

A study of classification methods to
identify sound signals of a washing machine

Chiharu Kazama
chiharu_k74@hotmail.com

Supervised by :
Claus Führer & Najmeh Abiri

April 28, 2019

Contents

| | |
|--|-----------|
| Acknowledgements | 3 |
| Populärvetenskaplig Sammanfattning på Svenska | 4 |
| 1 Introduction | 5 |
| 2 Fourier Series | 7 |
| 2.1 Sinusoid Functions | 7 |
| 2.2 Fourier Series | 9 |
| 2.3 Fourier Coefficients | 10 |
| 2.4 Euler's Formula | 11 |
| 2.5 Exponential Form | 12 |
| 2.6 Orthogonality | 12 |
| 3 Discrete Fourier Transform | 14 |
| 3.1 Fourier Transform | 14 |
| 3.2 The Dirac Delta Function | 17 |
| 3.3 The Comb Function | 18 |
| 3.4 Discrete Fourier Transform | 18 |
| 3.5 How to Interpret DFT Spectral Results | 20 |
| 3.6 Computation of DFT | 22 |
| 4 Cooley-Tukey FFT Algorithm | 25 |
| 4.1 Matrix Form | 25 |
| 4.2 Mathematical Expression | 27 |
| 5 Audio Classification | 30 |
| 5.1 Washing Machine | 30 |
| 5.2 Experiment Environment | 31 |
| 5.3 Sampling and Sound Editing | 33 |
| 5.3.1 Sampling & Time Domain Aliasing | 34 |
| 5.3.2 Convolution Theorem | 34 |
| 5.3.3 Frequency Domain Aliasing | 37 |
| 5.3.4 Sampling Theorem | 38 |

| | | |
|------------------|---|-----------|
| 5.4 | Spectral Analysis with Audacity | 38 |
| 5.5 | Short-Time Fourier Transform | 41 |
| 5.6 | Window Functions | 42 |
| 5.7 | Feature Extraction | 44 |
| 5.8 | Problems | 47 |
| 6 | To Machine Learning | 50 |
| 6.1 | Outline of Deep Neural Network | 51 |
| 6.2 | Keras | 53 |
| 6.3 | Environment | 54 |
| 6.4 | Deep Learning with Keras | 54 |
| 6.4.1 | Making the Dataset | 55 |
| 6.4.2 | Making the Neural Network Model | 57 |
| 6.4.3 | Training the Model | 59 |
| 6.4.4 | Model Evaluation | 60 |
| 6.5 | Random Search for Hyperparameters | 62 |
| 6.6 | Results | 65 |
| 7 | Conclusion | 66 |
| Appendix: | | |
| A | Dirichlet Condition | 68 |
| B | Python Code - STFT | 69 |
| C | Python Code - Making Dataset | 71 |
| D | Python Code - Random Search | 73 |

Acknowledgements

This thesis becomes a reality with the kind support and help of many individuals. I would like to first thank Professor Claus Führer for his encouragement throughout this project, as well as Najmeh Abiri for her great support. Your counsel always gave me lots of inspirations.

I also thank to James Hakim and Amanuel Workneh who kindly helped with my computer related queries. It is a pleasure to thank my friends, Xénia Edvinsson, Hilda Lidén and Sam Sarwat Hanna Gergis who gave me brilliant advise.

I will forever be thankful to Siobhán Correnty. This work would not have been possible without you. Meeting you is one of the best things that ever happened to me in my life.

Finally, my deep and sincere gratitude to my family for their understanding, support and love.

Populärvetenskaplig Sammanfattning på Svenska

Syftet med denna uppsats är att undersöka och klassificera ljud från en tvättmaskin. Ett tvättmaskinsprogram har olika faser, t ex vattenfyllning, tvättning, centrifugering, etc. Varje fas har ett eget ljud, därför kan vi anta vilken av faserna det är från ljudet, även om vi inte ser tvättmaskinen. När ljudet 'a' skiljer sig från ljudet 'b', ska vågformen av 'a' vara annorlunda jämfört med vågformen av 'b'. Detta innebär att de frekvenser som utgör vågformerna är annorlunda.

Formeln för att få ut frekvenskomponenter i en vågform är Fourier-transformen. Fourier-transformen delar upp en vågform i frekvenskomponenter genom en kombination av enkla sinusvågor. Vi kan även rekonstruera den ursprungliga vågformen från dessa komponenter.

Denna uppsats består av tre delar. Den första studien fokuserade på Fourier-transformerna. Speciellt den diskreta Fourier-transformen (DFT) och den snabba Fourier-transformen (FFT) eftersom ljudsignalen analyseras av en dator som hanterar endast diskreta värden. Den andra, vi studerar ljud-extraktions metoder och diskuterar problemen. I sista delen, uppvisas ljudklassificering med en maskininlärningsmetod, kallad "supervised learning" med djupa neurala nätverk. En av hyperparameteroptimeringar, "random search" förklaras och används för klassificering av flera klasser.

Chapter 1

Introduction

The purpose of this thesis is to study the classifying sounds of a washing machine. A washing machine program has different stages e.g., water-filling, washing, spinning, etc. Each stage makes a different sound, therefore we can presume the stage from the sound, even though we don't see the washing machine. Frequencies are the building blocks of sounds. When a certain sound 'a' is different from another sound called 'b', then the waveform of 'a' should be different from 'b'. This implies that the frequencies which make up the waveforms are also different. If we could extract the features of each stage by their frequencies, then it may be possible to determine the stage of the washing program from a remote place, without seeing or hearing the washing machine.

Examining frequencies in the frequency domain is simply a different way of looking at a signal, compared to the traditional way of in the time domain. The advantages of using frequency representation of a signal are first, the frequency domain makes it simpler and clearer to see the information of the waveform compared to the time domain. Besides frequency representation needs a smaller space of storage. Second, the signal often needs to be processed, for example, through noise removal. For removing noise among synthesized sounds, we first find the frequency of the noise and then remove the frequency in the frequency domain. Then you can hear the sound without the noise. In many cases, modifications like this are easier to do in the frequency domain than in the time domain.

The tool for getting out frequency components of a waveform is the Fourier transform. The Fourier transform decomposes a waveform into frequency components by a combination of simple sinusoidal functions. We can even reconstruct the original waveform from these decomposed components. To put it simply, if we put a fruit juice into a Fourier transform machine then we get out information about all the ingredients which make up the fruit juice and

the quantities. If we put all the ingredients back into the Fourier transform machine then we can reproduce the fruit juice. The Fourier transform is a wonderful tool!

An input signal of a Fourier transform can be either continuous or discrete and it can be either periodic or aperiodic. The combinations of these two features creates the four types of Fourier transforms, described below and illustrated in Figure 1.1.





| Time domain | | Example Signal | Type of transform | Frequency domain | |
|-------------|-----------|---|--|------------------|-----------|
| continuous | aperiodic |  | Fourier transform (FT) | continuous | aperiodic |
| continuous | periodic |  | Fourier series (FS) | discrete | aperiodic |
| discrete | aperiodic |  | Discrete time Fourier transform (DTFT) | continuous | periodic |
| discrete | periodic |  | Discrete Fourier transform (DFT) | discrete | periodic |

Figure 1.1: The four types of Fourier transform. Figure adapted from [2].

In this thesis we mostly focus on the Discrete Fourier transform (DFT) but for the start, in Chapter 2, we discuss the Fourier series which is the foundation of the Fourier transform. We explain how it is possible to express a complicated wave function with the sum of simple wave functions and the required conditions.

In Chapter 3, when we study a signal on a computer, the signal has to be discretized. Then the input signal has discrete values. Here we derive DFT and focus specifically on how to interpret the DFT result because the result does not give us an actual amplitude.

In Chapter 4, we show how the fast Fourier transform (FFT) works. FFT is an efficient implementation of DFT. We get the same result but faster.

In Chapter 5, we describe the design of our experiment for classifying the sounds of a washing machine. We show some feature extraction methods and discuss about the problems.

In Chapter 6, we use machine learning to classify the sounds of the stages of the washing machine program. We explain the basic concept of a neural network and how we found hyperparameters for our models.

In Chapter 7, we have our conclusions.

Chapter 2

Fourier Series

2.1 Sinusoid Functions

When you listen to an orchestra, all instruments produce vibrations that propagate through the air then reach your ears. A vibration can be modeled by a sinusoidal waveform. A sinusoid (i.e., sine or cosine function) is a smooth periodic oscillation and can be expressed in terms of time by the form;

$$\begin{aligned}y(t) &= A \sin(\Omega t + \phi) \quad \text{or} \\y(t) &= A \cos(\Omega t + \phi)\end{aligned}\tag{2.1}$$

where

$$\begin{aligned}A &= \text{peak amplitude (non negative)} \\ \Omega &= \text{angular frequency (radians/second)} \\ \phi &= \text{phase shift (radians)} \\ t &= \text{time (seconds)}.\end{aligned}$$

The amplitude is the measure of distance and direction from zero. Therefore signal amplitudes can be negative values as well [7]. However when you express this mathematically, you use positive values. The angular frequency Ω measures angular displacement per second. Its units are therefore degrees or radians per second. Radians are used throughout this thesis when otherwise not mentioned. Ωt shows how many radians are displaced in t seconds. There is a relationship between angular frequency, period and frequency;

$$\Omega(\text{rad/sec}) = \frac{2\pi}{T} = 2\pi f.\tag{2.2}$$

T = period(seconds): the time it takes for a wave to oscillate once
 f = frequency(Hz): the number of times a wave oscillates in one second

The phase shift ϕ given in radians tells us the position of the sinusoid at $t = 0$. Sinusoids can be expressed in a unit circle using a rotated vector. For example, a 10 Hz cosine wave means, the cosine wave oscillates 10 times per second or the rotated vector in the unit circle rotates 10 times per second. Using (2.1), it can be written $y(t) = \cos(2\pi \cdot 10t)$. The python code of the function with amplitude 0.8 and the plot appear below:

```
A=0.8           #Amplitude
f=10           #Frequency
phi=0         #Phase
t=np.arange(0, 1, 0.001) #Time
y=A*np.cos(2*np.pi*f*t+phi) #Function y(t)
```

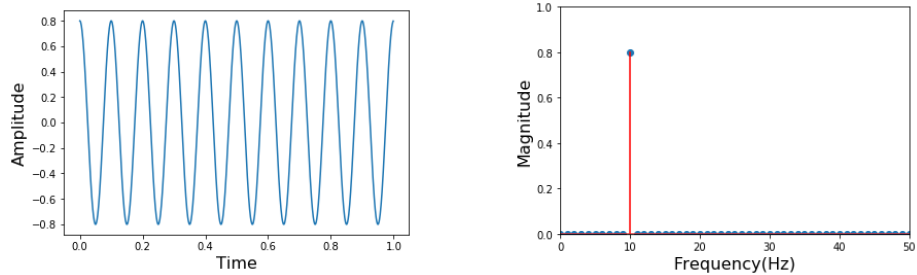


Figure 2.1: Plot of $y(t) = 0.8 \cos(2\pi \cdot 10t)$. The waveform (left) in the time domain and the spectrum (right) in the frequency domain.

This can be called a monochromatic wave, meaning the wave has only one frequency component. In this case, it is easy to find all the information (amplitude, frequency and phase) from its plot in the time domain. However, it is difficult to find such a waveform in a real sound. A real sound can be composed by thousands of sinusoids. Even a combination of three sinusoids, like the one we have in Figure 2.2 makes it very difficult to see directly all components of the function in the time domain.

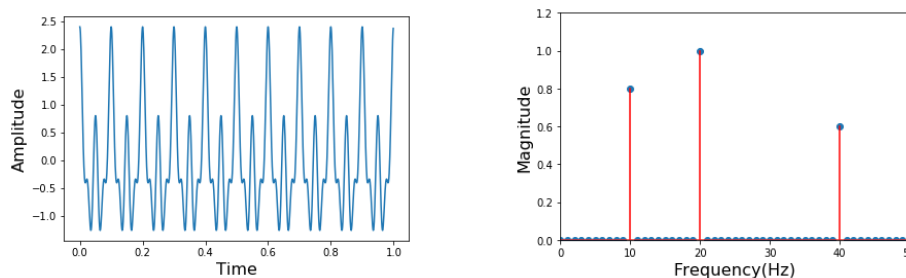


Figure 2.2: Plot of a sinusoid with amplitudes $a_1 = 0.8, a_2 = 1.0, a_3 = 0.6$, frequencies $f_1 = 10, f_2 = 20, f_3 = 40$. The waveform (left) and the spectrum (right).

2.2 Fourier Series

Jean Baptiste Joseph Fourier (1768-1830), a French mathematician and physicist submitted a paper in 1807 to the Institut de France on the use of sinusoids to represent temperature distributions. In the paper he stated that any continuous signal with finite period can be represented as the sum of an infinite series of properly chosen sinusoidal wave functions at different frequencies [2]. This is called Fourier Series now and this is the foundation of the Fourier transform.

In order for us to be able to find the Fourier series of a function, there are certain conditions that must be satisfied. Johann Peter Gustav Lejeune Dirichlet (1805-1859) [17] derived these conditions and they are called the Dirichlet conditions. Since the functions considered in this thesis are sound signals, they are assumed to be smooth and continuous. The Dirichlet conditions can be found Appendix A.

Theorem 2.2.1. (Fourier series) A periodic continuous function $x(t)$ with the period T can be expressed by the summation of sinusoids with frequencies that are integer multiples of the fundamental frequency f_0 .

$$\begin{aligned}
 x(t) &= a_0 + a_1 \cos(2\pi \frac{1}{T}t) + a_2 \cos(2\pi \frac{2}{T}t) + a_3 \cos(2\pi \frac{3}{T}t) + \dots \\
 &\quad + b_1 \sin(2\pi \frac{1}{T}t) + b_2 \sin(2\pi \frac{2}{T}t) + b_3 \sin(2\pi \frac{3}{T}t) + \dots \\
 &= a_0 + \sum_{k=1}^{\infty} (a_k \cos(2\pi f_0 kt) + b_k \sin(2\pi f_0 kt))
 \end{aligned} \tag{2.3}$$

Fundamental frequency f_0 is given by $f_0 = \frac{1}{T}$. All $a_0, a_k, b_k, k = 1, 2, \dots, \infty$ correspond to the amplitude of the sinusoids.

Since period T is given, the fundamental frequency f_0 is determined. The theorem says that the function $x(t)$ can be expressed by the sum of sinusoids

with frequencies $[\Omega_0, 2\Omega_0, 3\Omega_0, 4\Omega_0 \dots]$ where $\Omega_0 = 2\pi f_0$. f_0 has the biggest period of all the decomposed sinusoids and one time oscillation in T . $2f_0$ has exactly 2 times as many oscillations in T and $3f_0$ has exactly 3 times as many oscillations in T . This is necessary if the function x is to be periodic, otherwise the pattern would not repeat[1]. If there is even one sinusoid that doesn't exactly fit in the period T , the function x will not repeat its shape every T seconds. Therefore, the frequencies need to be integer multiples of f_0 .

2.3 Fourier Coefficients

In this section, we will determine amplitudes a_0, a_k, b_k by using area as discussed in [1].

At first, to find out a_0 , we observe the area enclosed by the horizontal axis and each sinusoid in (2.3). We take the integral in the period T for both sides of (2.3). Note that the area of all individual sinusoids in $\sum_{k=1}^{\infty} (a_k \cos(2\pi f_0 kt) + b_k \sin(2\pi f_0 kt))$ will be zero in the period since they fit perfectly in the period T . Therefore the area of a_0 over T which is $a_0 \cdot T$ is equivalent to the area of $x(t)$. Solving for a_0 we obtain $a_0 = \frac{1}{T} \int_0^T x(t) dt$.

To find a_k and b_k , we use area again as a tool.

We start with a_1 . As we mentioned above, the area for $a_1 \cos(2\pi f_0 t)$ enclosed by the horizontal axis in the period T is zero. Now we would like the area for every sinusoid to become zero except this component, $a_1 \cos(2\pi f_0 t)$. The only way we can make the area of $a_1 \cos(2\pi f_0 t) \neq 0$ and all the other terms' area zero is by multiplying by $\cos(2\pi f_0 t)$. Now the area of $x(t) \cos(2\pi f_0 t)$ is equal to the area of $a_1 \cos(2\pi f_0 t) \cos(2\pi f_0 t)$. Solving for a_1 we obtain $a_1 = \frac{2}{T} \int_0^T x(t) \cos(2\pi f_0 t) dt$. Similarly, we can repeat this process for other a_k and b_k . Multiplying by the sinusoids associated with the given coefficient extracts the area as $\frac{AT}{2}$ and makes the other terms disappear. A denotes for a_k and b_k .

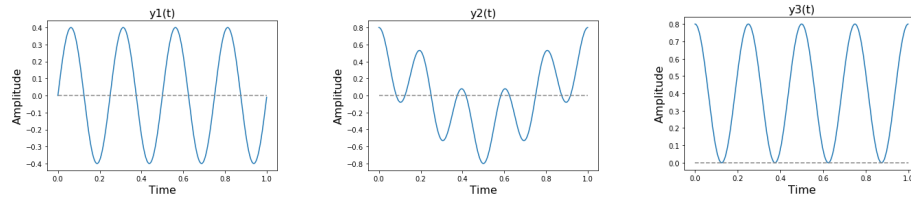


Figure 2.3: From the left, $y_1(t) = A \cos(2\pi ft) \sin(2\pi ft)$ the area is 0, $y_2(t) = \cos(2\pi ft) \cos(2\pi f_1 t)$ the area is 0, $y_3(t) = A \cos(2\pi ft) \cos(2\pi ft)$ the area is $\frac{AT}{2}$.

So finally we introduce Fourier coefficients as below.

