M \cdot \vec{x}$$

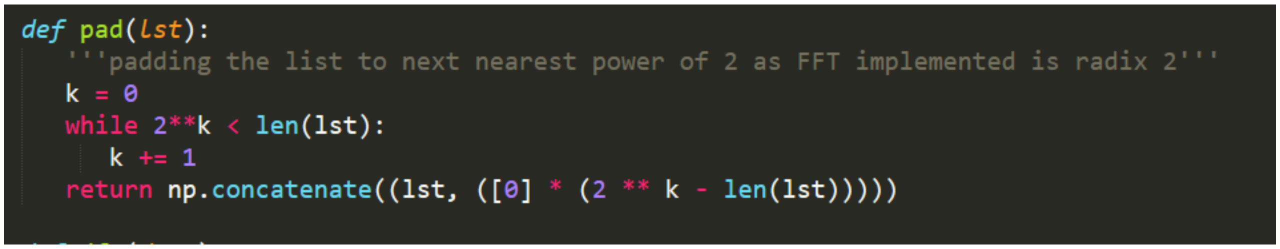
with the matrix M given by

$$M_{kn} = e^{-i\,2\pi\,k\,n\,/\,N}.$$

3 Implementation & Analysis

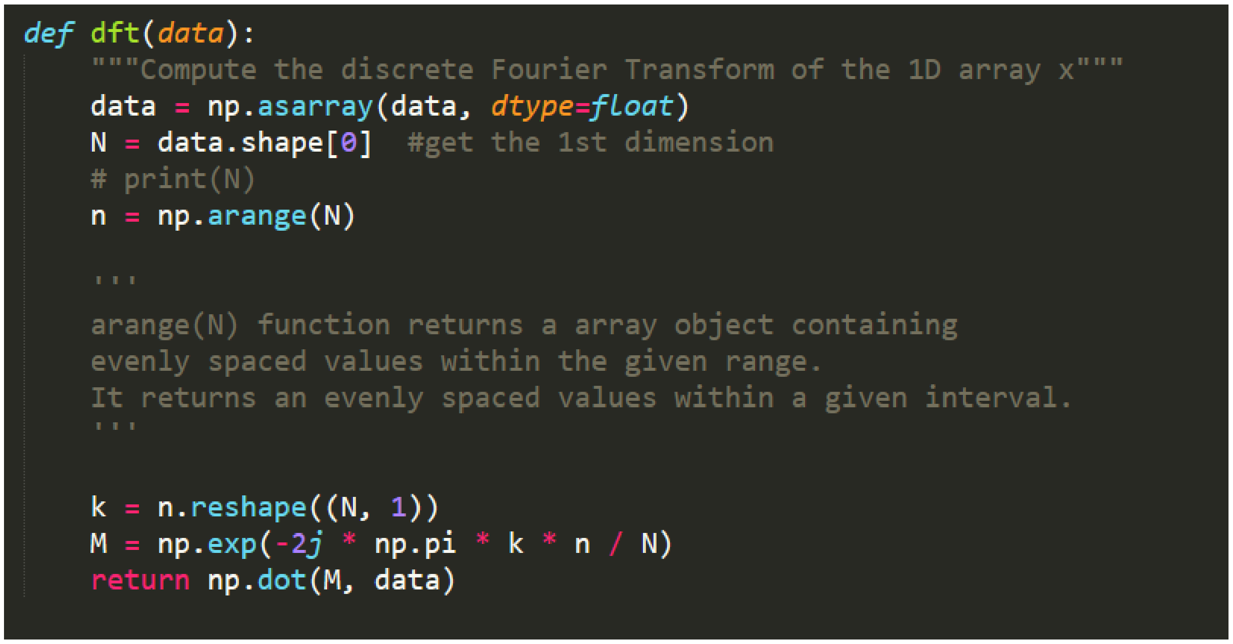
The implementation of the task was done in python environment and all source codes along with input files are available in the "src" folder attached to this report document.

