Task 3: Discrimination of reflected sound signals by applying Gabor transform and CNN classifier

Frankfurt University of Applied Sciences
Master Information Technology (M.Eng.)
Machine learning Project
Prof. Dr. Andreas Pech

HOANG LONG, NGUYEN 1067534

THANH DUY, NGUYEN 1067783

THANH TOAN, TRUONG 1185050

hoang.nguyen3@stu.fra-uas.de
duy.nguyenthanh2@stud.fra-uas.de
tthanh@stud.fra-uas.de

***Abstract*—Within the scope of this report, our team introduces an approach to discriminate between two distinct objects A or B, by analyzing the time-frequency dependency of their reflected acoustic wave signal readings. Firstly, by applying Gabor transformation to the given data, we produced the respective time-frequency representation(TFR) of each echoed signal. In the further steps, we considered the spatial relation appearing in the TFR as a distinctive feature that we could exploit to detect the signal source. To classify those TFRs, as we treated them as images, we applied the well-known machine learning techniques in this field: Convolutional Neural Network(CNN). As the result, the workflow that we have designed can perform with high reliability in categorizing two object reflected signals within the MATLAB environment.**

Keywords—Reflected wave, Gabor transformation, Time-frequency representation(TFR), Convolutional Neural Networks(CNN) , MATLAB

DISCLAIMER

We declare that this report of the project is a product of our personal work, unless otherwise have been referenced. We also declare that all opinions, results, conclusions, and recommendations are our own and may not represent the policies or opinions of Frankfurt University of Applied Sciences (FRA-UAS)

I. PROJECT INTRODUCTION

With in this project, a scenario is assumed: an object with a distinctive surface area located in front of an ultrasonic sensor. The sensor emits an incident wave towards the object's surface and received the echo wave back. The time signal resulting from this period is the convolution of the incoming wave with the surface properties of reflecting object. With the knowledge of the incident acoustic wave as well as the analysis of the echo signal, we can conclude on which surface produced the reflected wave. This is mainly the task that we focused to solve in this report.

This report consists of six part. In the first one is the project introduction in brief as well as the overall paper structure. In the second part, we would like to point out few related points in the literature review to prepare for the project. Additionally, in the third part, few works done by the students that are one of any foundations are introduced. In the consequence, we will carefully present and explain our approach to solve the task in the part fourth of this paper. The results from multiple testing and experiments will be included in the part fifth. Finally, at the end, we will give our conclusion for this project as well as few of recommendations to keep improving the performance of the workflow in the future.

II. LITERATURE REVIEW

A. Gabor Transmormation:

Gabor transform was used to generate the dataset from the reflected acoustic sound signal readings. It is a special application of short time window Fourier Transform [1].

Before applying the Fourier Transform, Gabor Transform applies a Gaussian window at the specified time. This makes the time point near the examined time more relevant or has more weight. The output of every Gabor Transform is a frequency map of the current examined time t [1].

$$G_x(\tau,\omega) = \int_{-\infty}^{\infty} x(t) e^{-\pi(t-\tau)^2} e^{-j\omega t}\, dt$$

For a whole reading of a time series t increases until it reaches last t minus the window length. This Process creates the spectrogram images, which will be used as training, testing and validating data throughout this work [1].

B. Convolutional Neural Network:

In machine learning, neural network is a technique to mimic the operation of human brain. There are layers in a neural network and the structure of the layers can be defined as input, hidden, and output layers. Nodes or neurons make up. Nodes in a layer are connected to the adjacent layers and a value is assigned to each connection called weight. Therefore, a node can be thought as a function where the input values are multiplied with the respective weights, summed up and then put into activation functions. This is the general definition of a neural network [2].

Taking a step further from the general definition, CNN introduces convolution layers, hence the name, for feature extraction. The convolutional layers extract features and create feature maps. The complexity of the feature also increases as the layers are being stacked. Between convolution layers, sub-sampling or pooling layers are often

inserted to reduce the size and removing insignificant details of the feature maps while keeping the main feature. This helps reducing memory and computational power [2].

III. Related works

Hoang Long, Nguyen and Thanh Duy, Nguyen has done a previous work in the same topic "Audio Classification: CNN Network vs. LSTM Network – A Comparison"[2] [3] belonging to the course "Computational Intelligence" from Prof. Dr. Andreas Pech at FRA-UAS in summer semester 2020. These works are one of many foundations of this project.