Theorem 2.3.1. (Fourier coefficients) When we say that (2.3) is the Fourier series representation of x , the constants a_0, a_k and b_k are given as follows;

$$\begin{aligned} a_0 &= \frac{1}{T} \int_0^T x(t) dt \\ a_k &= \frac{2}{T} \int_0^T x(t) \cos(2\pi f_0 kt) dt \\ b_k &= \frac{2}{T} \int_0^T x(t) \sin(2\pi f_0 kt) dt, \quad k \in 1, 2, \dots, \infty. \end{aligned} \tag{2.4}$$

a_0 with zero frequency stands for DC component which is the average value of the signal over the period T . Note here the integration interval is used as $[0, T]$ but it can be $[-\frac{T}{2}, \frac{T}{2}]$ or $[T_1, T_2]$, as long as period T is the time it takes the waveform to repeat itself.

2.4 Euler's Formula

Euler's formula is one of the most famous formulas in all of mathematics and physics. It provides a link between the complex exponential and trigonometry.

$$\begin{aligned} e^{j\theta} &= \cos(\theta) + j \sin(\theta) \\ e^{-j\theta} &= \cos(\theta) - j \sin(\theta) \end{aligned} \tag{2.5}$$

e is the base of the natural logarithm, j is the imaginary unit which has identity $j = \sqrt{-1}$. Euler's formula can be used to obtain a lot of useful results or identities. For example;

$$\begin{aligned} \cos \theta &= \operatorname{Re}(e^{j\theta}) = \frac{e^{j\theta} + e^{-j\theta}}{2} \\ \sin \theta &= \operatorname{Im}(e^{j\theta}) = \frac{e^{j\theta} - e^{-j\theta}}{2j}. \end{aligned} \tag{2.6}$$

2.5 Exponential Form

Using the preceding expressions, (2.3) of the Fourier series can be written in complex exponential form;

$$\begin{aligned}
 x(t) &= a_0 + \sum_{k=1}^{\infty} (a_k \cos(2\pi f_0 kt) + b_k \sin(2\pi f_0 kt)) \\
 &= a_0 + \sum_{k=1}^{\infty} \left[\frac{a_k}{2} (e^{j2\pi f_0 kt} + e^{-j2\pi f_0 kt}) + \frac{b_k}{2j} (e^{j2\pi f_0 kt} - e^{-j2\pi f_0 kt}) \right] \\
 &= a_0 + \sum_{k=1}^{\infty} \left[\frac{a_k}{2} (e^{j\frac{2\pi}{T} kt} + e^{-j\frac{2\pi}{T} kt}) - j\frac{b_k}{2} (e^{j\frac{2\pi}{T} kt} - e^{-j\frac{2\pi}{T} kt}) \right] \\
 &= a_0 + \sum_{k=1}^{\infty} \left[\frac{a_k - jb_k}{2} e^{j\frac{2\pi}{T} kt} + \frac{a_k + jb_k}{2} e^{-j\frac{2\pi}{T} kt} \right] \\
 &= c_0 + \sum_{k=1}^{\infty} c_k e^{j\frac{2\pi}{T} kt} + \sum_{k=-1}^{-\infty} c_k e^{j\frac{2\pi}{T} kt} \\
 &= \sum_{k=-\infty}^{\infty} c_k e^{j\frac{2\pi}{T} kt} \tag{2.7}
 \end{aligned}$$

where

$$\begin{aligned}
 c_k &= \frac{a_k - jb_k}{2} \\
 c_{-k} &= \overline{c_k} = \frac{a_k + jb_k}{2} \\
 c_0 &= \frac{a_0}{2}.
 \end{aligned} \tag{2.8}$$

The Fourier coefficient in complex exponential form can be calculated by using (2.4) and (2.8) as;

$$c_k = \frac{1}{T} \int_0^T x(t) e^{-j\frac{2\pi}{T} kt} dt, \quad k \in (-\infty, \infty). \tag{2.9}$$

2.6 Orthogonality

In Section 2.3, we found the Fourier coefficients by extracting a certain area enclosed by the horizontal axis by multiplying by the sinusoids associated with the given coefficient, making the other areas zero. In this section, we will express this using inner product and orthogonality. Since the Fourier coefficients (2.4) are the same as (2.9), we show with complex exponential form.

Definition 2.6.1. (Inner product) Inner product of two complex functions $\Phi_m(x)$, $\Phi_n(x)$ is defined as integral over some interval $a \leq x \leq b$,

$$\langle \Phi_m, \Phi_n \rangle = \int_a^b \Phi_m^*(x) \Phi_n(x) dx \quad (2.10)$$

where $\Phi_m^*(x)$ is the complex conjugate of $\Phi_m(x)$.
We introduce the Kronecker delta;

Definition 2.6.2. (the Kronecker delta)

$$\delta_{mn} = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n. \end{cases} \quad (2.11)$$

In order to express the orthogonality of two complex functions, we have;

Definition 2.6.3. (Orthogonal functions) When the two complex functions are orthogonal to each other, the integral in (2.10) is zero whenever $m \neq n$ and not zero when $m = n$. Using the Kronecker delta,

$$\int_a^b \Phi_m^*(x) \Phi_n(x) dx = l \delta_{mn}, \quad l = \text{constant} \quad (2.12)$$

Let us check if our sinusoids are orthogonal functions.

$$\begin{aligned} \langle e^{j\frac{2\pi}{L}mt}, e^{j\frac{2\pi}{L}nt} \rangle &= \int_0^L e^{-j\frac{2\pi}{L}mt} e^{j\frac{2\pi}{L}nt} dt \\ &= \int_0^L e^{j\frac{2\pi}{L}(n-m)t} dt \\ &= \begin{cases} L & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases} \end{aligned} \quad (2.13)$$

Now it is clear that the sinusoids are orthogonal functions and therefore it is possible to extract the area by multiplying as described above to make the other areas zero.

In the Fourier series, we use the set of orthogonal functions $S \subset C(\mathbb{C})$. Therefore a periodic and continuous function can be created by taking linear combinations of the elements of S .

$$S = \{ \dots, e^{-3j\frac{2\pi}{L}t}, e^{-2j\frac{2\pi}{L}t}, e^{-j\frac{2\pi}{L}t}, 1, e^{j\frac{2\pi}{L}t}, e^{2j\frac{2\pi}{L}t}, \dots \} \quad (2.14)$$

Chapter 3

Discrete Fourier Transform

3.1 Fourier Transform

We have seen a method for decomposing a periodic continuous function. Figure 3.1 shows a function in the time domain and in the frequency domain. The abbreviation FS stands for Fourier series.

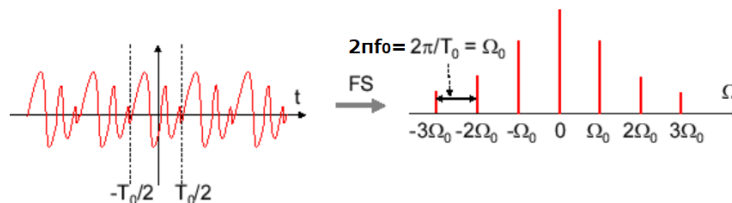


Figure 3.1: A wave function in the time domain (left) and the frequency domain (right) after applying FS between $[-\frac{T_0}{2}, \frac{T_0}{2}]$. The frequency resolution i.e., the spacing between frequencies in the frequency domain is $2\pi f_0 = \Omega_0$. Figure adapted from [35].

In this section, we observe the spectrum for non-periodic functions. Many real sounds are not perfectly periodic. The strategy we use here is as follows: begin with a non-periodic function and let the period get very large, i.e., $T \rightarrow \infty$. assume here that the function repeats over an infinite period of time [1]. The Fourier transform can be viewed as an extension of the Fourier series to non-periodic functions.

Assume that we extract a single period of the waveform in Figure 3.1 and extend the entire period to $[-T_0, T_0]$ (Figure 3.2) without changing the waveform.

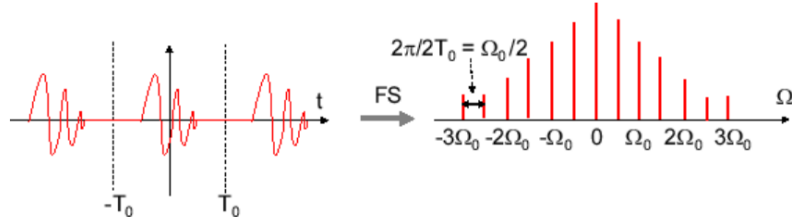


Figure 3.2: Period $2T_0 = [-T_0, T_0]$ gives the frequency resolution $\frac{\Omega_0}{2}$. Figure reproduced from [35].

As the waveform is the same but the period now is double, the spacing between frequencies, i.e., the frequency resolution becomes narrower, $\frac{\Omega_0}{4}$. Similarly, if we make the period larger $[-2T_0, 2T_0]$ then the frequency resolution becomes now $\frac{\Omega_0}{4}$.

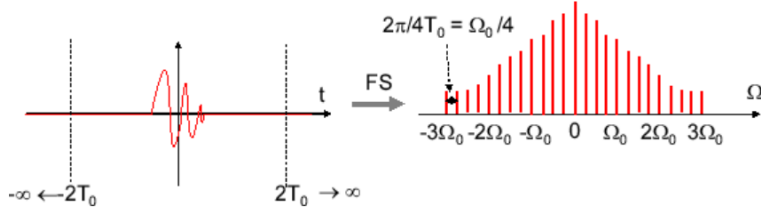


Figure 3.3: period $4T_0 = [-2T_0, 2T_0]$ gives the frequency resolution $\frac{\Omega_0}{4}$. Figure reproduced from [35].

How about if we have period $T_0 \rightarrow \infty$? As the T_0 increases, f_0 decreases. Therefore, the frequency resolution becomes smaller with no gaps between the frequencies when the limit $T_0 \rightarrow \infty$. Thus, all frequencies become known when $T_0 \rightarrow \infty$ [1]. This is the idea of the Fourier transform.

To derive the Fourier transform, once again we bring back the equations for the

Fourier series and the Fourier coefficient.

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j\frac{2\pi}{T}kt} \quad (3.1)$$

$$c_k = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-j\frac{2\pi}{T}kt} dt \quad (3.2)$$

Since we can write $\frac{1}{T}$ as Δf , (3.2) becomes;

$$c_k = \Delta f \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-j2\pi k\Delta f t} dt. \quad (3.3)$$

$k\Delta f$ is the fundamental frequency f_0 multiplied by integer k so we denote as f_k . Taking the limit $\Delta f \rightarrow 0$ corresponds to $T \rightarrow \infty$ thus we set (3.3) into (3.1) then;

$$\begin{aligned} x(t) &= \sum_{k=-\infty}^{\infty} \left(\lim_{T \rightarrow \infty} \Delta f \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-j2\pi f_k t} dt \right) e^{j2\pi f_k t} \\ &= \lim_{T \rightarrow \infty} \sum_{k=-\infty}^{\infty} \left(\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-j2\pi f_k t} dt \right) e^{j2\pi f_k t} \Delta f \\ &= \int_{-\infty}^{\infty} \underbrace{\left(\int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \right)}_A e^{j2\pi f t} df. \end{aligned} \quad (3.4)$$

B

The reason that f_k became f is that f_k represented discrete values, i.e., f_k jumped from a frequency to an another frequency, however these values are continuous when $T \rightarrow \infty$. Therefore f_k denotes now f .

As you see, $e^{-j2\pi f t}$ in A is a function of f and t . However, now multiply by $x(t)$ and integrate for t from $-\infty$ to ∞ which makes this as a function of f . In B, first, we have a multiplication of A and $e^{j2\pi f t}$ and the result is a function of f and t . Then this function is multiplied by Δf and summed up between $(-\infty, \infty)$.

Now we can separate this equation, and obtain equations of the Fourier transform and the inverse Fourier transform.

Definition 3.1.1. (Fourier transform) The Fourier transform of the function x is denoted by X .

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \quad (\text{Fourier transform}) \quad (3.5)$$

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df \quad (\text{Inverse Fourier transform}) \quad (3.6)$$

As we showed above, we assume as $T \rightarrow \infty$ in the Fourier transform, the frequency resolution goes to 0 in the Fourier series. Therefore, the sequence of Fourier coefficients becomes a continuous function $X(f)$.

3.2 The Dirac Delta Function

So far we have considered only continuous functions. In fact, real sound is continuous. However, the computer can only handle discrete values. Therefore we use an Analog-Digital converter to transform the original input signal to a discrete time-signal. To convert analog signal to digital signal we need **sampling** and **quantization**. Sampling means to discretize time and quantization is to discretize amplitude. We often use the word sampling for both sampling and quantization.

The period for measuring the discrete time-signal is called the **sampling period** and the number of data points taken in one second is called the **sampling frequency**. Sampling period and sampling frequency have an inverse relationship, just like the relationship between period and frequency in the previous chapter.

The Dirac delta function is a generalized function invented by Paul Adrien Maurice Dirac(1902-1984) [11] in order to express particles existing only in one special point e.g., point mass. Theoretically, the value is infinite at one point, but when integrating over the entire real line, it becomes the finite quantity, 1. This is described as below;

Definition 3.2.1. (the Dirac delta function)

$$\delta(t) = \begin{cases} 0 & t \neq 0 \\ \infty & t = 0 \end{cases} \quad (3.7)$$

$$\int_{-\infty}^{\infty} \delta(t) dt = 1. \quad (3.8)$$

Let $f(t)$ be a real continuous function. Then this will be satisfied for all t ;

$$\int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0). \quad (3.9)$$

It is called the **shifting property** of $\delta(t)$ [6]. If we use the form $\delta(t - t_1)$, then the point $t = t_1$ is the special point.

$$\int_{-\infty}^{\infty} f(t)\delta(t - t_1)dt = f(t_1) \quad (3.10)$$

As we have seen, we can take out a value of a function at any one point using convolution with the function and the Dirac delta function. We will discuss convolution in the later sections.

3.3 The Comb Function

Now we are ready for Dirac comb function which is also known as impulse train. The comb function looks like the second graph in Figure 3.4. The name is given because the form looks like a comb. This function is made by the sum of the Dirac delta functions.

Definition 3.3.1. (Dirac comb function)

$$\text{comb}_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - n \Delta t) \quad (3.11)$$

where n is index of the sample and Δt is sampling interval. Then we denote [3, 22] the discrete time-signal $x_s(t)$ as;

$$x_s(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - n \Delta t) \quad (3.12)$$

where $x(t)$ is an original input signal. We assume the points where the arrows stand are sample points (Figure 3.4).

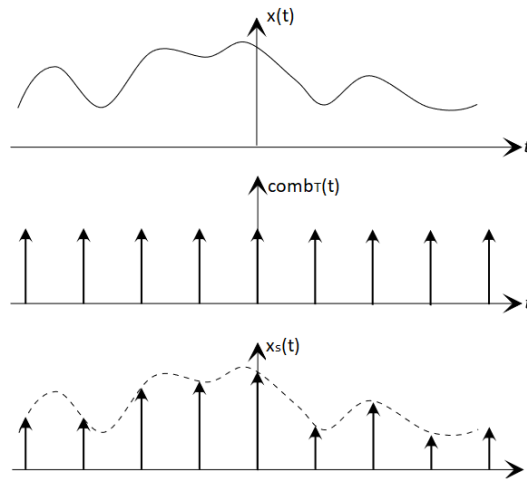


Figure 3.4: Sampling by a Dirac comb function: an original input signal $x(t)$ is multiplied by the comb function to produce the discrete time-signal $x_s(t)$. Figure adapted from [22].

3.4 Discrete Fourier Transform

Consider the Fourier transform (3.5) of the discrete time-signal $x_s(t)$ with a period T . The original input signal $x(t)$ is assumed $x(t) = 0$ outside of the

period T . We take N equally spaced samples of $x(t)$.

$$\begin{aligned}
X(f) &= \int_{-\infty}^{\infty} (x(t) \sum_{n=0}^{N-1} \delta(t - n \Delta t)) e^{-j2\pi ft} dt \\
&= \sum_{n=0}^{N-1} \int_{-\infty}^{\infty} (x(t) e^{-j2\pi ft}) \delta(t - n \Delta t) dt \quad (\text{using the shifting property}) \\
&= \sum_{n=0}^{N-1} x(n \Delta t) e^{-j2\pi f(n \Delta t)} \tag{3.13}
\end{aligned}$$

Just as with t , f also has to be discretized by sampling. This relationship is the key to understanding the Discrete Fourier transform [29].

$$\begin{aligned}
t = n \Delta t &= \frac{nT}{N}, \quad (0 \leq n \leq N-1) \\
f = k \Delta f &= \frac{kf_s}{N}, \quad (0 \leq k \leq N-1) \tag{3.14}
\end{aligned}$$

where Δf is frequency resolution, k is index of discrete frequency and f_s is sampling frequency defined by $f_s = \frac{N}{T}$. Then, (3.13) becomes;

$$X(k \Delta f) = \sum_{n=0}^{N-1} x(n \Delta t) e^{-j2\pi (k \Delta f)(n \Delta t)}. \tag{3.15}$$

Since,

$$\Delta f \cdot \Delta t = \frac{f_s}{N} \cdot \frac{T}{N} = \frac{1}{N} \tag{3.16}$$

(3.15) becomes;

$$X(k \Delta f) = \sum_{n=0}^{N-1} x(n \Delta t) e^{-j \frac{2\pi nk}{N}}.$$

Let $X_k = X(k \Delta f)$ and $x_n = x(n \Delta t)$ then,

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j \frac{2\pi nk}{N}}, \quad (\text{DFT}) \tag{3.17}$$

This is known as the Discrete Fourier transform (DFT) and the inverse discrete Fourier is

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j \frac{2\pi nk}{N}}, \quad (\text{IDFT}). \tag{3.18}$$

To summarize the notations,

$$\begin{aligned}
 T &: \text{ sampling period} \\
 f_s = \frac{N}{T} &: \text{ sampling frequency (sampling rate),} \\
 &\quad \text{the number of data points taken in one second} \\
 N = f_s \cdot T &: \text{ total number of sample points} \\
 \Delta t = \frac{T}{N} &: \text{ sampling interval} \\
 n \in [0 : N - 1] &: \text{ index of sample (discrete time)} \\
 \Delta f = \frac{f_s}{N} &: \text{ frequency resolution} \\
 k \in [0 : N - 1] &: \text{ index of discrete frequency.}
 \end{aligned}$$