We defined the following functions in the file "customFFT.py":



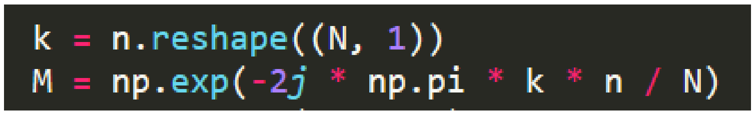
```
def pad(lst):
    '''padding the list to next nearest power of 2 as FFT implemented is radix 2'''
    k = 0
    while 2**k < len(lst):
        k += 1
    return np.concatenate((lst, ([0] * (2 ** k - len(lst)))))
```

Figure 1. Function PAD for padding the list to next nearest power of 2 as FFT implemented is radix 2



```
def dft(data):
    """Compute the discrete Fourier Transform of the 1D array x"""
    data = np.asarray(data, dtype=float)
    N = data.shape[0]  #get the 1st dimension
    # print(N)
    n = np.arange(N)

    '''
    arange(N) function returns a array object containing
    evenly spaced values within the given range.
    It returns an evenly spaced values within a given interval.
    '''

    k = n.reshape((N, 1))
    M = np.exp(-2j * np.pi * k * n / N)
    return np.dot(M, data)
```

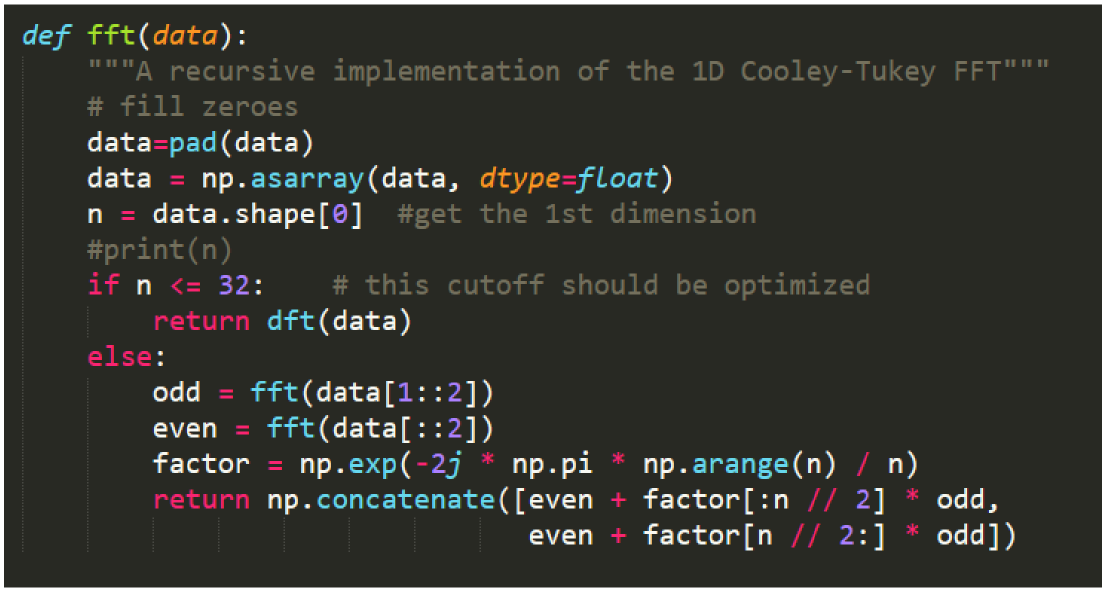
Figure 2. DFT Function to compute the discrete Fourier Transform of the 1D array x



```
k = n.reshape((N, 1))
M = np.exp(-2j * np.pi * k * n / N)
```

This code expressions specifies the implementation of the following transform part:

$$M_{kn} = e^{-i\,2\pi\,k\,n\,/\,N}.$$

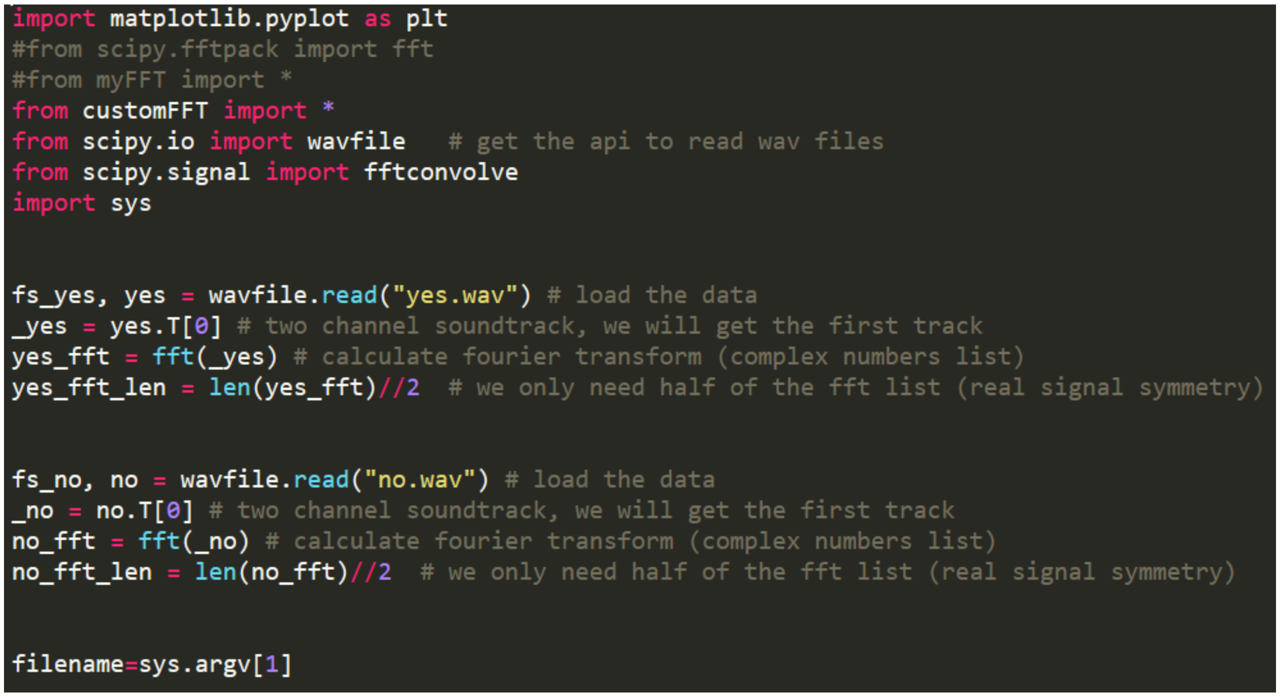


```python
def fft(data):
    """A recursive implementation of the 1D Cooley-Tukey FFT"""
    # fill zeroes
    data=pad(data)
    data = np.asarray(data, dtype=float)
    n = data.shape[0]  #get the 1st dimension
    #print(n)
    if n <= 32:    # this cutoff should be optimized
        return dft(data)
    else:
        odd = fft(data[1::2])
        even = fft(data[::2])
        factor = np.exp(-2j * np.pi * np.arange(n) / n)
        return np.concatenate([even + factor[:n // 2] * odd,
                               even + factor[n // 2:] * odd])
```

Figure 3. A recursive implementation of the 1D Cooley-Tukey FFT

We call the above function recursively for odd and even parts separately and at the end concatenate or combine results.

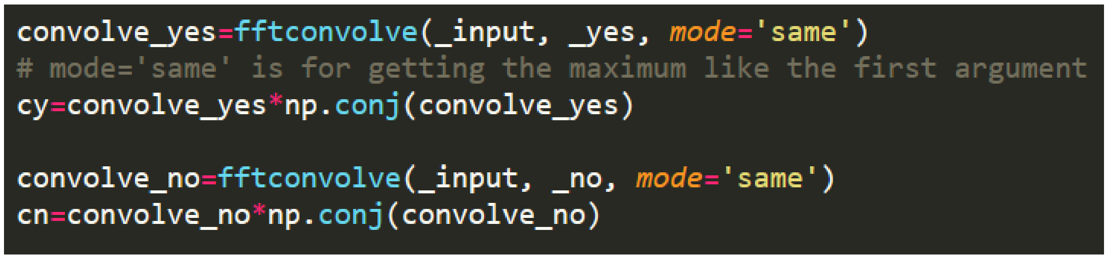
The input files are loaded via the following processes defined in the file “fft_plots.py”:



```python
import matplotlib.pyplot as plt
#from scipy.fftpack import fft
#from myFFT import *
from customFFT import *
from scipy.io import wavfile   # get the api to read wav files
from scipy.signal import fftconvolve
import sys


fs_yes, yes = wavfile.read("yes.wav") # load the data
_yes = yes.T[0] # two channel soundtrack, we will get the first track
yes_fft = fft(_yes) # calculate fourier transform (complex numbers list)
yes_fft_len = len(yes_fft)//2  # we only need half of the fft list (real signal symmetry)


fs_no, no = wavfile.read("no.wav") # load the data
_no = no.T[0] # two channel soundtrack, we will get the first track
no_fft = fft(_no) # calculate fourier transform (complex numbers list)
no_fft_len = len(no_fft)//2  # we only need half of the fft list (real signal symmetry)


filename=sys.argv[1]
```