IV. Our approach in details

A. *Overall workflow of our project*

To tackle the problem of detecting reflected object signals, we have designed our workflow as depicted in the Figure 1. Generally, we have coded three programs in MATLAB environment. The first one is responsible to transfer the raw data into the TFRs by applying Gabor transformation. The second one is meant for training a simple CNN architecture as a binary classifier, which will be discussed later in this report, to detect the key features in each TFR to classify them. The final code is used to give a prediction: either the input TFR belongs to object A or B by its previous gathered knowledge (pretrained network weights), which is the result of the second code. The fully description of every steps will be discussed carefully in details within this part of our report.

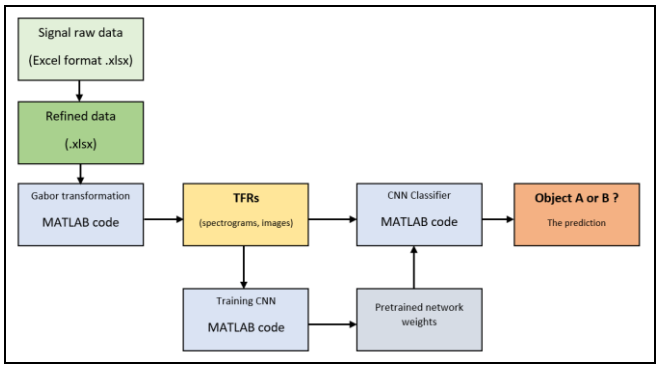


Figure 1: The workflow of our project to discriminate reflected sound signal

B. *Data importation from Excel to MATLAB*

Firstly, as the data is given by our professor as several Excel files (.xlsx) that contains numerous of reflected data readings. In details, there are nine files for object A including 315 readings and four files for object B including 200 readings. In the original format, each reading came with 3406 columns. It is noticeable that the first six values are irrelevant of the characteristic of the echo acoustic wave, thus we must refine the data by simply removing them. The plot of data can be done by Excel tool to get an initial overview structure of the signal, which as shown in the Figure 2. In the horizontal direction, it indicates the time relation of the data (3400 sampled data points) and in the vertical direction, it shows the signal magnitude in each specify moment of time (value ranges from -0.1 to 0.1).

In the next steps, we then gathered those 315 readings belonging to object A in multiple xlsx files and combined them into one completed xlsx file, named as 'combinedataforA.xlsx'. The similar step is also done for 200 data of object B to get the 'combinedataforB.xlsx'. The reason of those actions is to simplify the later step of importing those data into MATLAB environment.

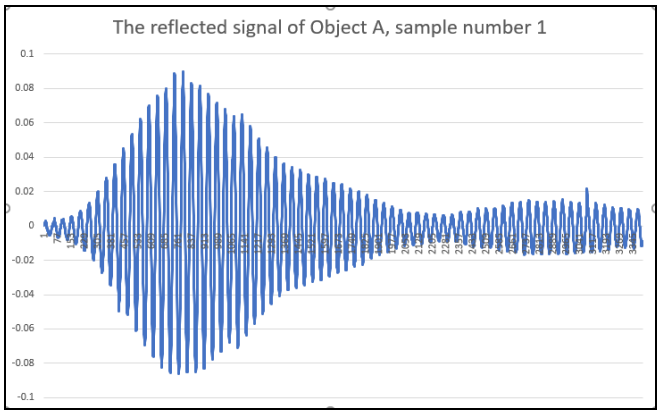
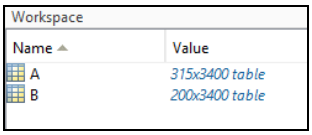


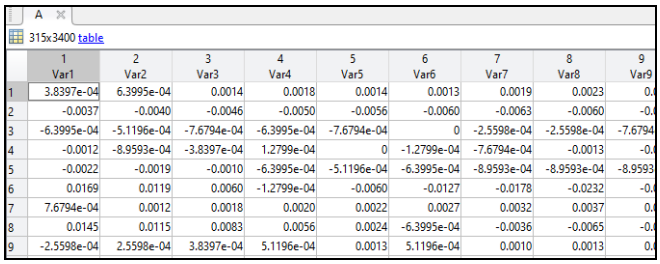
Figure 2: The sample (number 1) of a recorded echo wave back from the object A, initially generated by Excel