To compute the DFT is the same as computing the inner product between an input signal x_n and a complex sinusoid $e^{-j\frac{2\pi nk}{N}}$. Therefore DFT can be understood as the projection of an input signal onto a finite set of complex sinusoids. In the absolute value of the DFT result X_k which is complex number, we get the amount of the complex sinusoids which are present in the input signal of the frequency. We can find the absolute value by the following formula.

$$\text{The absolute value} = |X_k| = \sqrt{\text{Re}(X_k)^2 + \text{Im}(X_k)^2}$$

In the phase we identify the location of these sinusoids with respect to time zero [9]. To calculate phase, we do;

$$\phi = \arctan\left(\frac{\text{Im}(X_k)}{\text{Re}(X_k)}\right).$$

3.5 How to Interpret DFT Spectral Results

Audio sound is a real-valued signal. By using Euler's identity that we presented in Section 2.4, we can express a real-valued signal by a cosine function i.e., the sum of two complex sinusoids [9].

$$x_n = A_0 \cos\left(\frac{2\pi nl}{N}\right) = \frac{A_0}{2} e^{j\frac{2\pi nl}{N}} + \frac{A_0}{2} e^{-j\frac{2\pi nl}{N}} \quad (3.19)$$

where A_0 is an amplitude. Then the DFT of this real valued-signal is;

$$\begin{aligned}
 X_k &= \sum_{n=0}^{N-1} x_n e^{-j \frac{2\pi nk}{N}} \\
 &= \sum_{n=0}^{N-1} \left(\frac{A_0}{2} e^{j \frac{2\pi nl}{N}} + \frac{A_0}{2} e^{-j \frac{2\pi nl}{N}} \right) e^{-j \frac{2\pi nk}{N}} \\
 &= \sum_{n=0}^{N-1} \frac{A_0}{2} e^{j \frac{2\pi(l-k)n}{N}} + \sum_{n=0}^{N-1} \frac{A_0}{2} e^{-j \frac{2\pi(l+k)n}{N}} \\
 &= N \frac{A_0}{2}
 \end{aligned} \tag{3.20}$$

for $k = l, -l$ and 0 for rest of the k 's. As you can see, in order to represent real-valued signals, we need to have both positive and negative exponentials because of the cosine identity. For the result of DFT of a real-valued signal, the negative frequencies will be a mirror image of the positive frequencies in the spectrum. It is important to know that DFT does not provide us with the actual amplitude A_0 which was expressed in the Fourier series. However, it returns a ratio between the amplitude of the frequency components of the input signal. Below is an example of DFT implementation by python and a plot.

```

def DFT(x,N, fs ):
    X=np.array ( []) #initialize X
    nv=np.arange(-N/2, N/2) #time index
    kv=np.arange(-N/2, N/2) #frequency index
    for k in kv:
        sinusoid=np.exp(1j*2*np.pi*k*nv/N) #DFT-sinusoid
        X = np.append(X,sum(x*np.conjugate(sinusoid)))
    plt.stem(kv,abs(X))
    plt.axis([-fs/2, fs/2, min(X), max(X)+5])

```

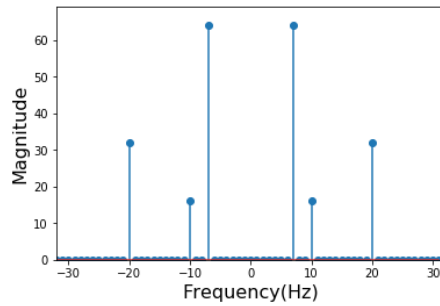


Figure 3.5: Plot of the spectrum of a real-valued signal for $N = fs = 64$, The function x contains; $f_1 = 7\text{Hz}$, $f_2 = 10\text{Hz}$, $f_3 = 20\text{Hz}$, $a_1 = 2$, $a_2 = 0.5$, $a_3 = 1$.

From Figure 3.5 we read that the absolute value of the spectrum are correct based on the calculations of equation (3.20). Also pay attention to the range of the x -axis, the time- and the frequency index in the code. When you want to plot frequencies correctly as (3.20), then this is the way to do it. For simplicity, assume period $T = 1$ so sampling frequency f_s is equal to N in this example.

However we do not actually need the negative frequencies to get the spectrum. We will need the negative frequencies when we reconstruct the original input signal. Therefore we calculate DFT using both time- and frequency index as $[0 : N - 1]$ and plot only positive frequency.

```
def DFT(x,N, fs ):
    X=np.array ([])
    n=np.arange(N)
    for k in range(N):
        sinusoid=np.exp(1j*2*np.pi*k*n/N)
        X = np.append(X,sum(x*np.conjugate(sinusoid)))
    plt.stem(float(fs)*np.arange(X.size)/float(N), abs(X))
    plt.axis([0, fs/2.0, min(X), max(X)+5])
```

Then the plot is the same as above but only positive frequencies side.

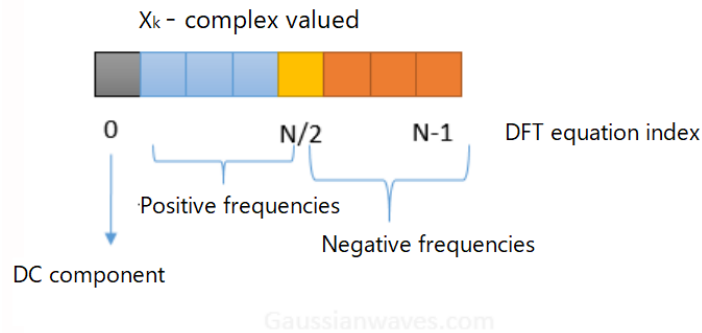


Figure 3.6: DFT results using the complex DFT for a real valued signal. Figure adapted from [23].

3.6 Computation of DFT

We can arrange the DFT (3.17) for signal processing and it can be expressed as;

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{nk} \tag{3.21}$$

where W_N is the **complex N -th roots of unity** such as $W_N = e^{-j\frac{2\pi}{N}} = \cos(\frac{2\pi}{N}) - j \sin(\frac{2\pi}{N})$. These can be plotted in the complex plane as below.

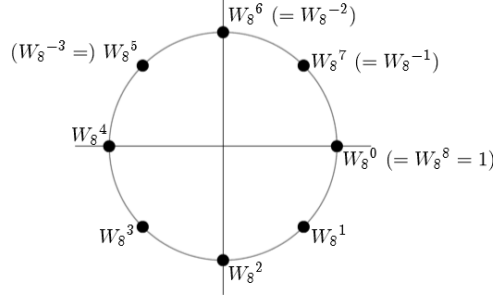


Figure 3.7: W_N for N equally divided points on the unit circle ($N = 8$). Figure reproduced from [31].

The inner product (3.21) for $N = 8$ can be expressed with a system of linear equations.

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{pmatrix} = \begin{pmatrix} W^{0\cdot0} & W^{1\cdot0} & W^{2\cdot0} & W^{3\cdot0} & W^{4\cdot0} & W^{5\cdot0} & W^{6\cdot0} & W^{7\cdot0} \\ W^{0\cdot1} & W^{1\cdot1} & W^{2\cdot1} & W^{3\cdot1} & W^{4\cdot1} & W^{5\cdot1} & W^{6\cdot1} & W^{7\cdot1} \\ W^{0\cdot2} & W^{1\cdot2} & W^{2\cdot2} & W^{3\cdot2} & W^{4\cdot2} & W^{5\cdot2} & W^{6\cdot2} & W^{7\cdot2} \\ W^{0\cdot3} & W^{1\cdot3} & W^{2\cdot3} & W^{3\cdot3} & W^{4\cdot3} & W^{5\cdot3} & W^{6\cdot3} & W^{7\cdot3} \\ W^{0\cdot4} & W^{1\cdot4} & W^{2\cdot4} & W^{3\cdot4} & W^{4\cdot4} & W^{5\cdot4} & W^{6\cdot4} & W^{7\cdot4} \\ W^{0\cdot5} & W^{1\cdot5} & W^{2\cdot5} & W^{3\cdot5} & W^{4\cdot5} & W^{5\cdot5} & W^{6\cdot5} & W^{7\cdot5} \\ W^{0\cdot6} & W^{1\cdot6} & W^{2\cdot6} & W^{3\cdot6} & W^{4\cdot6} & W^{5\cdot6} & W^{6\cdot6} & W^{7\cdot6} \\ W^{0\cdot7} & W^{1\cdot7} & W^{2\cdot7} & W^{3\cdot7} & W^{4\cdot7} & W^{5\cdot7} & W^{6\cdot7} & W^{7\cdot7} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} \quad (3.22)$$

Here we use W^{nk} instead of writing W_8^{nk} . As you see, if you compute the DFT formula as it is, you will need at least N^2 operations. In other words, the calculation order of DFT is $O(N^2)$. Thus, the amount of computations increases quadratically with N and it can take an unreasonably long time to calculate this, even with a computer, for large N .

The matrix of the complex N -th roots of unity is a helical structure in the complex number space [33]. This animation matrix shows exactly the same as above but this helps more to get the intuition of the DFT.

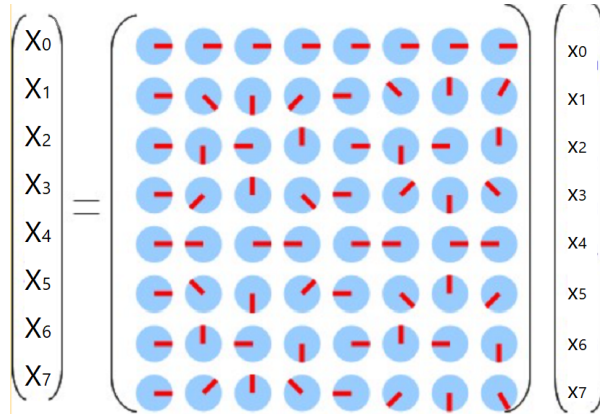


Figure 3.8: The animation of the complex 8th roots of unity matrix. Figure adapted from [33].

The first row (for X_0) of the complex 8th roots of unity matrix can be used for a measurement how much DC component is in the signal. In the second row (for X_1) we can see that it rotates clockwise by $1/8$ turns each time a circle shifts to the right. As the exponent of the second row of the matrix (3.22) increases by 1, it moves exactly like Figure 3.7. It is one revolution with 8 circles from left to right. X_1 contains the amount of the lowest frequency component. Using $N = 8$, X_1 is a frequency component of a $1/8$ sampling frequency, f_s . For example, if $f_s = 1000\text{Hz}$ and $N = 8$ then $1000/8 = 125\text{Hz}$ so X_1 shows the amount of the frequency 125Hz that is present in the input signal x . Now, we look at the third row (for X_2). The exponent of W is doubled compared with the second row therefore it has twice the angular velocity. Looking at the rotation speed of the thick line in the circle, you can see that it is rotating at twice the angular velocity. It has 2 rotations with 8 circles. Using the above example, X_2 shows the amount of the frequency 250Hz that is present in the input signal x . Similarly, angular velocities increase three times, four times, ..., seven times. However, the fastest velocity appears in X_4 which extracts the highest frequency component. With the above example, it is 500Hz . From the sixth row (X_5) the rotation becomes counter-clockwise and gets slower toward X_7 . As we discussed in the previous section, we get negative frequencies after $N/2$ and the negative frequencies are just a mirror of the positive frequencies. Therefore, the frequency component of the $1/8$ sampling frequency appears in to two places, X_1 (positive frequency) and X_7 (negative frequency).

Chapter 4

Cooley-Tukey FFT Algorithm

The Fast Fourier Transform (FFT) is an efficient implementation of DFT. FFT reduces the computation time significantly by taking advantage of the periodicity and symmetry of DFT. Such an algorithm was used by Carl Friedrich Gauss (1777–1855) [20] more than one hundred years earlier. However this early work was forgotten because it lacked the tool, i.e., a digital computer, to make it practical. FFT was rediscovered in the 1960s by two mathematicians, J. W. Cooley (1926-2016) and J. W. Tukey (1915-2000). Their article "An Algorithm for the Machine Calculation of Complex Fourier Series" was published at the right time which is the beginning of the computer revolution [2]. So they are referenced as the inventors of the FFT algorithm in most signal processing literature. There are several FFT algorithms. In this thesis, we will show the most common FFT algorithm which is named after the above two, the Cooley–Tukey FFT algorithm with a **divide and conquer** approach. To take advantage of this algorithm, it is good to choose N to be a power of two; $N = 2^m$.

4.1 Matrix Form

There are some useful properties to simplify the matrix (3.22).

- 1) $[W_N^k]^N = e^{-j2\pi \frac{kN}{N}} = 1 = W_N^0$
- 2) $W_N^{N-m} = W_N^N \cdot W_N^{-m} = 1 \cdot W_N^{-m} = W_N^{-m}$
- 3) $W_N^k = W_N^{k+lN}$ for all integer l .

Using 1) and 3) properties, we can rewrite the matrix (3.22) as;

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} \quad (4.1)$$

Furthermore, $W^{k-4} = e^{-j2\pi\frac{k-4}{8}} e^{-j2\pi\frac{k}{8}} = e^{j\pi} W^k = -W^k$, therefore $W^k = -W^{k-4}$. We apply this to $k = 4,5,6,7$.

$$\begin{pmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & -W^0 & -W^1 & -W^2 & -W^3 \\ W^0 & W^2 & -W^0 & -W^2 & W^0 & W^2 & -W^0 & -W^2 \\ W^0 & W^3 & -W^2 & W^1 & -W^0 & -W^3 & W^2 & -W^1 \\ W^0 & -W^0 & W^0 & -W^0 & W^0 & -W^0 & W^0 & -W^0 \\ W^0 & -W^1 & W^2 & -W^3 & -W^0 & W^1 & -W^2 & W^3 \\ W^0 & -W^2 & -W^0 & W^2 & W^0 & -W^2 & -W^0 & W^2 \\ W^0 & -W^3 & -W^2 & -W^1 & -W^0 & W^3 & W^2 & W^1 \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix} \quad (4.2)$$

Now looking at closely of the third row in the matrix (4.2), the one associated with $X(2)$, we see some useful symmetry. Therefore we can write as;

$$X(2) = \begin{pmatrix} W^0 & W^2 & -W^0 & -W^2 \end{pmatrix} \begin{pmatrix} x(0) + x(4) \\ x(1) + x(5) \\ x(2) + x(6) \\ x(3) + x(7) \end{pmatrix} \quad (4.3)$$

If we write like this, the computation can be reduced by a half. This can be used for all $X(\text{even})$ in this case, $X(0), X(2), X(4), X(6)$.

Now looking carefully at the row for $X(1)$, the difference between the right side and the left side is only the sign. So we can express as;

$$X(1) = \begin{pmatrix} W^0 & W^1 & W^2 & W^3 \end{pmatrix} \begin{pmatrix} x(0) - x(4) \\ x(1) - x(5) \\ x(2) - x(6) \\ x(3) - x(7) \end{pmatrix} \quad (4.4)$$

This can be said about all $X(\text{odd})$ which is $X(1), X(3), X(5), X(7)$ when $N = 8$. From the above,

$$\begin{pmatrix} X(0) \\ X(2) \\ X(4) \\ X(6) \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^2 & -W^0 & -W^2 \\ W^0 & -W^0 & W^0 & -W^0 \\ W^0 & -W^2 & -W^0 & W^2 \end{pmatrix} \begin{pmatrix} x(0) + x(4) \\ x(1) + x(5) \\ x(2) + x(6) \\ x(3) + x(7) \end{pmatrix} \quad (4.5)$$

$$\begin{pmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{pmatrix} = \begin{pmatrix} W^0 & W^1 & W^2 & W^3 \\ W^0 & W^3 & -W^2 & W^1 \\ W^0 & -W^1 & W^2 & -W^3 \\ W^0 & -W^3 & -W^2 & -W^1 \end{pmatrix} \begin{pmatrix} x(0) - x(4) \\ x(1) - x(5) \\ x(2) - x(6) \\ x(3) - x(7) \end{pmatrix} \quad (4.6)$$

Now, $W^{k-2} = e^{-j2\pi\frac{-2}{8}} e^{-j2\pi\frac{k}{8}} = e^{\frac{j\pi}{2}} W^k = jW^k$, therefore $W^k = -jW^{k-2}$. We apply this to the half of matrix (4.6).

$$\begin{pmatrix} X(1) \\ X(3) \\ X(5) \\ X(7) \end{pmatrix} = \begin{pmatrix} W^0 & W^1 & -jW^0 & -jW^1 \\ W^0 & -jW^1 & jW^0 & W^1 \\ W^0 & -W^1 & -jW^0 & jW^1 \\ W^0 & jW^1 & jW^0 & -W^1 \end{pmatrix} \begin{pmatrix} x(0) - x(4) \\ x(1) - x(5) \\ x(2) - x(6) \\ x(3) - x(7) \end{pmatrix}$$

Then we again find symmetries or symmetries with different signs so we do the same procedure as before.

$$\begin{aligned} \begin{pmatrix} X(0) \\ X(4) \end{pmatrix} &= \begin{pmatrix} W^0 & W^0 \\ W^0 & -W^0 \end{pmatrix} \begin{pmatrix} x(0) + x(4) + x(2) + x(6) \\ x(1) + x(5) + x(3) + x(7) \end{pmatrix} \\ \begin{pmatrix} X(2) \\ X(6) \end{pmatrix} &= \begin{pmatrix} W^0 & W^2 \\ W^0 & -W^2 \end{pmatrix} \begin{pmatrix} x(0) + x(4) - (x(2) + x(6)) \\ x(1) + x(5) - (x(3) + x(7)) \end{pmatrix} \\ \begin{pmatrix} X(1) \\ X(5) \end{pmatrix} &= \begin{pmatrix} W^0 & W^1 \\ W^0 & -W^1 \end{pmatrix} \begin{pmatrix} x(0) - x(4) - j(x(2) - x(6)) \\ x(1) - x(5) - j(x(3) - x(7)) \end{pmatrix} \\ \begin{pmatrix} X(3) \\ X(7) \end{pmatrix} &= \begin{pmatrix} W^0 & -jW^1 \\ W^0 & jW^1 \end{pmatrix} \begin{pmatrix} x(0) - x(4) + j(x(2) - x(6)) \\ x(1) - x(5) + j(x(3) - x(7)) \end{pmatrix} \end{aligned} \quad (4.7)$$

In this section 4.1 we referred to [32].