Figure 4. The 2 files "yes.wav" and "no.wav" are taken as base files to compare with the input filename taken from command line argument

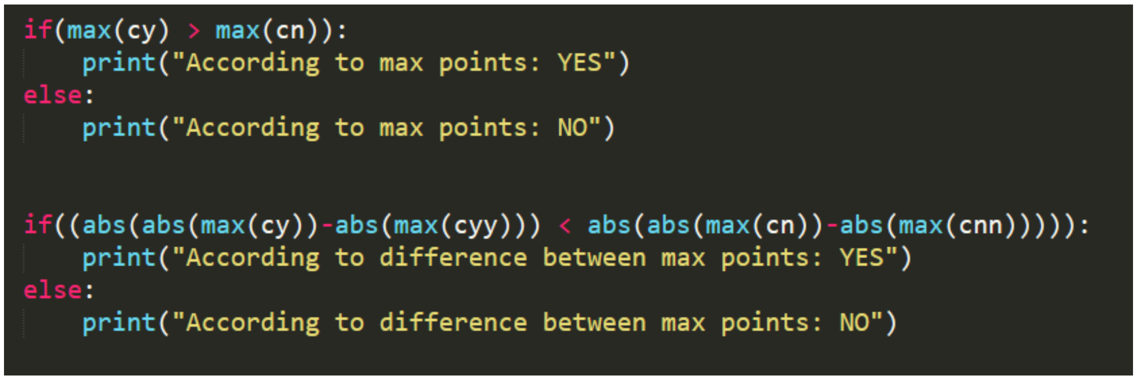
The next step is to find the convolution of FFT forms of input and base files in order to compare them in frequency domain. And to make it easier to compare we need to get rid of the imaginary parts, as we can only compare the real parts. So the conjugation is done like the following:



```
convolve_yes=fftconvolve(_input, _yes, mode='same')
# mode='same' is for getting the maximum like the first argument
cy=convolve_yes*np.conj(convolve_yes)

convolve_no=fftconvolve(_input, _no, mode='same')
cn=convolve_no*np.conj(convolve_no)
```

Figure 5. How to Convolute and conjugate in order to compare the input and base files

Then finally the decision making process compares the maximum points values and difference between them to finalize if the input file is recognized as voice “yes” or “no”:

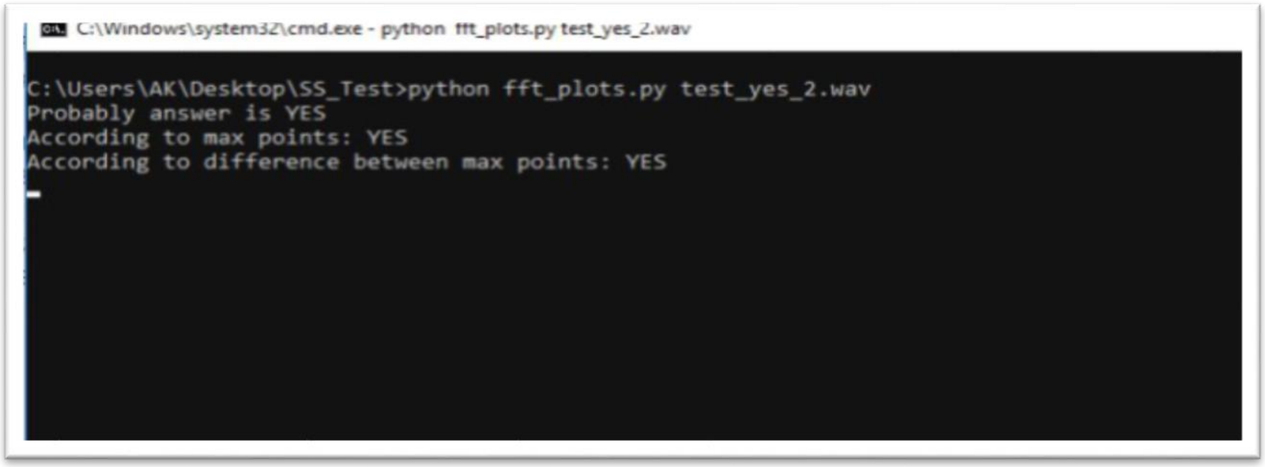


```
if(max(cy) > max(cn)):
    print("According to max points: YES")
else:
    print("According to max points: NO")


if((abs(abs(max(cy))-abs(max(cyy))) < abs(abs(max(cn))-abs(max(cnn))))):
    print("According to difference between max points: YES")
else:
    print("According to difference between max points: NO")
```