After getting two combined data sheets ready for importing, we prepared the MATLAB code to transfer those 315x3400 (object A) and 200x3400 (object B) vectors into readable types. By the MATLAB documentation, it is simple to apply the "readtable" function [4] to relocate the data from the xlsx into table type. However, regards to the implementation of readtable function, it might take up few minutes to complete, depends on the current running hardware, thus it is recommended by us that this step might only be initialized only at the first time or if some additional data might have been injected into the original one, as shown in Figure 3(top). For only the purpose of further testing with the old set only, workspace save, and load function is highly recommended to shorten the processing time. The proof of success import is demonstrated in the Figure 3(middle) and (bottom).

```
% Frankfurt University of Applied Sciences - FRA-UAS
% Faculty of Computer Sciences and Engineering
% Machine Learning Project Winter Semester 2020/21 by Prof.Dr.Andreas Pech
% Group Task 3: Sensor reading classification by applying Gabor transformation and
% machine learning binary classifiers
% NGUYEN, HOANG LONG 1067534 hoang.nguyen3@stu.fra-uas.de
% TRUONG, THANH TOAN 1185050 tthanh@stud.fra-uas.de
% NGUYEN, THANH DUY 1067783 duy.nguyenthanh2@stud.fra-uas.de

%% Import data from XLS Excel file - should only run once to save time !
clear; close all; clc
A = readtable('combinedataforA.xlsx');
B = readtable('combinedataforB.xlsx');
```





	1 Var1	2 Var2	3 Var3	4 Var4	5 Var5	6 Var6	7 Var7	8 Var8	9 Var9
1	3.8397e-04	6.3995e-04	0.0014	0.0018	0.0014	0.0013	0.0019	0.0023	0.0
2	-0.0037	-0.0040	-0.0046	-0.0050	-0.0056	-0.0060	-0.0063	-0.0060	-0.0
3	-6.3995e-04	-5.1196e-04	-7.6794e-04	-6.3995e-04	-7.6794e-04	0	-2.5598e-04	-2.5598e-04	-7.6794
4	-0.0012	-8.9593e-04	-3.8397e-04	1.2799e-04	0	-1.2799e-04	-7.6794e-04	-0.0013	-0.0
5	-0.0022	-0.0019	-0.0010	-6.3995e-04	-5.1196e-04	-6.3995e-04	-8.9593e-04	-8.9593e-04	-8.9593
6	0.0169	0.0119	0.0060	-1.2799e-04	-0.0060	-0.0127	-0.0178	-0.0232	-0.0
7	7.6794e-04	0.0012	0.0018	0.0020	0.0022	0.0027	0.0032	0.0037	0.0
8	0.0145	0.0115	0.0083	0.0056	0.0024	-6.3995e-04	-0.0036	-0.0065	-0.0
9	-2.5598e-04	2.5598e-04	3.8397e-04	5.1196e-04	0.0013	5.1196e-04	0.0010	0.0013	0.0

Figure 3: (top) The data import code section; (middle, bottom) Recorded data is successfully imported into MATLAB Workspace

To progress the data, it is feasible to pull out each row of the whole data A or B, convert them from table into array type for convenience in later steps, by MATLAB "table2array" function [4]. For example, as depicted in Figure 4(top), the code block is showing the steps of extracting the first row from object A combined data bank. Regards to the inhomogeneous values distributed in the magnitude of each data points, we have normalized the 1x3400 vector by dividing every element by its absolute maximum value. Finally, we decided to plot the reading again to compare with the initial plot done by Excel to ensure the data is intact after relocating from Excel to MATLAB environment as depicted in the Figure 4(bottom), as the shape of signal is preserved, only with the normalized magnitude.

```
t = (1:width(A)); % Time vector 1:3400 samples
c = 1;
S = table2array(A(c,1:width(A))); % Extract data from the table
S = S./max(max(S),abs(min(S))); % normalizing
plot(t,S); axis([1 width(A) -1 1]); xlabel('Samples'); ylabel('Amplitude')
title(" Object A Reading "+c);
grid on
```

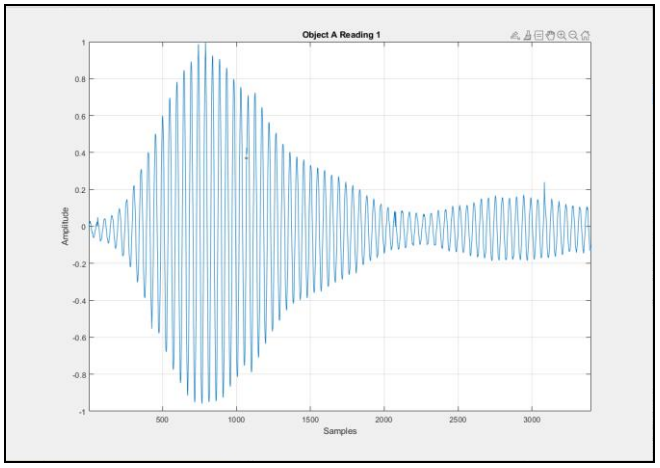


Figure 4: (top) Extracting and normalizing the sample number 1 code section (bottom) The plot of normalized reading of sample number 1 of reflected wave from the object A done by MATLAB