4.2 Mathematical Expression

Here we show with a mathematical expression.

We start with (3.21) for $N = 8$,

$$X_{(k/8)} = \sum_{n=0}^{N-1} x_n W^{nk}. \quad (4.8)$$

We divide x_n with even and odd numbers,

$$\begin{aligned} X_{(k/8)} &= \sum_{n=0}^{\frac{N}{2}-1} x_{(2n)} W^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x_{(2n+1)} W^{(2n+1)k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{(2n)} W^{2nk} + W^k \sum_{n=0}^{\frac{N}{2}-1} x_{(2n+1)} W^{2nk}. \end{aligned} \quad (4.9)$$

We set;

$$\begin{aligned}\text{even group} &:= x_{(2n)} = p(n) \\ \text{odd group} &:= x_{(2n+1)} = q(n).\end{aligned}$$

Then,

$$X_{(k/8)} = \sum_{n=0}^{\frac{N}{2}-1} p(n)W^{2nk} + W^k \sum_{n=0}^{\frac{N}{2}-1} q(n)W^{2nk}. \quad (4.10)$$

Note, since $W^{2nk} = e^{2\frac{-j2\pi}{N}nk} = e^{\frac{-j2\pi}{N}nk}$, therefore we are now using 4th roots of unity to express this equation [1].
Now we set $W^{2nk} = W'^{nk}$ and again divide $p(n)$ and $q(n)$ with even and odd numbers.

$$\begin{aligned}X_{(k/8)} &= \sum_{n=0}^{\frac{N}{2}-1} p(n)W'^{nk} + W^k \sum_{n=0}^{\frac{N}{2}-1} q(n)W'^{nk} \\ &= \sum_{n=0}^{\frac{N}{4}-1} p(2n)W'^{2nk} + \sum_{n=0}^{\frac{N}{4}-1} p(2n+1)W'^{(2n+1)k} \\ &\quad + W^k \left(\sum_{n=0}^{\frac{N}{4}-1} q(2n)W'^{2nk} + \sum_{n=0}^{\frac{N}{4}-1} q(2n+1)W'^{(2n+1)k} \right)\end{aligned} \quad (4.11)$$

We assign these functions like this;

$$\begin{aligned}p(2n) &= a(n) \\ p(2n+1) &= b(n) \\ q(2n) &= c(n) \\ q(2n+1) &= d(n).\end{aligned}$$

Then,

$$X_{(k/8)} = \sum_{n=0}^{\frac{N}{4}-1} a(n)W'^{2nk} + W^k \sum_{n=0}^{\frac{N}{4}-1} b(n)W'^{2nk} + W^k \left(\sum_{n=0}^{\frac{N}{4}-1} c(n)W'^{2nk} + W'^k \sum_{n=0}^{\frac{N}{4}-1} d(n)W'^{2nk} \right). \quad (4.12)$$

Note, since $W'^{2nk} = e^{2\frac{-j2\pi}{N/2}nk} = e^{\frac{-j2\pi}{N/4}nk}$, we can express this equation with 2nd roots of unity [1].

As we have seen, we divide the given problem into smaller subproblems of same type and solve these subproblems and we proceed recursively until we reach a stage where no more division is possible. This approach is called divide and

conquer. In the last stage, we have 24 operations which is reduced a lot from 64 operations. Generally, for N points, the N^2 cost can be reduced $N \log_2 N$ by the divide and conquer method without changing the result if N is a power of 2. When $N = 8$, it might not seem like a big difference. However, imagine $N = 131072$ which we used for our project. It is now a huge difference for the computation time.

Chapter 5

Audio Classification

Audio classification consists of extracting significant features from a sound and classifying them into a most suitable sound group. As we mentioned before, frequency is sometimes more practical to work with than the waveform of a sound. With development of computers and the discovery of FFT, the study of audio classification has progressed significantly. One of the most interesting studies in this field is *machine hearing*. Machine hearing is the ability of computers to recognize surrounding sounds as human beings do [14].

Take for example Shazam, which is a smartphone app which recognizes music. Shazam analyzes songs from around the world and stores them as waveform data. These are also saved as an *audio fingerprint* in its own database. An audio fingerprint is a digital summary of sound features that can be used to identify an audio sample or quickly locate similar information in an audio database. When a user downloads Shazam and taps the screen, then the smartphone listens to the surrounding sound with the built-in microphone and digitizes the sound. When the device captures the fingerprint of the song, Shazam accesses its own database and compares if this fingerprint matches with one in the database. Since all songs have unique features, Shazam can identify songs [19]. This may give some clues to what we are going to do.

5.1 Washing Machine

We are going to classify sounds of a washing machine. A washing program steps through the different stages e.g., water-filling, washing, spinning, etc. Each stage has a different sound therefore we can determine the stage from the sound even though we don't see the washing machine. Therefore the sounds in each stage have unique features. We can describe the differences by looking at the frequencies. The stage can then be identified by the frequency. It may then

be possible to get information about the stage of the washing program from a remote place without seeing or hearing the washing machine. In summary, our goal is to classify sounds of a washing program by features of the frequencies using FFT (Short-time Fourier transform (STFT)) and to detect a stage from a short sound clip.

In the beginning, we considered this procedure:

1. Preparing the environment
2. Sampling sounds of washing machine
3. Splitting sounds by each stage
4. Short-time Fourier transform (STFT)
5. Spectrum analysis

We describe how the experiment proceeded.

5.2 Experiment Environment

The washing machine used in our experiment is : w365H LE, Electrolux.

It appears to be an industrial washing machine (see Figure 5.1 (a)). It is placed in the basement. The laundry room is quite noisy. Even when all the machines are turned off, there are still sounds, especially from these pipes in Figure 5.1 (b).

According to the washing machine manual [12],

$$\begin{array}{lll} 52\text{rpm} & f = 52/60 \approx 0.866\text{Hz} & \text{(spinning for washing)} \\ 1100\text{rpm} & f = 1100/60 \approx 18.33\text{Hz} & \text{(maximum spinning(centrifugation)).} \end{array}$$

Therefore we expect that those frequencies appear in the frequency domain.



(a) The washing machine



(b) The noisy pipes in the laundry room



(c) The USB microphone

Figure 5.1: Environment

USB microphone : Mini Mikrofon med USB kontakt ¹

Table 5.1: The microphone specifications

| | |
|--------------------|----------------------|
| Frequency Response | 50Hz - 16kHz |
| Sensitivity | -30dB \pm 3dB |
| Impedance | \leq 2200 Ω |

We used a mobile app for recording sounds at first but the frequency of around 18Hz which we were supposed to be able to see couldn't be confirmed in the spectrum. So instead we used the microphone above. In the specification of the microphone, the frequency response between 50Hz to 16kHz is what is guaranteed, but the observed range is certainly larger.

Software: Audacity which is a free, open source digital audio editor. We used for recording, exporting as wav.file and editing e.g., cutting and deleting.

¹There is no company information. Please refer this site for the product.
https://www.24.se/datortillbehor/datortillbehor/datortillbehor-mikrofon/mini-mikrofon-med-usb-kontakt?gclid=EAIaIQobChMIrfe-1o7y4AIVC6WaChOCGwcAEAQYASABEGluPD_BwE (Accessed 20190308)

To get every sample as similar as possible, the same bathmat is washed using the same washing program every time. However, it is impossible to get the same sound every time because it is not possible to control the timing at which the mat falls and what kind of sound is made.

5.3 Sampling and Sound Editing

For sampling, we used the USB microphone and Audacity. The washing program which is used all the time is "Permanent Press wash program, max 40°C. Half load".

Table 5.2: Program

| Number | Program |
|--------|------------------------|
| 1 | press the start-button |
| 2 | spin slow-middle-slow |
| 3 | water-in |
| 4 | washing |
| 5 | water-out |
| 6 | water-in |
| 7 | rinsing |
| 8 | water-out |
| 9 | water-in |
| 10 | rinsing |
| 11 | water-out |
| 12 | water-in |
| 13 | rinsing |
| 14 | water-out |
| 15 | spin slow-full-slow |
| 16 | finish |

The whole program takes about 32 minutes. We recorded all sounds at once. This one unbroken section of a sound is normally called a *track*. Then, the track can be split by each stage. We can process everything with Audacity. There is a 'click' sound between each stage. The click is made as the sign that the next stage will come soon but not right away. So if we cut the sample at each click, the next stage sample will have sounds of the previous stage. This is not good for extracting features. Also, there are several moments when the machine is very calm i.e., it doesn't make much sound. For sounds during those moments, even the human ear can't distinguish the different stages. It happens often between stages. So we called the sound of the moment as "between". In the beginning "between" is included as a sample in stages. Later the clips are split more finely and "between"s are deleted.

5.3.1 Sampling & Time Domain Aliasing

As we discussed in Section 3.2, a real sound is necessary to be discretized by time and amplitude when we analyze it on a computer. Then, at what time intervals should we sample the input signal? If you sample the input signal finely, you will definitely be able to reconstruct the original input signal. However, the amount of data will be enormous. On the other hand, if the sample interval is too large, the original information will be lost.

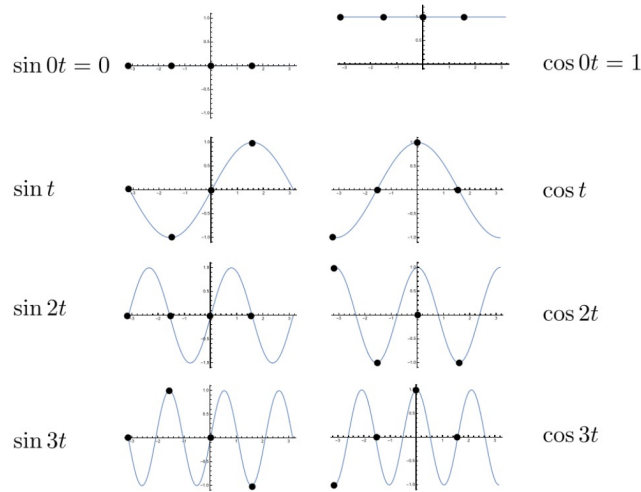


Figure 5.2: Sampling $N = 4$. Figure reproduced from [30].

Figure 5.2 shows that points in $\cos 3t$ and $\cos t$ have exactly the same position and we see the same with $\sin 0t$ and $\sin 2t$. These two signals contain the same four samples, yet are very different signals. This is called a ***time domain aliasing***. Aliasing is an effect that makes different signals indistinguishable when sampled. When this happens, the analog signal may not be completely reconstructed from the sampled signal [2]. So, this is not good sampling. N has to be increased. Then what is a reasonable sample interval?

5.3.2 Convolution Theorem

Before we discuss the sampling interval, we introduce the convolution theorem which actually came up in Chapter 2. The convolution theorem states a fundamental property of the Fourier transform and this is the most important theorem we need to understand the concept of signal processing.

The convolution theorem for Fourier transform states that convolution in the time domain is equivalent to multiplication in the frequency domain and vice

versa [3, 5, 10].

Theorem 5.3.1. (convolution theorem)

If two signals $x(t)$ and $y(t)$ are Fourier transformable, then the Fourier transform of a convolution of the two signals $x(t)$ and $y(t)$ is the product of the Fourier transform. Also the Fourier transform of a product of the two signals is the convolution of the Fourier transform.

$$\text{FT}[x(t) * y(t)] = X(f) \cdot Y(f) \quad (5.1)$$

$$\text{FT}[x(t) \cdot y(t)] = X(f) * Y(f) \quad (5.2)$$

Note that $X(f) = \text{FT}[x(t)]$ and $Y(f) = \text{FT}[y(t)]$ where FT stands for operation of the Fourier transform.

Proof. (for (5.1)) The convolution of the two signals $x(t)$ and $y(t)$ is defined by;

$$x(t) * y(t) = \int_{-\infty}^{\infty} x(\tau)y(t - \tau) d\tau. \quad (5.3)$$

Suppose the convolution of $x(t)$ and $y(t)$ is Fourier transformable. Then the Fourier transform of their convolution will be;

$$\begin{aligned} \text{FT}[x(t) * y(t)] &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} x(\tau)y(t - \tau) d\tau \right] e^{-j2\pi ft} dt \\ &= \int_{-\infty}^{\infty} x(\tau) \int_{-\infty}^{\infty} y(t - \tau)e^{-j2\pi ft} dt d\tau \\ &= \int_{-\infty}^{\infty} x(\tau) \int_{-\infty}^{\infty} y(t)e^{-j2\pi f(t+\tau)} dt d\tau \quad (\text{change of variables}) \\ &= \int_{-\infty}^{\infty} x(\tau)e^{-j2\pi f\tau} \int_{-\infty}^{\infty} y(t)e^{-j2\pi ft} dt d\tau \\ &= \int_{-\infty}^{\infty} x(\tau)e^{-j2\pi f\tau} Y(f) d\tau \\ &= X(f) \cdot Y(f). \end{aligned}$$

□

This can also be applied to the Discrete Fourier transform [5]. A sampled signal is generated by multiplying a continuous input signal with a comb function. Now multiplication in the time domain results in convolution in the frequency domain. This means that the original input signal's frequency spectra would be slid across the spectra of the comb function and summed up.

In Figure 5.3, we summarize the above relationship.

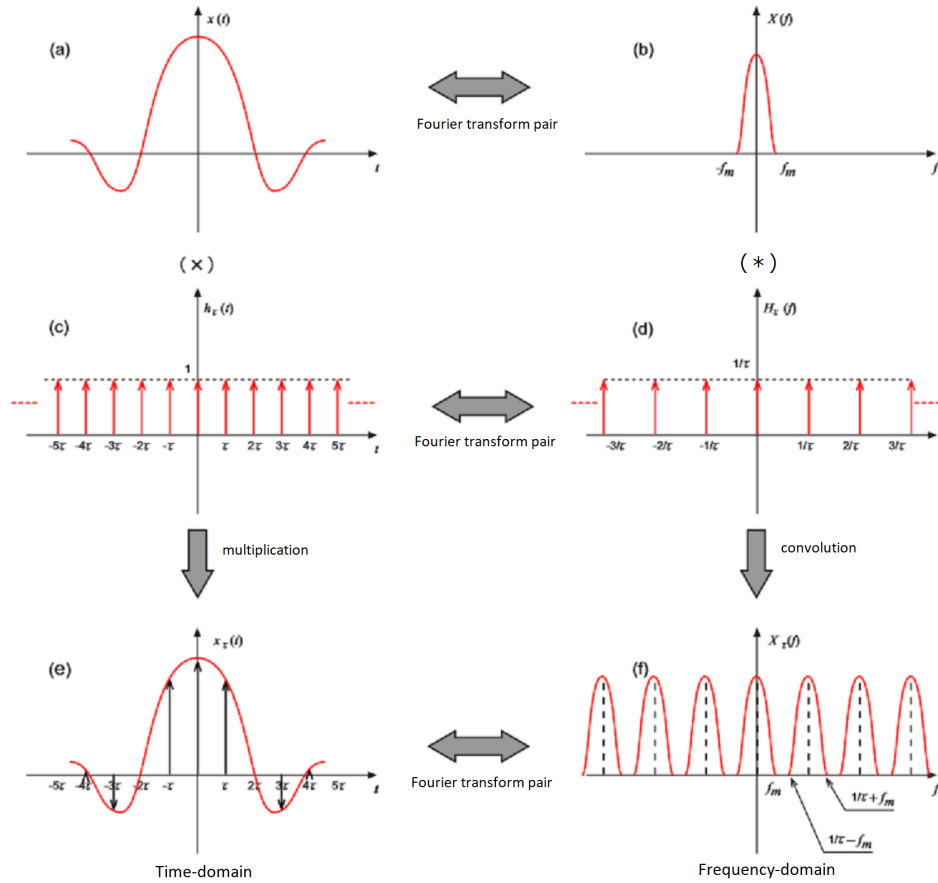


Figure 5.3: (a): a continuous input signal $x(t)$, (b): FT of (a), (c): a comb function $h_\tau(t) = \sum_{-\infty}^{\infty} \delta(t - n\tau)$, (d): FT of (c), (e): discrete time-signal $x_\tau(t)$, (f): FT of (e). Figure adapted from [34].

f_m is **Bandwidth** which is the difference between the upper and lower frequencies. (a)(b), (c)(d) and (e)(f) are a Fourier transform pair. To derive (d), we use the Fourier coefficients of $h_\tau(t)$. Since (c) is a periodic function which $\delta(t)$ is repeated by period τ , we can write;

$$C_k = \frac{1}{\tau} \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} \delta(t) e^{-j\frac{2\pi}{\tau}kt} dt = \frac{1}{\tau}. \quad (5.4)$$

So we can rewrite $h_\tau(t)$ as Fourier series;

$$h_\tau(t) = \frac{1}{\tau} \sum_{k=-\infty}^{\infty} e^{j\frac{2\pi}{\tau}kt}. \quad (5.5)$$

The Fourier transform of (5.5) is then,

$$\begin{aligned}
 H_\tau(f) &= \int_{-\infty}^{\infty} \left(\frac{1}{\tau} \sum_{k=-\infty}^{\infty} e^{j\frac{2\pi}{\tau}kt} \right) e^{-j2\pi ft} dt \\
 &= \frac{1}{\tau} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-j2\pi(f-\frac{k}{\tau})t} dt \\
 &= \frac{1}{\tau} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{\tau}\right)
 \end{aligned} \tag{5.6}$$

This shows that the Fourier transform of a comb function with a period τ becomes a comb function with a period $\frac{1}{\tau}$.

Using (5.3) to calculate the convolution of (b) and (d), we get the Fourier transform of multiplication (a) and (c). Thus, (f) is a periodic function repeating with sampling frequency $\frac{1}{\tau}$.

This is an example that maximum frequency f_m is smaller than sampling frequency $\frac{1}{\tau}$. Let us think about when the case that f_m is larger than sampling frequency $\frac{1}{\tau}$.