Figure 6. The comparisons of the results to make decision about the input file

4 Simulation Results and Discussions

*Note that you can find all the test input files in “src” folder attached to this project report file.


C:\Windows\system32\cmd.exe - python fft_plots.py test_yes_2.wav

```
C:\Users\AK\Desktop\SS_Test>python fft_plots.py test_yes_2.wav
Probably answer is YES
According to max points: YES
According to difference between max points: YES
```

Figure 7. Running the program with "test_yes_2.wav" as input file

Running the program with "test_yes_2.wav" file as input showed/recognized that it is a “YES”

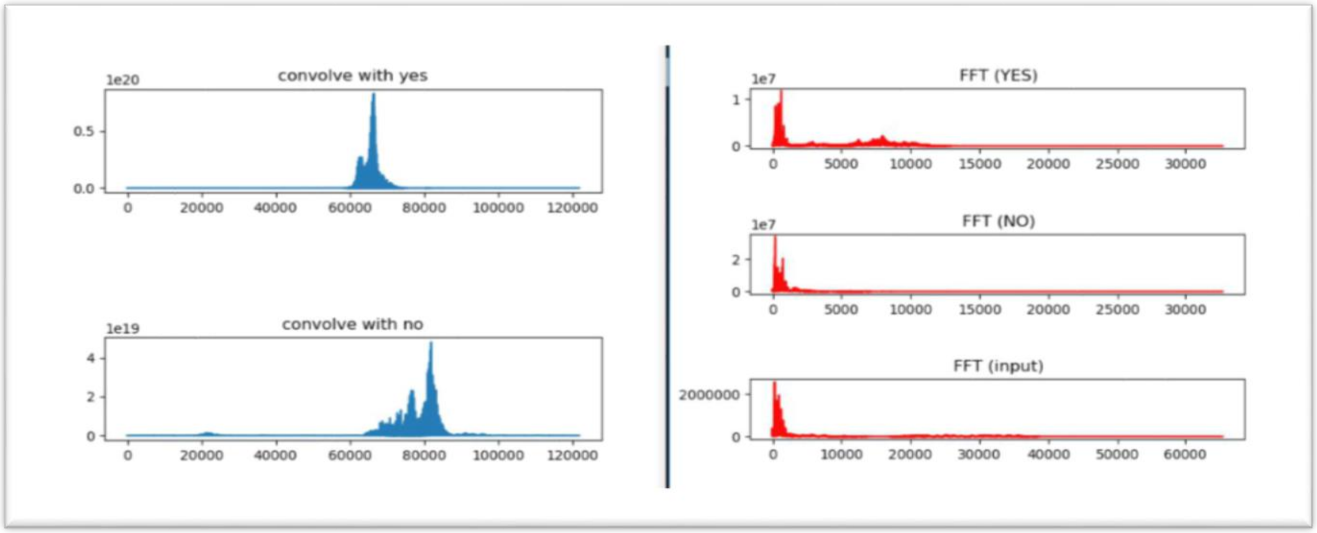
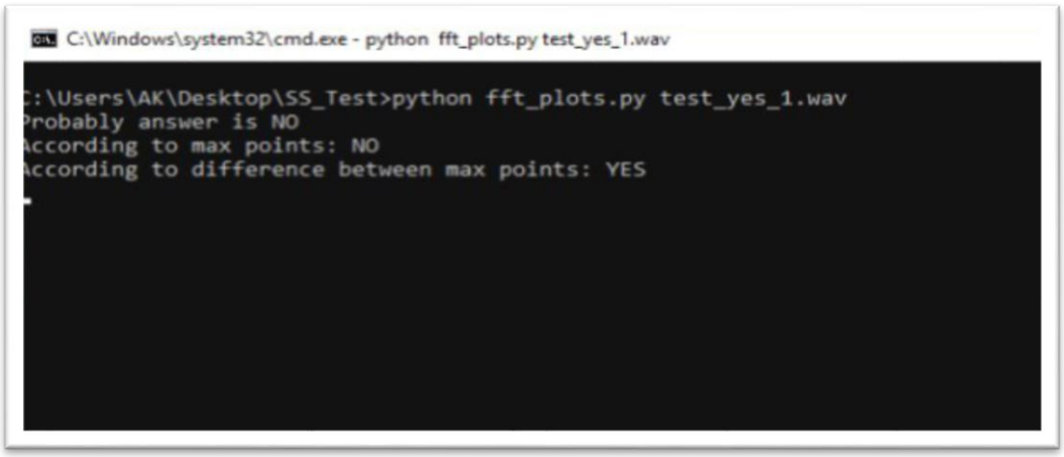