It is possible to store the plots as time series data streams and directly export them to a deep learning models such as 2D-CNN to learn to classify, however these plots contain only the magnitude-time dependency of the echo waves. This relation would not bring a significant difference between two class A and B; therefore, we must investigate a better way to continue to analysis the data. One of many potential methods can be mentioned as the Gabor transformation, by creating a TFR of the signal. In comparison with the magnitude-time plots, TFRs can display a deeper meaning in both frequency and time domain, thus would help the CNN classifier improve its performance considerably. The TFRs are sometimes denoted as the spectrograms in the later part of this report, but the meaning persists.

C. Gabor Transform Implementation in MATLAB

Mr. Nathan Kutz, who have done a magnificent lecture in time-frequency analysis and Gabor transform from his online course: "inferring structure of complex system" by the department of Applied Mathematics at the University of Washington[5]. His guidance is our foundation for the further steps in our project. Following by his lecture, we have successfully transformed both reflected signals from A and B to their according TFRs. Those TFRs, or namely spectrograms can indicate that in a particular time span which are the dominant frequencies of the signal and how intensive power they have. This is the key for the CNN classifier to distinct between two classes. For example, as demonstrated in the Figure 5, there are two reflected data signal A number one and B number one and their respective spectrograms. It points out the fact these two spectrograms as their TFRs can be even differed easily with our human eyes. From a two-dimension representation (the magnitude and time dependency), Gabor transform brings a new dependency between time, frequency and spectrum power magnitude that moves it into three-dimension representation, which enriches the packed information per readings. Nevertheless, now data can be not only treated as 2-D data stream, but also can be an image, which enables CNN to prove its superior in classifying.

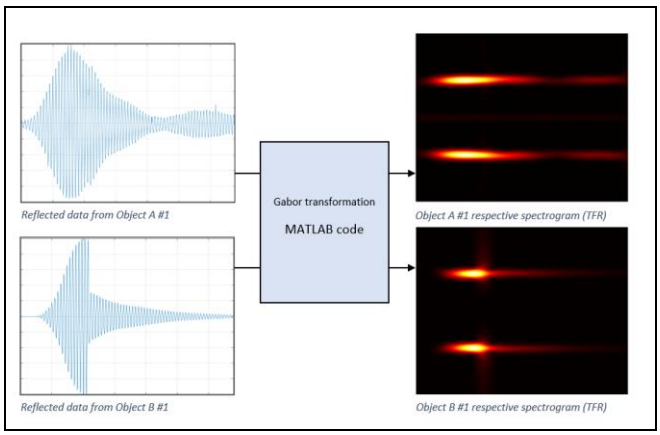


Figure 5: The generated TFRs (spectrograms, on the right side) of object A and B reflected signals accordingly (on the left side)

To get a better understanding about the properties of these spectrograms, it is practical to go through line by line in Mr. Nathan Kutz 's implementation of the Gabor transformation in MATLAB environment.

```
%% Applying Gabor transformation
% Reference Nathn Kitz - Time Frequencey Analysis & Gabor transforms
% Applied Mathematics - University of Washington

L = 10; %Signal duration = 10s assumption
n = 3400; % Number of data points
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
```

Figure 6: In Gabor transform code section, first three parameters are defined namely L, n, and k

At the beginning of his code, the signal duration is required to define other components in the later steps. Within this project, unfortunately the information of the length of signal is unknown, therefore we have decided to assume that every signal happens in ten seconds. We believe that this assumption would not bring any errors to the final predictions, but the deeper understanding about the accuracy of those dominant frequences in signal might be affected. As depicted in the Figure 6, the parameters namely L equals to 10 seconds, n as the number of total data points in each reading equals to 3400, and k is frequency componet for the later Fast Fourier Transform(FFT). Next, ks will be the product of "fftshift" function[4] of k. Apparently, the discussions about the FFT in details are out of the scope of this report, thus we only care about the application of the results of the FFT.