5.3.3 Frequency Domain Aliasing

Imagine we would sample less often which means the comb function (c) becomes a larger period than τ . Then the period of (d) is smaller than $\frac{1}{\tau}$. When sampling frequency $\frac{1}{\tau}$ becomes smaller than f_m , then the curves in the convolution begin to overlap like Figure 5.4. This is called a **frequency domain aliasing**.

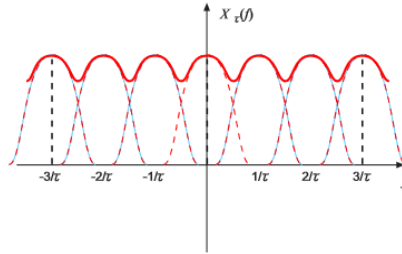


Figure 5.4: Aliasing in the frequency domain. Figure reproduced from [34].

Basically, aliasing depends on the sampling frequency and frequency content of the signal. In order to avoid this overlapping, we have to assume $f_m \leq \frac{1}{\tau} - f_m$ which is the same as to choose a sampling frequency satisfying;

$$\frac{1}{\tau} \geq 2f_m. \tag{5.7}$$

This is called the sampling theorem. We are trying to give maximum information about the original signal with the least amount of data.

5.3.4 Sampling Theorem

Theorem 5.3.2 (Nyquist-Shannon Sampling theorem). The original signal can be reconstructed when the sampling frequency f_s is greater than twice the maximum frequency f_{max} of the signal being sampled;

$$f_s > 2f_{max}. \quad (5.8)$$

We call the Nyquist frequency $f_n = \frac{f_s}{2}$ which is a half of the sampling frequency. As long as the Nyquist frequency exceeds the maximum frequency of the signal being sampled, the original analog signal can be reconstructed without loss.

Consequently, we have to set f_s greater than $2f_{max}$. However in many cases, we do not know f_{max} of the sound. Therefore, in sound analysis, f_{max} is set to be twice or more than 20kHz which is the maximum frequency of human hearing. For example, the sampling frequency is set to 44.1kHz in a music CD. Therefore, we also use the 44.1kHz as sampling frequency all the time.

5.4 Spectral Analysis with Audacity

In Audacity, there is a function in which you can view any sound as a *spectrum*. Spectrum converts a selected sound to a graph of amplitudes in the vertical axis against frequencies(Hz) in the horizontal axis. *Decibel (dB)* is the unit to measure the intensity of a sound and it is used in most cases.

$$\text{dB} = 20 \log_{10}(\text{absolute value of FFT result}).$$

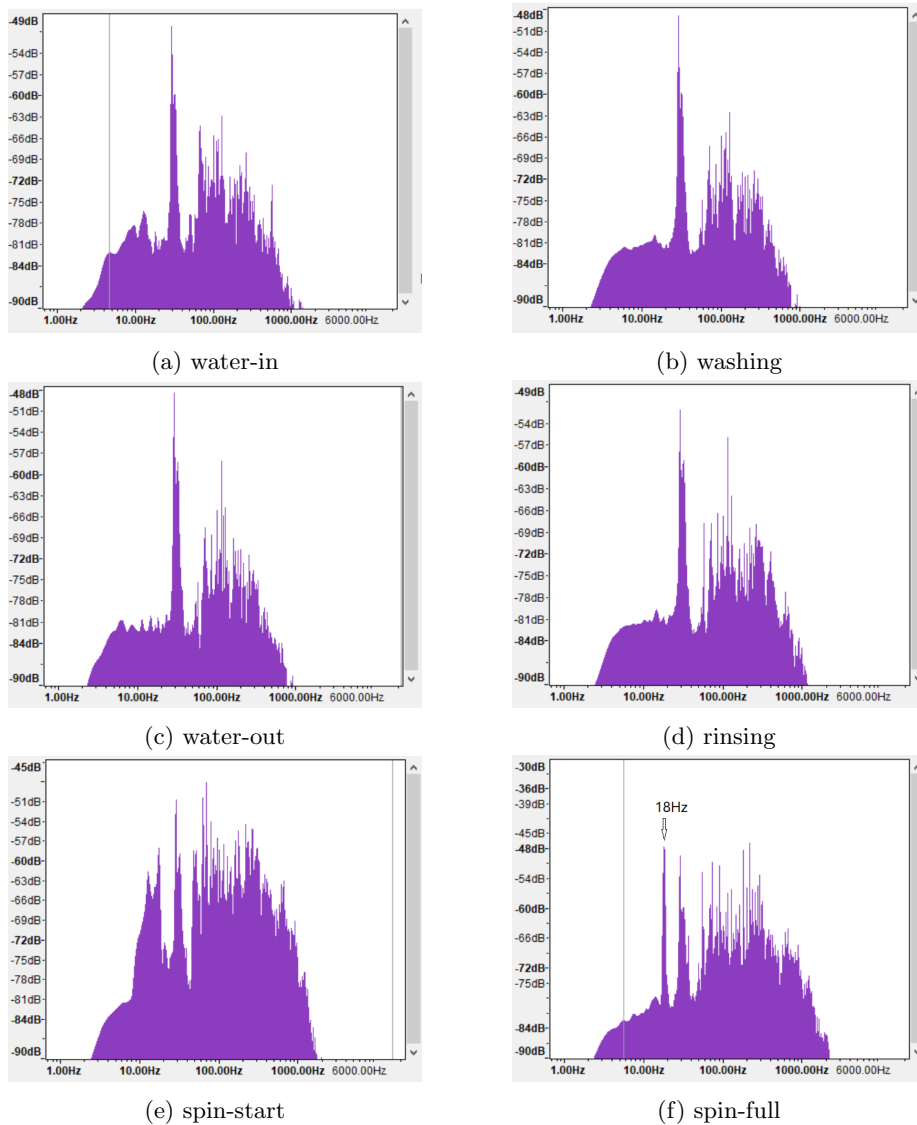


Figure 5.5: Spectrum view by Audacity

Note that each stage has a different time duration. The "washing" stage is very long about 20 minutes on the other hand, the "water-in" stage is about 10 seconds. Furthermore, we have to speak about Figure 5.5 and its presentation. We confirmed the 18Hz frequency in the stage 'spin-full' (f). However this spectrum view is not very practical to find features of the other stages because sounds are varying over time and this spectrum is the result of DFT i.e., all frequencies that have appeared over a whole stage.

There is another way to visualize sounds in Audacity. This is called *spectrogram*. Spectrogram shows how the energy in different frequency bins changes over time. The vertical axis shows frequencies(Hz) and the horizontal axis shows time(s). This is the result of the Short-time Fourier transform (STFT). We will take up STFT in the next section. Here you can see only a few colors but there are six color bands in the Audacity spectrogram view and the brightness of colors shows the strength of the sound (amplitude). This spectrogram view is analyzed by using the same clips as in the above spectrum analysis.

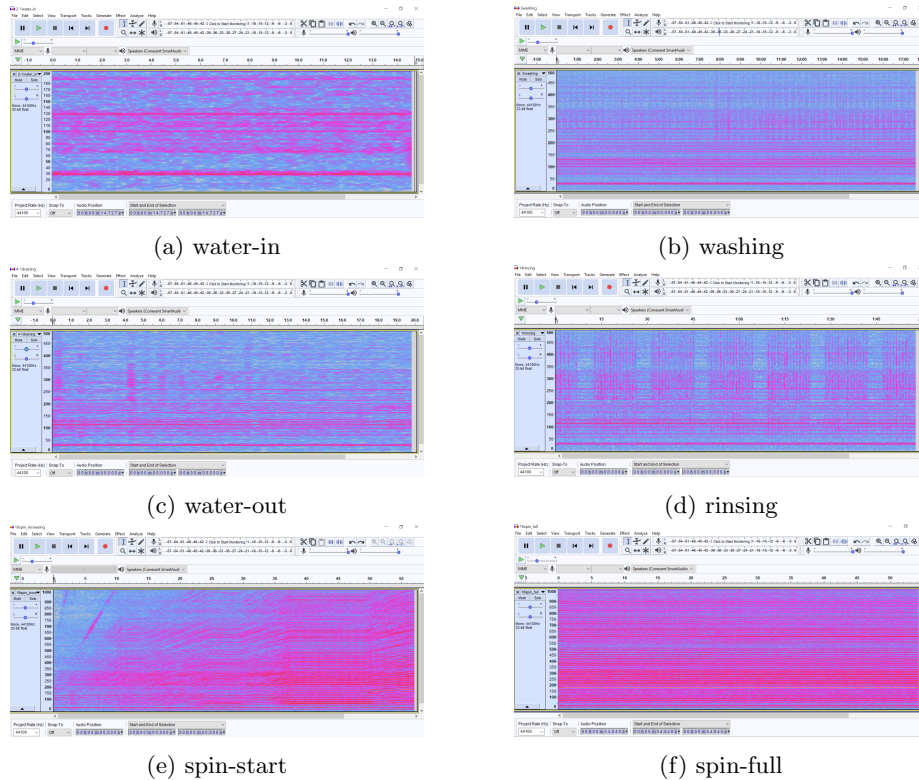


Figure 5.6: Spectrogram view by Audacity

We can almost identify every stages by the spectrogram. However our goal is to detect a stage from a short sound clip. "Spin-full" has a clear feature, which is evident from the spectrogram and the spectrum analysis. Even though we split the clip of the "spin-full" smaller, we will be able to capture the feature. How about the other stages? It depends on the time duration. At least what we understand from the spectral analysis, we need to use the short-time Fourier transform (STFT) to classify the stages.

5.5 Short-Time Fourier Transform

For example, an unexpected squeaking sound may be generated at a certain moment with a combination of a pair of gears. Even if we could capture the sound signal containing this squeaking sound, if we do FFT over the whole signal, the spectrum of the squeaking sound will be buried by other frequency components and it probably can not be determined. Also, even if we observe the original input signal in waveform, it is very difficult to grasp exactly at which time the squeaking noise occurred.

For such sound analysis, short-time Fourier transform (STFT) is the solution. Our implementation of STFT can be found in Appendix.

$$X_l[k] = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} w[n]x[n + Hl]e^{-j\frac{2\pi nk}{N}}, \quad l = 0, 1, \dots \quad (5.9)$$

where

w : window function

l : frame number

H : hop size.

This is the Short-time Fourier Transform equation. Basically we apply the FFT over short periods of time with a sliding window.

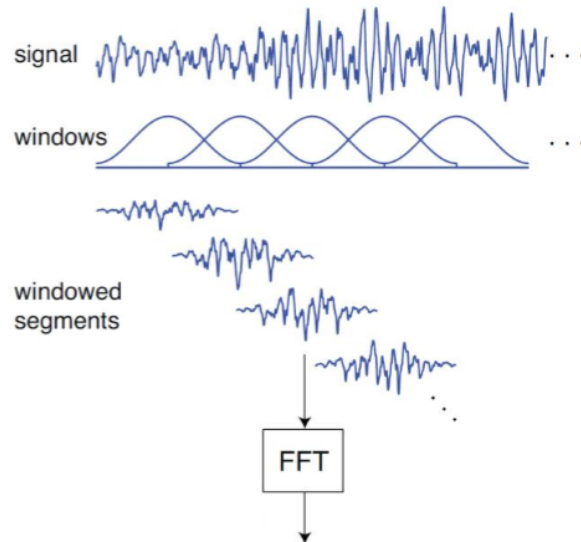


Figure 5.7: The image of STFT. Figure reproduced from [21].

The reason for using a window function is that the Fourier transform assumes that the signal is a periodic function which means the two endpoints of the cutout signal are interpreted as though as if they were connected together. However, if you cut out an original input signal at random, the endpoints do not connect well which makes the effects of the leakage (see Figure 5.8) and frequency components not present in the original input signal appear. By applying a window function, the endpoints can be smoothly connected to form a periodic function and it can reduce the leakage therefore frequencies are now displayed more correctly [7].

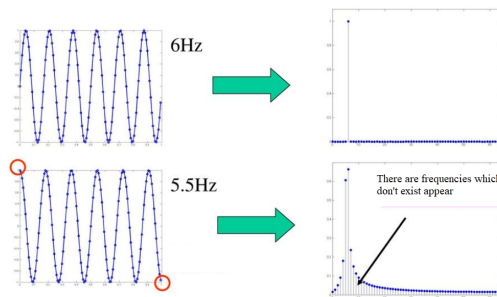


Figure 5.8: The endpoints are the same position (topleft), The endpoints are the different position (downleft) so the leakage appears in the frequency domain. Figure adapted from [38].

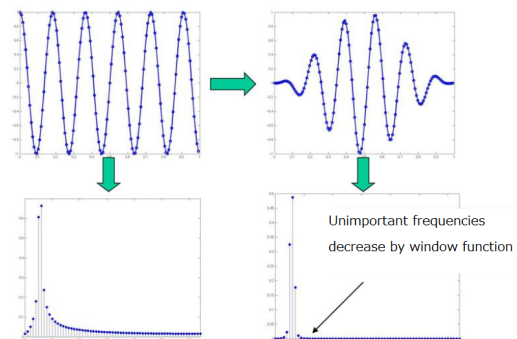


Figure 5.9: The frequencies not present in the original input signal decrease when a window function is applied (right). Figure adapted from [38].

5.6 Window Functions

There are several different types of window functions.

For selecting an appropriate window function you need to know the estimate frequency content of the signal or you need to know which frequency should be emphasized. Each window function has its own characteristics and suitability for different applications. Here, we introduce popular window functions [36] and their spectrum.

(1) The Rectangular window:

From the definition, it is 1 for the sample element n from 0 to $N - 1$ and all 0 for the others. It is equivalent to simply sampling N samples from the waveform. Frequency resolution of the main component is good, however, if no consideration is given to the discontinuity at both ends of the section, the spectrum will become widely spread as shown in Figure 5.10. This means it is not suitable for detecting the spectrum of low frequencies.

$$w(n) = \begin{cases} 1, & 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \quad (5.10)$$

(2) The Hanning window:

The characteristic of this window gently gets smaller toward both endpoints. Since it becomes zero, even if the original waveform is discontinuous, the values at both ends become zero. There is also a disadvantage that the endpoints of the original signal component are not reflected in the spectrum.

$$w(n) = \begin{cases} 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right), & 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \quad (5.11)$$

(3) The Hamming window:

This is an improvement to the Hanning window. The shape is exactly the same, but both ends do not become zero. Therefore, a small amount of discontinuity remains.

$$w(n) = \begin{cases} 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right), & 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \quad (5.12)$$

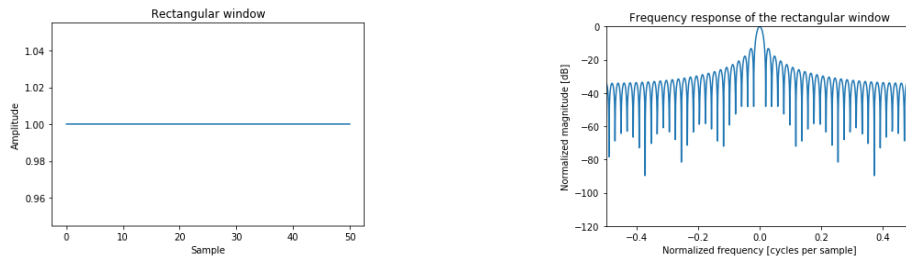


Figure 5.10: The Rectangular window

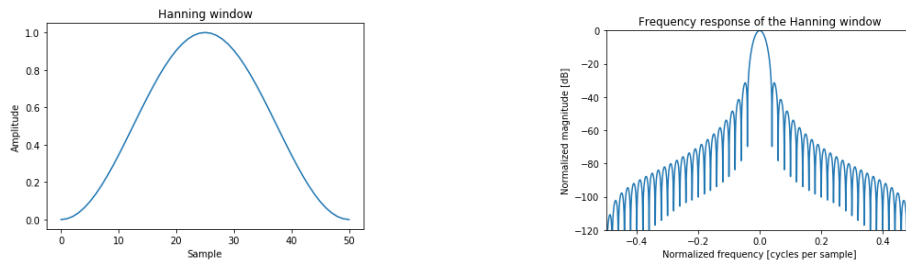


Figure 5.11: The Hanning window

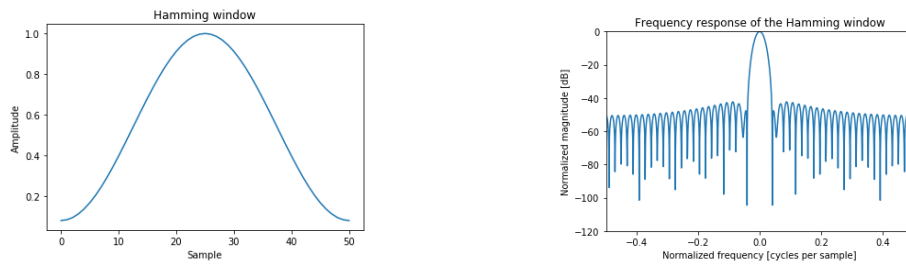


Figure 5.12: The Hamming window

In our project, we use the Hamming window which is popular for audio signal processing.

5.7 Feature Extraction

Our washing machine sounds are positioned as environmental sounds in this classification scheme. Here we have not just artificial sound but also noise in our samples.

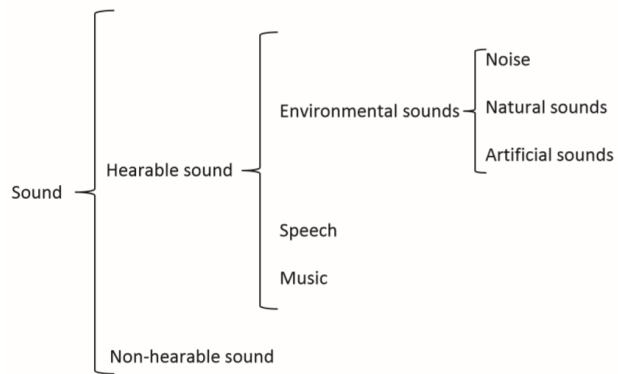


Figure 5.13: General sound classification scheme. Figure reproduced from [14].

Although we have tried spectrogram and spectrum, there are many audio feature extract techniques in signal processing. In this section, we introduces a few more methods. For the interested reader we recommend the article [14] which provides a broad view of the existing approaches to audio feature extraction. First, here are the group categories for feature extraction techniques in Figure 5.14.