Figure 8. (from left to right) The convolution of input file("test_yes_2.wav") with base files,   The FFT forms of input and base files



```
C:\Windows\system32\cmd.exe - python  fft_plots.py test_yes_1.wav

:\Users\AK\Desktop\SS_Test>python fft_plots.py test_yes_1.wav
robably answer is NO
ccording to max points: NO
ccording to difference between max points: YES
```

Figure 9. Running the program with "test_yes_1.wav" as input file

Running the program with "test_yes_1.wav" file as input showed/recognized a different result and the reason for that is the noise which was present inside this wav file.

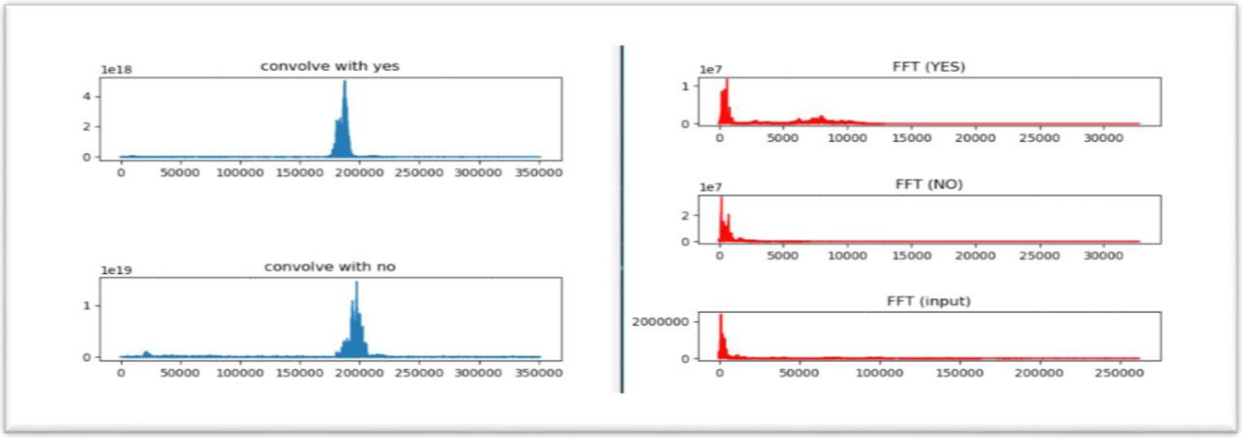
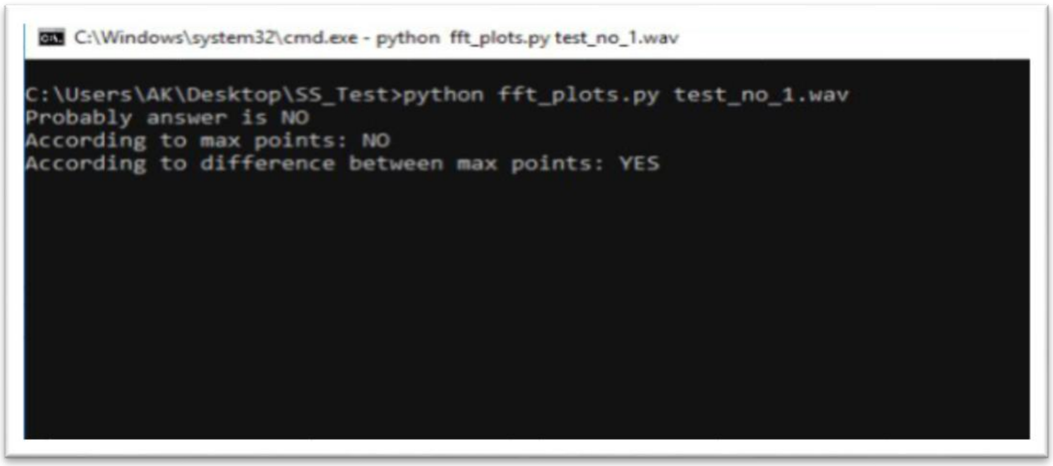