We can optionally verify the effectiveness of the FFT by applying the transform in the whole range of signal (in complete ten seconds). The normalized result is shown in the Figure 7, which depicts the new relation in the frequency range and the spectrum power respectively of the signal from object A number one. The dominant frequency of the signal is approximately 44.61 Hz, other weaker frequencies only contributed with the spectrum power less than 2% compared to the dominant. If we analyze the sampling frequency of 340 Hz (under assumption of within ten seconds, 3400 data points are recorded) to the maximum frequency of 44.61 Hz, the Nyquist–Shannon sampling theorem holds true at our assumption.

```
% Plot the overall FFT of the whole signal
St = fft(S);
plot(ks,abs(fftshift(St))/max(abs(fftshift(St))))
```

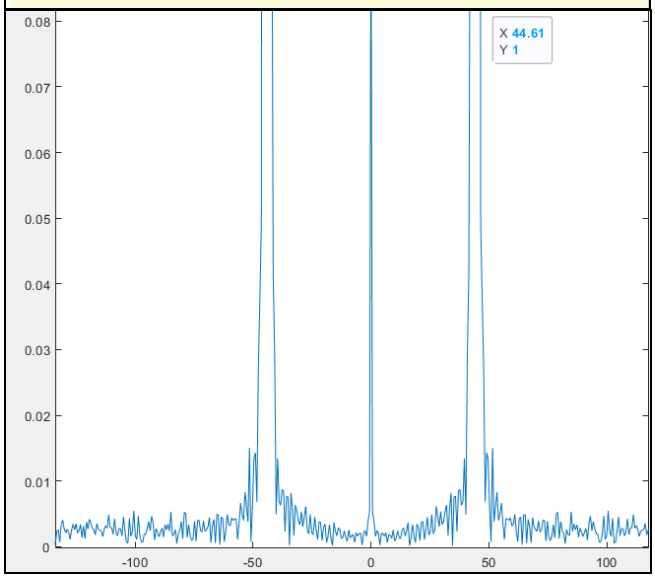


Figure 7: (top) The FFT code for the completed signal; (bottom) The complete FFT over ten seconds of signal plot from object A reading number one; in X-direction: contributed frequencies and in Y-direction: their power spectrum density

The FFT points out the dominant frequencies and their spectrum power density in the reflected signal, but if we only apply over the whole range of signal, we are missing the time relation. Thus, the Gabor transformation is introduced to be the solution for this problem. In Gabor transformation, there is a filter which slides over the range of signal what we wanted to analysis. In every moment of time, the Gabor filter will pull out a little piece of signal and try to investigate the most influential frequencies at that time window using FFT. This technique will be conducted repeatedly until the end of the signal, where the FFT results are stacked over each other. At the end of process, the stack of dominant frequencies and their spectrum power density overtime represents the time-frequency representation (TFT) or namely the spectrogram.

In Mr. Nathan Kutz 's implementation, we have added few customizations for easier demonstration in MATLAB code as shown in the following Figure 8 (top and bottom). Firstly, the "Sgt_spec" is generated as an empty array to store the spectrogram in later steps. The time vector "tslide" is then realized to defined how many data points are skipped between each FFT calculation. We have agreed to set "tslide" to be at 0:20:3400 (171 time steps-or 20 data points are skipped) regards the rule of thumbs during experiments respects to the progress time and the feature quality of the final TFR. If "tslide" is chosen to be denser, it is obvious that the code will take more time to complete, but the final spectrogram will be more accurate (or be sharper by the meaning of an image). On the other hand, if "tslide" is sparser, it is faster to compute but the spectrogram will lose many feature details as the tradeoff.