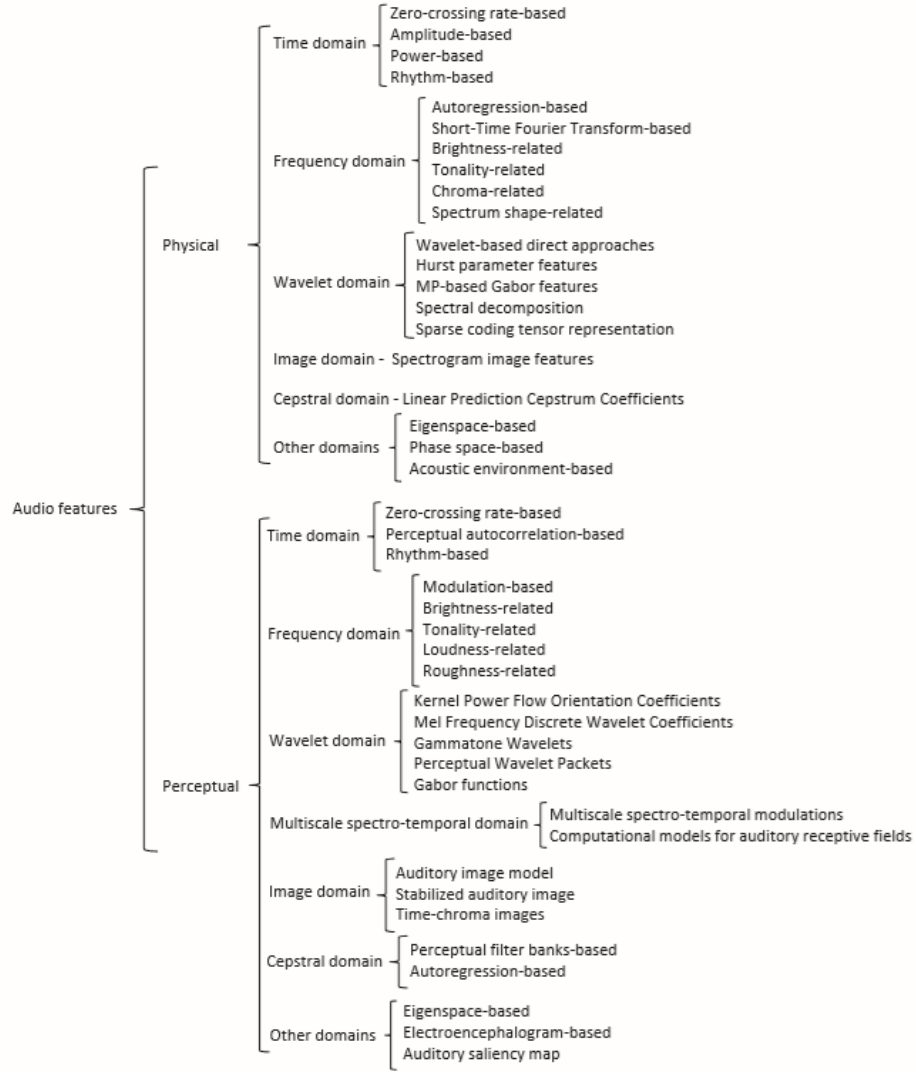


Figure 5.14: Taxonomy of audio features extraction techniques. Figure reproduced from [14].

For example, *Zero-crossing rate (ZCR)* is widely used in speech and music discrimination/classification and environmental sound recognition [14]. ZCR is one of Zero-crossing rate-based methods and analyzed in the time domain.

$$\text{ZCR} = \frac{1}{M-1} \sum_{m=0}^{M-1} |\text{sign}(x(m)) - \text{sign}(x(m-1))| \quad (5.13)$$

where M is the total number of samples in a processing window, sign is 1 for positive arguments and 0 for negative arguments. $x(m)$ is the magnitude of m -th sample of the wave form. The algorithm is very simple since only the sign change of the wave form is counted. However, we couldn't find any useful result from ZCR in our case.

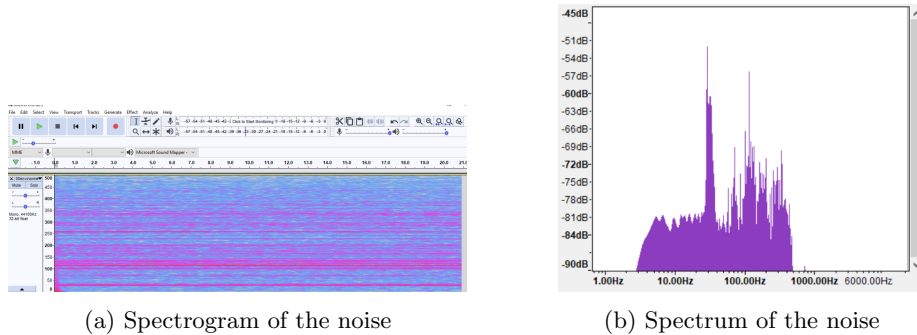
Another method that we tried is *Short-time-energy(STE)*. This is one of the power-based methods. It analyses also in the time domain or even in the frequency domain.

$$STE = \sum_{m=0}^{M-1} x^2(m) \quad (5.14)$$

This can be computed as the average energy per processing window. It may be more suitable for detecting voice/silence. Our samples have sounds constantly and the sound volume doesn't change much except during the spinning stage. So this doesn't give a good result for our case either.

5.8 Problems

In Section 5.2, we already mentioned the noise of the laundry room. Here is the plot of the spectrum and the spectrogram of the noise. All electrical switches that we are allowed to push are off.



The spectrum (b) has a very similar form to the other stages in Figure 5.5 (except the spinning stage). We may take away the remarkable frequencies, e.g., around 30Hz but it is still difficult to extract features in other stages. This is because the sound of the noise is also changing with time.

Now we step back and see the differences of sound types which are speech, music and environmental sound.

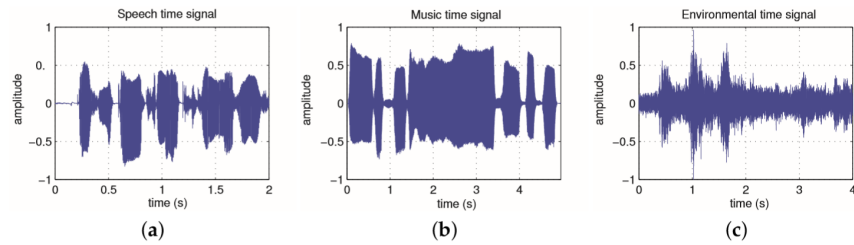


Figure 5.16: Waveform in time domain:(a)speech signal, (b)music signal (trumpet), (c)environment sound signal (traffic street). Figure reproduced from [14].

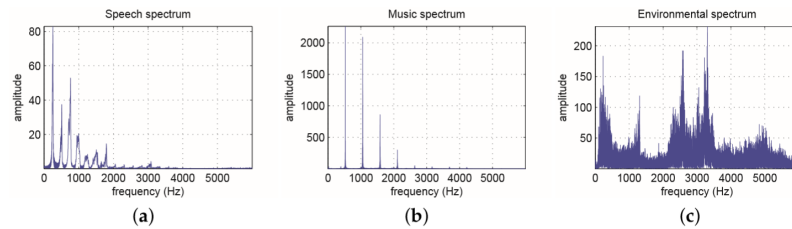


Figure 5.17: Frequency domain for extracted region of Figure 5.16 :(a)speech signal, (b)music signal (trumpet), (c)environment sound signal (traffic street). Figure reproduced from [14].

Despite what is shown in the figures, there is a certain periodicity in the time domain for music and speech signals so harmonic structures can be presented in their spectrum. On the other hand, the spectrum of the environmental sound is extremely complex, i.e., not periodic. Moreover, speech and music are composed of a limited dictionary of sound units i.e., phonemes and musical notes [14]. They also have some rules e.g., grammar, combination of tones and duration. On the contrary, most environmental sounds can be irregular and unpredictable since any sounds except music and speech can be included in this category.

From these points of view, analyzing a washing machine sound specially with significant unwanted noise is very challenging.

The difficulties of extracting features of our washing machine sounds include:

- * the sounds & the unwanted sound (noise) vary by time
- * the quality of the sound is not good
- * getting the same sound every time is impossible

- * the range of most frequencies are unknown

- * multiple actions occur at the same time for example, in the 'water-in' stage, the drum is also rotating while filling water. This makes it difficult to extract features of the stages.

Note also noises are not necessarily coming only from the pipes. This also occurs when any parts come into contact with vibration of the washing machine parts e.g., motors or coolant systems [15].

Chapter 6

To Machine Learning

Our goal is to classify sounds of our washing program by features of the frequencies and to detect a stage from a short sound clip. We tried several feature extracting methods to separate our sound data by the stages but we haven't found any good method yet. As we have shown in the previous section, there are many obstacles to extract features. However, *machine learning* may be possible to classify our data. Machine learning is one of the studies in artificial intelligence and it is a technique and method by which a computer realizes the learning ability of a human being. Examples of this includes, examining, calculating, predicting and judging without teaching explicitly. Machine learning is using algorithms to be able to extract features from data and represent them in a model. Then, we can use the model to infer things about other data which we have not modeled yet [8].

For example, regard to the task of detecting a car. Conventionally, we humans first decide features which are considered important for detecting cars e.g., tires or windshields and then make a model so that a computer can detect cars. However, machine learning automatically finds such features (patterns) from the data. We don't need to teach features to a computer.

We introduce some use of Machine learning for example, virtual personal assistants e.g., Siri or Google. Another example of this is classifying spam mails or product recommendations for you. Machine learning can be found everywhere nowadays.

Supervised learning is an algorithm of machine learning in which both input and desired output (label) data are provided at the same time for classification. We use this algorithm and create a model to identify the stages of our washing machine sounds. We are going to explain the steps as follows.

1. Outline of Deep neural network
2. Keras
3. Environment

4. Deep learning with Keras
5. Random search for hyperparameters
6. Results

6.1 Outline of Deep Neural Network

Neural network is one of the research topics in machine learning. It is a system inspired by the function of neurons in the human brain. *Deep learning* is a method for classification or regression problems using a model called a *deep neural network*. Although we have heard of neural networks vigorously in recent years, the idea itself existed from the 1950s [28]. Figure 6.1 shows an example of a *perceptron*¹ which is an algorithm that receives multiple signals as input and outputs one signal.

Each circle in Figure 6.1 is called a *neuron* (it is also called a *node* or a *unit*). Both input signals are multiplied with their weights (w_1, w_2) and summed up. We call the summation a weighted sum. If the weighted sum is larger than a threshold θ , it outputs 1 (it can be said that the neuron is firing). If the weighted sum is smaller than θ , it gives 0.

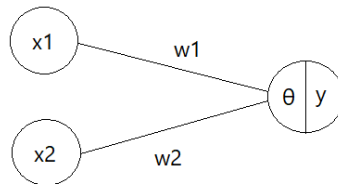


Figure 6.1: A simple perceptron with two input units: x_1, x_2 , two weight parameters: w_1, w_2 , a threshold: θ , an output: y .

Using a mathematical expression, we can write;

$$y = \begin{cases} 0, & x_1 w_1 + x_2 w_2 \leq \theta \\ 1, & x_1 w_1 + x_2 w_2 > \theta. \end{cases} \quad (6.1)$$

The function used here is called a *step function* which is one type of *activation function*. We will come back to the activation function in later sections. This simple perceptron can solve only linearly separable problems. However, most interesting problems are not linearly separable like Figure 6.2.

¹It is also called a simple perceptron or an artificial neuron. However, in this thesis we use the perceptron for an algorithm which uses step function and an artificial neuron for using another activation functions.

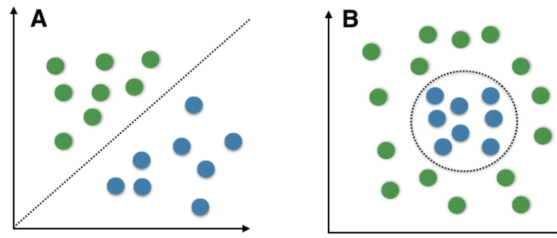


Figure 6.2: A : a linear problem, B: non-linear problems. Figure reproduced from [26].

In order to deal with complicated problems, layers are accumulated as shown in the next figure. Also output can become any decimal number between 0 to 1 instead of just binary values 0 and 1. This is due to an improvement of activation functions. Then, a simple perceptron which uses only a step function now is called a neural network when it uses another activation function. Often we refer to deep neural networks which consist of more than three *hidden layers* (layers between input- and output layer). In the Figure 6.3, the hidden layers are in blue color.

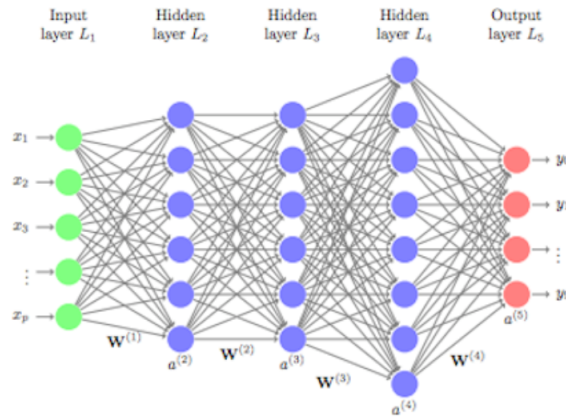


Figure 6.3: A multi-layered neural network. In this figure, biases are not considered. Figure reproduced from [24].

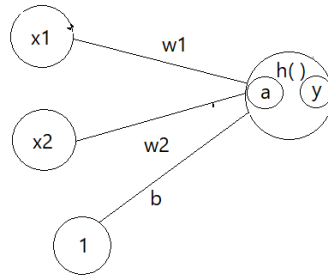


Figure 6.4: An artificial neuron with two input units: x_1, x_2 , two weight parameters: w_1, w_2 , a bias: b , a weighted sum: a , an activation function: h , an output: y .

Note that the idea of bias b comes from a threshold θ . Weights can only change the amount of steepness of the y . Bias help to shift the entire curve. Output y can be expressed;

$$\begin{cases} y = h(a) \\ a = x_1 w_1 + x_2 w_2 + b \end{cases} \quad (6.2)$$

where h is an activation function (except step function). We can choose one activation function per layer. Figure 6.4 has only 2 input signals but notice that input signal x is usually a vector or sometimes a matrix. In our project, we used 8192 input signals for the frequency data and 131072 input signals for the raw data.

Every input x is connected to every neuron in the next layer (Figure 6.3). This type of network is called a fully connected network. There are some variations but in our project, this fully connected (deep) neural network is used.

What Neural Networks (NN) do is, simply receive an input x , and then the neural network sets off a chain reaction and finally outputs a certain value y (such as a scalar or a vector). This output y is obtained as a probability distribution. A NN learning means to minimize a error function (error between the output y and the desired output i.e., label). The weights and the biases in the NN are adjusted in the direction to minimize the error function.

Deep neural network users have increased a lot because these methods give a result with high accuracy. Also supercomputers made it possible to extract insights from large, complex data sets.

6.2 Keras

TensorFlow is the most popular machine learning framework. It is an open source library for numerical computation that makes machine learning faster and easier. However TensorFlow interface is a bit challenging for a new user.

Keras is a neural networks library written in Python that wraps the efficient numerical libraries of TensorFlow, CNTK and Theano. By using Keras, you can implement relatively short code because all mathematical operations are already prepared. Therefore, Keras makes it possible to write code more intuitively and more concisely than using TensorFlow. That is why we chose to use Keras to build and train our model.

6.3 Environment

Here we introduce the tools what we set up for studying deep learning.

Python environment:

Anaconda

Jupyter Notebook

TensorFlow (including Keras)

NumPy

matplotlib

scikit-learn

PC hardware:

To perform machine learning or specially deep learning, having a fast GPU (Graphics Processing Unit) is very important. It is needed to perform a large amount of product sum operations in a short time. The improvement in processing speed by GPU is just too huge to ignore. So we use a computer with a NVIDIA TITAN X (Pascal) graphics card at Lund university.

Operating System:

Since we use the above GPU computer from our computer, we set up Ubuntu 18.04.1 LTS in our computer to be able to control the computations remotely.

6.4 Deep Learning with Keras

The flow of classification using deep learning is below.

1. Making dataset
2. Making a (deep) neural network model
3. Training the model by the given training dataset
4. Evaluating the model

6.4.1 Making the Dataset

It is said that the first thing you need for creating a classification model by deep learning is a large quantity of good quality data. Good quality data means the sample includes features of the *classes* ('the stages' in our case) which we want to classify. Furthermore, all samples need to be the same size. We have already split our sound track by stages but they need to be split further. Quantity is really depending on the sample. If you have good samples which reflect the features then we don't need much data. Obviously, if our samples are poor, we need lots of data. Here are the datasets (Table 6.1) which are prepared for the tutorials of Keras [16] and we will see the number of data which is used for each classification.

Table 6.1: The amount of datasets for multi-class classification. Table adapted from [25]

| Dataset | Contents | Class | Train data | Test data |
|---------------|--|-------|------------|-----------|
| MNIST | 28x28 Handwritten digit recognition | 10 | 60000 | 10000 |
| cifar10 | 32x32 Small color images classification | 10 | 50000 | 10000 |
| fashion-mnist | 28x28 Fashion articles images classification | 10 | 60000 | 10000 |
| imdb | Movie reviews sentiment classification | 2 | 25000 | 25000 |
| reuters | Newswire topics classification | 46 | 8972 | 2246 |

This is how we proceeded to make our datasets.

1. Extract features

Since we do supervised learning, the computer learns from labeled training data. In this way we give both input and label. For example, as the above MNIST case, we give a 2D-image of handwriting "9" and also a label that it states this is "9". The algorithm receives a 2D-image (input x) and the answer (label) at the same time, then the algorithm tries to find a pattern (function) which is mapping from the input to the output.

First, we make sound clips of each stage. The number of classes that we want to classify is five and the details are in Table 6.2.

The extraction of the sound clips is proceeded like this. We listen to the sound and keep only the part we recognize of the stages. However, all clips are cut even smaller, around 3 seconds each². These 3 seconds sound clips are difficult to distinguish.