Figure 10. (from left to right) The convolution of input file("test_yes_1.wav") with base files, The FFT forms of input and base files



```
C:\Windows\system32\cmd.exe - python  fft_plots.py test_no_1.wav

C:\Users\AK\Desktop\SS_Test>python fft_plots.py test_no_1.wav
Probably answer is NO
According to max points: NO
According to difference between max points: YES
```

Figure 11 Running the program with "test_no_1.wav" as input file

Running the program with "test_no_1.wav" file as input showed/recognized that it is a “NO”

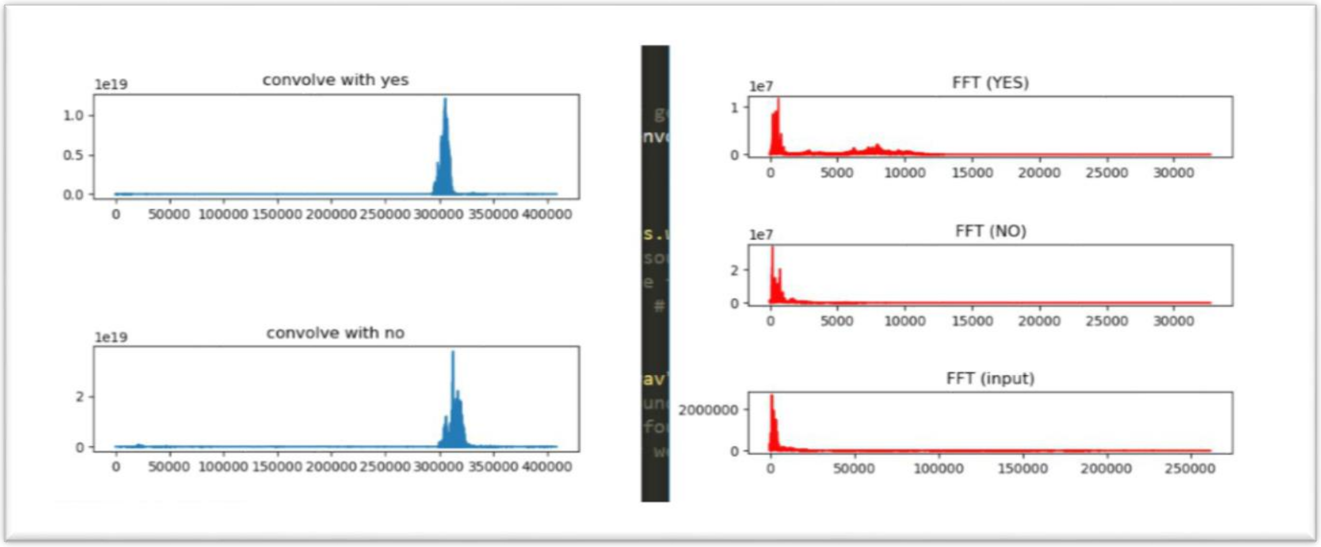


Figure 12. (from left to right) The convolution of input file("test_no_1.wav") with base files,   The FFT forms of input and base files

To conclude the results obtained from our tests, we have to mention that these different input wav audio files were recorded in different environment situations. And the errors are due to excessive noise and probably program/source code faults or bugs.

5 References

[1] http://paulbourke.net/miscellaneous/dft

[2] https://sail.usc.edu/~lgoldste/

[3] https://researchgate.net/publication/281843840_Speaker_Recognition_and_Fast_Fourier_Transform

[4] https://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/

[5] http://www.differencebetween.net/technology/difference-between-fft-and-dft/