2. Pre-processing

All the sound clips are read in Python and normalized to the range $[-1,1)$.

²To get benefits of FFT, we set the number of sample $N = 2^{17} = 131072$. Since the sampling frequency $f_s = 44100\text{Hz}$ which means 44100 points per second was taken for the sampling. So one sound clip becomes around 3 second($\frac{131072}{44100} \approx 2.972s$)

3. Apply FFT after a window function (STFT)

Each 3 seconds clip is applied to FFT after being multiplied by the Hamming window then the result i.e., the absolute value of the FFT are stored in an array. As we discussed in the sampling theorem, we use only half of the frequency bins i.e., 65536 frequency bins. We decided to take average of every 8 frequency bins so we can reduce the size. Now the FFT result of each clip is of the shape (1, 8192).

We also prepared an array for the label of the stages. For example, if the sound clip is a part of the "washing" stage, then we set the label as "1". The Python codes for making datasets are attached in the Appendix C. We also made a raw dataset. This is made by steps 1 and 2 without step 3. Each clip of the raw data is of the shape (1, 131072).

Table 6.2: The amount of datasets by the stages for both the frequency dataset and the raw dataset

| Label number | Stage | Train data | Test data |
|--------------|---------------|------------|-----------|
| 0 | spinning_full | 500 | 82 |
| 1 | washing | 500 | 99 |
| 2 | water_in | 500 | 68 |
| 3 | water_out | 500 | 105 |
| 4 | rinsing | 500 | 106 |

You may notice that our datasets are extremely small compared with MNIST datasets that contain 60,000 handwritten numerical images and 10,000 test images. It is better to have a large number of good quality data but sometimes it is difficult to collect a large number of data in a short time. In reality, there are many situations where it is necessary to analyze something with the data which we have now.

Import the datasets and shuffle.

```
(x_train) = np.loadtxt('x_train.dat')
(y_train) = np.loadtxt('y_train.dat')
(x_test) = np.loadtxt('x_test.dat')
(y_test) = np.loadtxt('y_test.dat')

x_train, y_train = shuffle(x_train, y_train)
x_test, y_test = shuffle(x_test, y_test)
```

Both x contains (sample, frequency) and both y contains (sample, label). There is no essential difference between training data and test data. But they shouldn't be the same. In case training data is in order by category, then it should be put in a random order by shuffle.

6.4.2 Making the Neural Network Model

```
model = Sequential()
model.add(Dense(128, input_dim=x_train.shape[1],
activation='relu',
kernel_regularizer=keras.regularizers.l2(1e-3)))
model.add(Dropout(0.2))

model.add(Dense(64, activation='tanh',
kernel_regularizer=keras.regularizers.l2(1e-4)))
model.add(Dropout(0.2))

model.add(Dense(5, activation='softmax',
kernel_regularizer=keras.regularizers.l2(1e-5)))
```

Here, we select a form of the model. In the above example, it is configured with an input layer, two hidden layers and an output layer. The first hidden layer has 128 nodes and uses 'relu' as the activation function. The second hidden layer has 64 nodes and uses 'tanh' as the activation function. The last layer has 5 nodes which is the number that you want to classify and uses 'softmax' as the activation function.

Below are some explanations of common words used in a classification model in Keras:

Batch size : The number of data to be passed at one iteration is called batch size. You divide the size of your samples by this number and send the portions in for training. It can be as big as the size of your data which means you are sending the whole data in at once.

Number of class : The number of the category that you want to classify.

Epoch number : In general, we will do learning several times using the same training data in order to improve the accuracy of the model. This is called iterative learning. The number of iterations is called epoch number.

Sequential : There are two ways to build Keras models, e.g., sequential and functional. Sequential models represent models connecting each layer of a neural network in order. We can add layers with `model.add()`.

Hidden layers : Layers of nodes between the input and output layers (Figure 6.3).

Dense : Densely connected layer. All nodes are connected to the nodes of the previous layer. We can decide the number of nodes in a hidden layer. This can be any number but it needs to be chosen as a suitable number.

Overfitting : The purpose of learning is to increase the accuracy of unknown (test) data. However it sometimes happens that the accuracy of the training data is very high but the accuracy of the test data is not good. This is

called overfitting. Overfitting refers to a model that fits the training data too well.

Dropout : One method to prevent overfitting and improve the accuracy of the model is called dropout. Using dropout, a part of the node is randomly deleted (overwritten with 0) at each update during training time, which helps prevent overfitting. Dropout(0.2) means 20 percent will be dropped out.

Regularization : In learning networks, the weights and the bias are updated automatically in an algorithm. Regularization keeps the weights small. There are many ways for regularization. The main purpose of using regularization is to control overfitting.

Activation function : To each neuron of the hidden layers, we apply a function called an activation function. There are several activation functions but all are non-linear functions. Here are some commonly used.

(1) sigmoid function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (6.3)$$

(2) tanh function

$$\text{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (6.4)$$

(3) relu(Rectified Linear Unit) function

$$\text{relu}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (6.5)$$

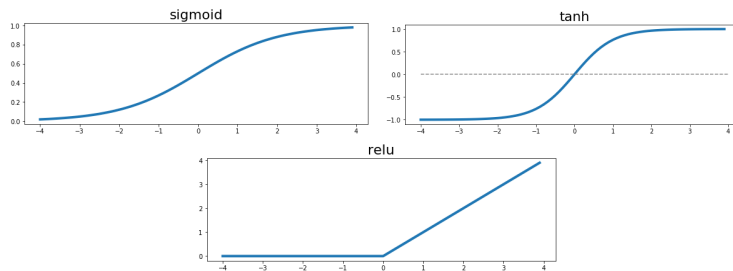


Figure 6.5: Activation functions

As a final activation function, the **softmax** function is often used for multi-class classification [13]. The softmax function returns an array of length 5 (in our case), with probability scores between 0 and 1 which all sum up to 1.

To make the model work in Keras we need to compile the model.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Loss function : This is a method of evaluating how much your neural network model does(not) match your dataset. It is showing the level of the performance of the neural network. The goal is to minimize this value. There are several loss functions. The common loss functions are, Mean Squared Error (MSE) and Cross Entropy Loss. We can't predict beforehand which will be best. It depends on your dataset and your problem.

Optimizer : This is how your neural network model updates the weights of every layer based on the data it sees and its loss function. Optimization algorithms help us to find a minimum value of a loss function. There are several optimizer functions as well. SGD, Adam, Adadelta and RMSProp are popular algorithms in the field of multi-class classification.

Metrics : Metric values can be recorded at the end of each epoch on the training/validation dataset. Here 'accuracy' is selected and displayed.

Backpropagation : This is an algorithm for training a neural network. If the output dose not match the label, each of the weights and the bias in the network are adjusted in the direction to minimize the value of the loss function.

Hyperparameters : Among the parameters, the parameters that humans adjust are called hyperparameters. All of the above mentioned e.g., activation function, number of hidden layers, rate of dropout (rate) and optimization function etc are hyperparameters. Hyperparameters can not be changed during training since they are defining the architecture of the network. Also learning can not be done correctly unless these are properly set. However there are no books that introduce the best hyperparameters for our datasets. Much experience or examination is required in order to find the optimal hyperparameter for our datasets.

6.4.3 Training the Model

Training the neural network model to call model.fit method.

```
model.fit(x_train, y_train,
          validation_split=0.3,
          verbose=1,
          epochs= 5,
          batch_size= 20 )
```

Validation_split : We split the training data to a learning data and a validation data. The validation dataset is used to give an estimate of model skill while tuning hyperparameters of the model. The number of the validation_split is how

many percent of the training data will be the validation data. In the code, the last 30% of the training data is used for verification.

Verbose : verbose=0 shows nothing. verbose=1 shows an animated progress bar like below:

```
Epoch 1/5  
60000/60000 [=====] - 34s 571us/step - loss: 0.2215 - acc: 0.9342  
Epoch 2/5  
60000/60000 [=====] - 33s 546us/step - loss: 0.0960 - acc: 0.9707  
Epoch 3/5  
60000/60000 [=====] - 32s 536us/step - loss: 0.0698 - acc: 0.9781  
Epoch 4/5  
60000/60000 [=====] - 33s 547us/step - loss: 0.0536 - acc: 0.9825  
Epoch 5/5  
60000/60000 [=====] - 32s 533us/step - loss: 0.0439 - acc: 0.9862
```

Figure 6.6: 5 epochs training and the result

As the model trains, the loss and accuracy metrics are displayed. This model reaches an accuracy of about 0.9862 (or 98.6%) on the training data.

6.4.4 Model Evaluation

Now we compare how the model performs on the other dataset i.e., the test dataset.

```
test_loss=model.evaluate(x_test,y_test , batch_size= 20)  
y_true=keras.utils.to_categorical(y_test)  
y_score=model.predict(x_test)  
p_score=label_ranking_average  
_precision_score(y_true , y_score)
```

By iterative learning, the value of the loss function often gradually declined. However, the value of the loss function is only for the training dataset. It is not certain whether we can demonstrate the same level of accuracy in new datasets. The goal is to make an algorithm which approximates so well using new input datasets that we predict the output as close as the label for that data. Here we explain how to evaluate a model.

| | | Prediction | |
|--------|----------|------------|----------|
| | | Positive | Negative |
| Actual | Positive | TP | FN |
| | Negative | FP | TN |

Figure 6.7: Confusion matrix. Figure reproduced from [37].

This is called confusion matrix and it is often used to describe the performance of a classification model. To evaluate a model, we compare the predicted output with the actual output (label). There are 4 patterns which are;

- TP (True-Positive) Predict (Yes) → Answer (Yes)
- FP (False-Positive) Predict (Yes) → Answer (No)
- FN (False-Negative) Predict (No) → Answer (Yes)
- TN (True-Negative) Predict (No) → Answer (No).

We introduce some basic measures that are computed from a confusion matrix for binary classification.

- Accuracy = all correct predictions/total predictions = $\frac{TP+TN}{ALL}$
- Error rate = incorrect predictions/total predictions = $\frac{FP+FN}{ALL}$
- Precision = correct positive predictions/total positive predictions = $\frac{TP}{TP+FP}$
- True positive rate(Recall) = $\frac{TP}{TP+FN}$ e.g., in a medical screen test used to identify a disease, the true positive rate is defined as the proportion of people with the disease who have a positive result.
- True negative rate = $\frac{TN}{TN+FP}$ e.g., in the above test, this is the proportion of people without the disease who will have a negative result.
- F1 score = $\frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$
F1 Score is the weighted average of Recall and Precision. Therefore, F1 score takes both FP and FN into account. Like in our case that we have

uneven class distributions, the F1 score is used more often than accuracy [27].

How to evaluate your model depends on what you are analyzing it for. Like the above medical screen test, if you don't have a disease but you get a positive result (FP), then you may need to go for another test to detect the disease but it is considered less of a problem than the case where you have a disease but you get a negative result (FN).

For our case, we first evaluate our model by the **mean reciprocal rank (MRR)** which is often used to evaluate a model for multi-class classification. MRR is calculated as;

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \quad (6.6)$$

where Q is the query (in our case, the sound samples) and rank_i refers to the rank of the correct answer. For example, after we trained a model with a training dataset, we evaluated the model with a test dataset which consisted of 5 sound samples of each stage. For Query 1, our model predicts stage 3 as the highest possibility and stage 2 as the second highest possibility and stage 0 (the correct answer) as the third highest possibility. Then the reciprocal rank will be $1/3$ for this query. Similarly, we calculate every reciprocal rank and sum up and divide by the number of the sound samples.

Table 6.3: Example for MRR

| Query | Proposed Results | Correct response | Rank | Reciprocal rank |
|-------|------------------|------------------|------|-----------------|
| 1 | 3,2,0,1,4 | 0 | 3 | 1/3 |
| 2 | 4,0,2,1,3 | 1 | 4 | 1/4 |
| 3 | 0,1,3,4,2 | 2 | 5 | 1/5 |
| 4 | 3,0,2,1,4 | 3 | 1 | 1 |
| 5 | 3,4,2,1,0 | 4 | 2 | 1/2 |

Given those five samples, we calculate the mean reciprocal rank as;

$$\text{MRR} = (\frac{1}{3} + \frac{1}{4} + \frac{1}{5} + 1 + \frac{1}{2})/5 \approx 0.4567 \quad (6.7)$$

The MRR values (p_score) of our best models are in Table 6.4. The result of the confusion matrix and the classification report for our best models can be found in Section 6.6.

6.5 Random Search for Hyperparameters

In the previous section, we have seen that a neural network adjusts parameters i.e., the weights and the bias. However hyperparameters i.e., all other parame-

ters can not be changed. We have to adjust them. Here we introduce one way of hyperparameter optimization called **random search**. Below is an example for making dictionaries.

```
'n_nod': sorted(np.random.choice(range(50,3500),
size = np.random.choice(range(3,6))), reverse= True)
'dropout': np.random.uniform(0,1.0),
'act_func': np.random.choice(['tanh', 'relu', 'sigmoid'])
```

Here we use NumPy. We let Python pick hyperparameters randomly from the dictionary that we prepared as above. Then we save the model which gives the good precision.

```
if precision_score > 0.63 :
    np.save('file_name', your_dictionary_name)
```

We prepared two kinds of datasets which are the frequency data (FFT result) and the raw data. The raw data is the normalized [-1, 1) sounds clips without applying FFT. Table 6.4 shows the best three models of the frequency data and the raw data from the random search. p_score is the precision that we got from the model.

Table 6.4: Best three hyperparameters from the random search

| Layer | Hyper | Frequency data | | | Raw data | | |
|---------|-----------|----------------|-----------|-----------|----------|---------|---------|
| Input | Nodes# | 8192 | 8192 | 8192 | 131072 | 131072 | 131072 |
| 1 | Dense | 1804 | 2213 | 2457 | 3097 | 3798 | 3097 |
| | lambd | 0.001 | 0.1 | 0.001 | 0.001 | 0.001 | 0.0001 |
| | act | relu | relu | relu | sigmoid | relu | sigmoid |
| | drop | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 2 | Dense | 815 | 2150 | 1733 | 2976 | 1578 | 2767 |
| | lambd | 0.01 | 0.1 | 0.001 | 0.001 | 0.1 | 0.001 |
| | act | relu | relu | relu | tanh | sigmoid | relu |
| | drop | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| 3 | Dense | 243 | 1908 | 588 | 233 | 180 | 1260 |
| | lambd | 0.01 | 0.01 | 0.01 | 0.01 | 0.0001 | 0.01 |
| | act | relu | tanh | relu | tanh | tanh | sigmoid |
| | drop | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| 4 | Dense | | 656 | 253 | | | |
| | lambd | | 0.01 | 0.1 | | | |
| | act | | relu | relu | | | |
| | drop | | 0 | 0 | | | |
| 5 | Dense | | 251 | | | | |
| | lambd | | 0.01 | | | | |
| | act | | tanh | | | | |
| | drop | | 0 | | | | |
| Output | Dense | 5 | 5 | 5 | 5 | 5 | 5 |
| | act | softmax | softmax | softmax | softmax | softmax | softmax |
| | optimizer | Adadelata | Adadelata | Adadelata | Adam | Adam | Adam |
| | lr | 0.0001 | 0.0001 | 0.00001 | 0.00001 | 0.0001 | 0.0001 |
| | Epochs | 20 | 60 | 60 | 80 | 60 | 80 |
| | Batch | 64 | 512 | 512 | 64 | 128 | 32 |
| p-score | | 0.60261 | 0.61181 | 0.65566 | 0.61101 | 0.62826 | 0.63043 |

There are several methods for tuning hyperparameters. We introduce another method that is called **grid search**. Grid search is different from the random search in the sense that we check every combination of hyperparameters. We may be able to find the best combination of hyperparameters which give the best accuracy. However if we use grid search, we need a lot of computational resources e.g., sometimes the computation time takes a couple of months. On the other hand, random search might not give us the best hyperparameters however, it is a more efficient technique when we consider time and cost.

6.6 Results

This is the results of the confusion matrix using each best model.

Table 6.5: The frequency data

| | | Predicted | | | | |
|--------|---|-----------|----|---|-----|---|
| | | 0 | 1 | 2 | 3 | 4 |
| Actual | 0 | 18 | 64 | 0 | 0 | 0 |
| | 1 | 0 | 1 | 0 | 98 | 0 |
| | 2 | 0 | 4 | 0 | 64 | 0 |
| | 3 | 0 | 2 | 0 | 103 | 0 |
| | 4 | 0 | 1 | 0 | 105 | 0 |

Table 6.6: The raw data

| | | Predicted | | | | |
|--------|---|-----------|---|---|-----|---|
| | | 0 | 1 | 2 | 3 | 4 |
| Actual | 0 | 82 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 99 | 0 |
| | 2 | 0 | 0 | 0 | 68 | 0 |
| | 3 | 0 | 0 | 0 | 105 | 0 |
| | 4 | 0 | 0 | 0 | 106 | 0 |

0: spinning_full

1: washing

2: water_in

3: water_out

4: rinsing

It is clear that the learning was not done well for either case. Almost every sample was predicted as the stage 'water_out' except the 'spinning_full' stage. Below is the result of the classification report.

Table 6.7: The frequency data

| | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| 0 | 1.00 | 0.22 | 0.36 | 82 |
| 1 | 0.01 | 0.01 | 0.01 | 99 |
| 2 | 0.00 | 0.00 | 0.00 | 68 |
| 3 | 0.28 | 0.98 | 0.43 | 105 |
| 4 | 0.00 | 0.00 | 0.00 | 106 |

Table 6.8: The raw data

| | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 82 |
| 1 | 0.00 | 0.00 | 0.00 | 99 |
| 2 | 0.00 | 0.00 | 0.00 | 68 |
| 3 | 0.28 | 1.00 | 0.43 | 105 |
| 4 | 0.00 | 0.00 | 0.00 | 106 |

where the support is the number of samples of the true response that lie in that class.

Chapter 7

Conclusion

Our goal was to classify sounds of the five stages of the washing program using short sound clips we recorded ourselves. We didn't use 'between' sounds and other complex sounds because we realized that these seemed very difficult to classify. As the purpose of this thesis is more of a study rather than the development of a robust software tool or a mobile phone application, we did not invest more time on this certainly important issue. Here, we discuss the result and point out ways for future improvements.

As we expected, it is possible to predict the 'spinning_full' stage from the raw data (see Table 6.6). Using the frequency data from the 'spinning_full' stage, some sound samples of this stage could also be identified correctly. This is because of the well-defined frequency generated by the long rotation at constant speed. However, almost all the other sound clips were predicted as the 'water_out' stage even though we excluded the above complex sounds. The classification by machine learning was used in this thesis for prototyping reasons but undoubtedly, for more reliable results we would have needed larger datasets and more advanced models to extract significant information from the noise often coming from the laundry room. On the data acquisition side, we can not be sure, that a tiny \$10 USB microphone is fully sufficient for this task.

Additionally, the 'spinning_full' stage has a clear feature and it can already be identified. Therefore, we could do machine learning without this stage. Or if we use 'noise' sound clips then we may be able to detect that the washing program is finished.

How about the size of the sound clip? We decided to choose a sound clip of around 3 seconds to make sure that it is sufficiently long to contain significant sound features needed to identify a particular washing stage. Even with the human ear, it is sometimes difficult to recognize the stage of a sound which is shorter than 3 seconds. Also we would like to have the number of the sample points in the sound clip as a power of 2 in order to get benefits of FFT. The next

largest size would be around 6 seconds. This sound clip may have contained more features of the stage, however it would have taken much more time to collect the data. Additionally, if we had used such large samples, we would have needed a huge memory space in the computer. We note here that the frequency resolution of a DFT is defined as $\frac{f_s}{N}$. Therefore if we were to choose a large sound clip(=large sample points N), then the frequency resolution would have become smaller. We used $N = 131072$ and took the average of every 8 frequency bins so our frequency resolution was approximate 2.69Hz and our input dimension was 8192. We don't want the frequency resolution to be smaller because that would have led to a larger input dimension. That is another reason why we didn't use a 6 second sound clip. However, to solve this problem, there is a method called **Mel Frequency Cepstral Coefficients (MFCCs)** which can reduce the dimension without causing much damage to the information. It would be a nice way to extend this project.

We are very satisfied with our study. We learned many things throughout this thesis project such as Fourier transform, feature extraction methods, digital signal processing and classification with machine learning. We can find a myriad of things that oscillate in this universe. This means that all these things can be expressed by waveforms, and we can apply FFT and analyze them by their frequencies. Movement of a comet, exchange rates and our brain waves are just a few examples!

Also, we learned the basics for classification by deep learning. When we take in signals into a computer, features such as colors, strength and positions etc, are converted to numbers. This means we can apply this classification by deep learning, even though the input signal is not frequency. Our only requirement is that the signal contains some features.

This study is incredibly practical and all these concepts are used widely throughout the world. Therefore we are hopeful that this thesis is an ideal entry point for future work for someone who starts to study mathematics and physics in university.

Appendix A

Dirichlet Condition

Johann Peter Gustav Lejeune Dirichlet (1805-1859) [17] derived these conditions [18] and they are called the Dirichlet conditions.

1. $f(t)$ is periodic and single-valued everywhere.

We say $f(t)$ is a periodic function when a function $f(t)$ is such that;

$$f(t + T) = f(t)$$

where T is a period.

A single-valued function means that the function is One to One which means every element of the range of the function corresponds to exactly one element of the domain.

2. $f(t)$ has a finite number of discontinuities in one period and the discontinuities must be finite.
3. $f(t)$ has a finite number of extrema in one period.
4. the integral over one period must be less than ∞ .

$$\int_T^{T+t} |f(t)| dt < \infty, \quad \forall t.$$

Appendix B

Python Code - STFT

```
import wave
from scipy.fftpack import fft
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import get_window

def stft(filename='01 water_in_t', answer=2, window='hamming', M=131072,
N=131072, H=65536):

    wavf = filename + '.wav'
    wr = wave.open(wavf, 'r')

    # information of the wav file
    ch, sampwidth, fr, fn, -, - = wr.getparams()
    total_time = fn / fr

    #transform to ndarray from bites
    data_string = wr.readframes(fn)
    wr.close()
    # transform to ndarray , normalized on [-1,1)
    if sampwidth == 2:
        data = np.frombuffer(data_string, dtype='int16')
        x = data / 32768.0
    elif sampwidth == 4:
        data = np.frombuffer(data_string, dtype='int32')
        x = data / 2147483648.0

    w = get_window(window, M, False) # window, M-size
    M = w.size # size of analysis window
    #for M is not N, we do zero-padding
    hM1 = (M+1)//2 # half analysis window size by rounding
    hM2 = M//2 # half analysis window size by floor
```

```

x = np.append(np.zeros(hM2), x)           # add zeros at beginning
x = np.append(x, np.zeros(hM2))         # add zeros at the end

pin = hM1                                # initialize sound pointer in middle
pend = x.size - hM1                       # last sample to start a frame
xmX = []                                  # Initialise empty list for mX

while pin <= pend:
    x1 = x[pin-hM1:pin+hM2]               # select one frame of input sound
    hN = (N//2)+1                         # size of positive spectrum
    w = w/sum(w)                          # normalize analys window
    xw = x1*w                             # window the input sound
    X = fft(xw)                            # compute FFT
    mX = abs(X[:hN])                      # compute absolute value of positive side

# xmX.append(np.array(mX))                # append output to list
pin += H                                  # advance sound pointer
change_shape = mX.reshape(8192,8)         #this is for making matrix smaller
after_mean = np.mean(change_shape, axis=1)
xmX.append(np.array(after_mean))          # append output to list

xmX = np.array(xmX)                       # convert to numpy array

return xmX, teach_answer

```

Appendix C

Python Code - Making Dataset

```
import numpy as np
import STFT
import collections

"""
0: spinning_full
1: washing
2: water_in
3: water_out(draining)
4: rincing
"""

names = ['7draining', '16spin_full', '2-1water_in', '3washing', '4-1draining', ...]
vals = [3, 0, 2, 1, 3, ...]

mat_list = []
ans_list = []

for name, val in zip(names, vals):
    mat, ans = STFT.stft(name, val)
    mat_list.append(mat)
    ans_list.append(ans)

x_train = np.vstack(mat_list)
y_train = np.vstack(ans_list)

rec=y_train.reshape(-1)
```

```

counting_train = collections.Counter(rec)
print(counting_train)

np.savetxt('x_train.dat', x_train)
np.savetxt('y_train.dat', y_train)

"""
making test_matrix and test_answer
"""

names3 = ['01water_in_t', '02washing_t', '03draining_t', '04water_in_t', ...]
vals3 = [2, 1, 3, 2, ...]

mat_list3 = []
ans_list3 = []

for name, val in zip(names3, vals3):
    mat3, ans3 = STFT.stft(name, val)
    mat_list3.append(mat3)
    ans_list3.append(ans3)

x_test = np.vstack(mat_list3)
y_test = np.vstack(ans_list3)
res=y_test.reshape(-1)
counting_test = collections.Counter(res)
print(counting_test)

np.savetxt('x_test.dat', x_test)
np.savetxt('y_test.dat', y_test)

```

Appendix D

Python Code - Random Search

```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Input
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
from keras import backend as K
import tensorflow as tf
from sklearn.utils import shuffle
from sklearn.metrics import label_ranking_average_precision_score
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

x_train = np.loadtxt('x_train.dat')
y_train = np.loadtxt('y_train.dat')

x_test = np.loadtxt('x_test.dat')
y_test = np.loadtxt('y_test.dat')

x_train, y_train = shuffle(x_train, y_train)
x_test, y_test = shuffle(x_test, y_test)

def pipeline(inp_dim,
             n_nod,
             drop = [],
             act_fun = 'relu',
             out_act_fun = 'sigmoid',
```



```

        opt_method = 'Adam',
        cost_fun = 'binary_crossentropy',
        lr_rate = 0.01,
        num_out = 5,
        lambd = 0.0,
        batch_size = 20,
        epochs = 10):

main_input = Input(shape=(inp_dim,), dtype='float32', name='main_input')

X = main_input
actfun_used=[]
lambd_used=[]

for j,nod in enumerate(n_nod):

    temp1=np.random.choice(INPUT['act_fun'])
    temp3=np.random.choice(INPUT['lambd'])

    X = Dense(nod,
               activation = temp1,
               kernel_regularizer=keras.regularizers.l2(temp3))(X)
    X = keras.layers.Dropout(drop[j])(X)
    actfun_used.append(temp1)
    lambd_used.append(temp3)
output = Dense(num_out, activation = out_act_fun)(X)

method = getattr(keras.optimizers, opt_method)

model = keras.models.Model(inputs=[main_input], outputs=[output])
model.compile(optimizer = method(lr = lr_rate),
              loss = cost_fun)

return model, actfun_used, lambd_used

for _ in range(1000):

INPUT = {'inp_dim': x_train.shape[1],
         'n_nod':sorted(np.random.choice(range(50,4000),
         size =np.random.choice(range(3,6))), reverse= True),
         'drop':[0.3, 0.2, 0.1, 0., 0., 0., 0., 0.],
         'act_fun': ['tanh', 'relu', 'sigmoid'],
         'out_act_fun': 'softmax',
         'opt_method': np.random.choice(['RMSprop', 'Adam', 'Adadelta']),
         'cost_fun': 'sparse_categorical_crossentropy',
         'lr_rate': np.random.choice([1e-1, 1e-2, 1e-3, 1e-4, 1e-5]),
         'num_out': 5 ,

```

```

        'lambd': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5],
        'batch_size': np.random.choice([64, 128, 256, 512]),
        'epochs': np.random.choice([20, 40, 60])}

model, actfun_used, lambd_used = pipeline(**INPUT)

es_cb=keras.callbacks.EarlyStopping(monitor='val_loss', patience=6,
verbose=1, mode='auto')
history = model.fit(x_train, y_train,
                    validation_split=0.3,
                    verbose=0,
                    epochs= INPUT['epochs'],
                    batch_size= INPUT['batch_size'],
                    callbacks=[es_cb])

test_loss = model.evaluate(x_test, y_test, batch_size= INPUT['batch_size'] )
print('Test_loss:', test_loss)

y_true = keras.utils.to_categorical(y_test)
y_pred = model.predict(x_test)
p_score = label_ranking_average_precision_score(y_true, y_pred)
print('precision_score', p_score)

if p_score > 0.65 :
    np.savez('great_hyper1.npz', INPUT, actfun_used, lambd_used, p_score)

target_names = ['0: spinning_full', '1: _washing', '2: _water_in', '3: _water_out',
'4: _rincing']
y_pred=np.argmax(y_pred, axis=1)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=target_names))

```

Bibliography

- [1] Transnational College of Lex, *Who Is Fourier? A Mathematical Adventure 2nd Edition*. U.S.A.: Language Research Foundation, 1997
- [2] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1997
- [3] Bernard Mulgrew, Peter Grant and John Thompson, *Digital signal processing, Concept & Applications* N.Y: Palgrave, 1999
- [4] Julius O. Smith III, *Mathematics of the discrete Fourier transform(DFT) with audio applications* 2nd ed. U.S.A.: W3K Publishing, 2008
- [5] Julius O. Smith III, *Spectral audio signal processing* U.S.A.: W3K Publishing, 2011
- [6] William L.Briggs, Van Emden Henson, *The DFT, An Owner's Manual for the Discrete Fourier Transform* U.S.A.: the Society for Industrial and Applied Mathematics, 1995
- [7] Richard G. Lyons, *Understanding Digital Signal processing* U.S.A.: Addison Wesley Longman, Inc., 1997
- [8] Josh Patterson and Adam Gibson, *Deep Learning, A Practitioner's Approach* U.S.A.: O'Reilly Media, Inc., 2017
- [9] Xavier Serra and Julius O Smith, III, *DFT of real sinusoids*, lecture notes. COURSERA, Audio Signal Processing for Music Applications, Universitat

- Pompeu Fabra of Barcelona & Stanford University, 2018. <https://www.coursera.org/learn/audio-signal-processing/home/info>
- [10] NATIONAL PROGRAMME ON TECHNOLOGY ENHANCED LEARNING, *The Convolution Theorem*, Module 2:Signals in Frequency Domain Lecture 18, lecture note. <https://nptel.ac.in/courses/117101055/downloads/Lec-18.pdf> (Accessed 20181215)
- [11] Wikipedia, *Paul Dirac*, https://en.wikipedia.org/wiki/Paul_Dirac (Accessed 20181226)
- [12] Electrolux, *Washing machine Service Manual* http://tools.professional.electrolux.com/Mirror/Doc/ELS/SMA/SM_438920181_W365-3300H_Exacta_EN.pdf (Accessed 20190108)
- [13] Stacey Ronaghan, *Deep Learning: Which Loss and Activation Functions should I use?*. <https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8> (Accessed 20190120)
- [14] Francesc Alías, Joan Claudi Socoró and Xavier Sevillano, *A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds*, Applied Sciences, MDPI, 2016 <https://www.mdpi.com/2076-3417/6/5/143> (Accessed 20190110)
- [15] Marcel Janda, Ondrej Vitek and Vitezslav Hajek, *Noise of Induction Machines*, IntechOpen Limited, 2012. <https://www.intechopen.com/books/induction-motors-modelling-and-control/noise-of-induction-machines> (Accessed 20190112)
- [16] TensorFlow, *Tutorials, Get Started with TensorFlow*, <https://www.tensorflow.org/tutorials/> (Accessed 20190112)
- [17] Wikipedia, *Johann Peter Gustav Lejeune Dirichlet*, https://sv.wikipedia.org/wiki/Johann_Peter_Gustav_Lejeune_Dirichlet (Accessed 20181201)
- [18] Neelu Kumari, *Notes on Dirichlet conditions in Fourier transform*, Our blog education. <https://blog.oureducation.in/dirichlet-conditions/> (Accessed 20181206)
- [19] Christophe, *How does Shazam work*, Coding Geek, A blog about IT, programming and Java, 2015. <http://coding-geek.com/how-shazam-works/> (Accessed 20181206)
- [20] Michael T. Heideman· Don H. Johnson c. Sidney Burrus *Gauss and the History of the Fast Fourier Transform*, IEEE ASSP MAGAZINE OCTOBER 1984. https://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf (Accessed 20190211)

- [21] Ricardo Gutierrez-Osuna, *L6: Short-time Fourier analysis and synthesis*, lecture notes. Introduction to Speech Processing, TEXAS AM UNIVERSITY, 2002. <http://research.cs.tamu.edu/prism/lectures/sp/16.pdf> (Accessed 20190122)
- [22] MIT OpenCourseWare, *Sampling and the Discrete Fourier Transform*, lecture notes. 2.161 Signal Processing, Continuous and Discrete, Fall 2008. <https://ocw.mit.edu/courses/mechanical-engineering/2-161-signal-processing-continuous-and-discrete-fall-2008/study-materials/samplingdft.pdf> (Accessed 20181211)
- [23] Mathuranathan Viswanathan, *How to Interpret FFT results - complex DFT, frequency bins and FFTShift*, GaussianWaves, 2015. <https://www.gaussianwaves.com/2015/11/interpreting-fft-results-complex-dft-frequency-bins-and-fftshift/> (Accessed 20190126)
- [24] UC Business Analytics R Programming Guide, *Feedforward Deep Learning Models*, http://uc-r.github.io/feedforward_DNN (Accessed 20190308)
- [25] Keras documentation, *Datasets*, <https://keras.io/datasets/> (Accessed 20190308)
- [26] Sebastian Raschka, *Naive Bayes and Text Classification - Introduction and Theory*, http://sebastianraschka.com/Articles/2014_naive_bayes_1.html (Accessed 20190313)
- [27] Renuka Joshi, *Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures*, <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/> (Accessed 20190314)
- [28] Koki Saitoh ゼロから作るDeepLearning、Pythonで学ぶディープラーニングの理論と実装, Japan: O'Reilly Japan, Inc. 2016
- [29] aidiary 人工知能に関する断創録, 離散フーリエ変換, <http://aidiary.hatenablog.com/entry/20110611/1307751369> (Accessed 20181212)
- [30] Mayuko Iwamoto, 第9回フーリエ変換と離散フーリエ変換(6月9日), lecture notes. Shimane University. <http://www.math.shimane-u.ac.jp/~miwamoto/2016mmm1/mmm1-9.html> (Accessed 20190125)
- [31] Naoto KOUYAMA, 第4章離散フーリエ変換, lecture notes. Toyama University. <https://kouyama.sci.u-toyama.ac.jp/main/education/2006/infomath/pdf/text/text10.pdf> (Accessed 20190301)
- [32] Akira Asano 2011年度秋学期 解析応用 第6回 第1部・フーリエ解析/離散フーリエ変換と高速フーリエ変換, lecture

- notes, 2011. <http://racco.mikeneko.jp/Kougi/2011a/AAN/2011aaan06.pdf> (Accessed 20190301)
- [33] PIC AVR 工作室, FFT・複素数入門, http://picavr.uunyan.com/warehouse_fft3.html (Accessed 20190211)
- [34] Onosokki 計測コラム $emm136$ 号
基礎からの周波数分析 (7) - 「時間信号のサンプリング」 https://www.onosokki.co.jp/HP-WK/eMM_back/emm136.pdf (Accessed 20190211)
- [35] Shingo Kagami,
やる夫で学ぶデジタル信号処理, Tohoku University. <http://www.ic.is.tohoku.ac.jp/~swk/lecture/yaruodsp/main.html> (Accessed 20181126)
- [36] ロジカルアーツ研究所, 窓関数を用いる理由, 2019. <https://www.logical-arts.jp/archives/124> (Accessed 20190116)
- [37] kaeken,
/scikit-learn/分類精度を評価する際に使われる混同行列(Confusion matrix)について. kaekenのData Science探究ブログ <http://kaeken.hatenablog.com/entry/2018/02/02/210000> (Accessed 20190308)
- [38] Setoka Karikome, デジタル信号処理3, 2002. <https://slidesplayer.net/slide/11519754/> (Accessed 20190116)