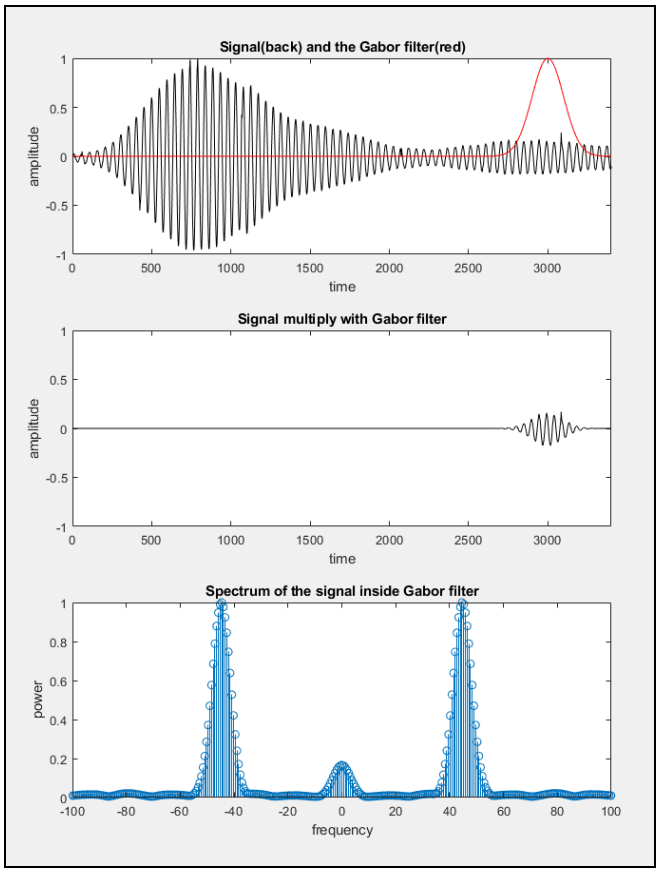
In the next step, we made a loop function to slide the Gabor filter over the completed signal according to Mr. Kutz instruction. The filter is defined in every step as the function "g". Under his guidance, the filter is an exponential function that forms a bell shape. The width of the bell can be manipulated by add the gain factor to the function. Once again, at this point, we have chosen the value of 0.00005 ($5*10^{-5}$) for the gain by the rule of thumb during testing. Like the "tslide", changing the width of the filter bell will affect the total running time and feature details of TFRs.

The filter "g" will be multiplied with reflected acoustic signal "S" to get the smaller "Sg", which is the little piece of data over the filter window. Then "Sg" will be applied the FFT to search for the contributed frequencies and their power density. "Sgt" will be the result as the product of the function fft(Sg). It is important that we do not normalize the FFT result within the loop, therefore the dynamic of signal can be preserved into the TFR. At this point, the Gabor transform is successfully deployed, and we only need to store the result by stacking every "Sgt" over each other to represent their time relation. This explained the code line "Sgt spec = [Sgt_spec; abs(fftshift(Sgt))];"

For the further code lines in the rest of Figure 8 (top), they are only our “quality of life” customizations for the convenient representation during the calculation Gabor transformation. We added three subplots: first one means for displaying the reflected wave “S” in black and the sliding window filter “g” (bell-shaped) in red color; second one shows the “Sg” a piece of data pulled out from the origin; and the last one depicts the relation between the dominant frequencies and their normalized spectrum power density in that piece of data “Sg” at the point of time “j” along the “tslide”. Therefore, we aimed to have a nice animation of the moving filter over the complete data, which will explain properly the calculation process of Gabor transformation.

In the bellow of the Figure 8 (bottom), there are few code lines for the display of the final TFR or spectrogram. At the beginning, we must transpose the matrix which stored the spectrogram “Sgt_spec” because it is easier to understand if the time relation is along the horizontal axis. The new vertical axis then will show the dependency in contributed frequency in the signal. The color appears in the plot are coded in “hot” by the “colormap” function by MATLAB: as more it dark is, the smaller value it had, as more as it bright is, the larger value it stored, and it faded from bright red to yellow, then to orange and finally black. It is common to plot the spectrogram in this mode by the researchers, but it is also possible to change the color code to different one, for example mode “default” or “gray”. The final TFR can be seen in the Figure 9, which is the product of the reflected wave from object A reading number one.

```
Sgt_spec = [];
tslide = 0:20:3400; % Moving Gabor filter every 20 samples !
figure('Name',"Gabor transform of reading "+c)
for j = 1:length(tslide)
g = exp(-5*1e-5*(t-tslide(j)).^2); %Gabor filter function
Sg = g.*S;
Sgt = fft(Sg);
Sgt_spec=[Sgt_spec; abs(fftshift(Sgt))];
sb1 = subplot(3,1,1); plot(t,S,'k',t,g,'r')
title('Signal(back) and the Gabor filter(red)')
sb1.XLabel.String = 'time'; sb1.YLabel.String = 'amplitude'; axis([0 3400 -1 1])
sb2 = subplot(3,1,2); plot(t,Sg,'k')
title('Signal multiply with Gabor filter')
sb2.XLabel.String = 'time'; sb2.YLabel.String = 'amplitude'; axis([0 3400 -1 1])
sb3 = subplot(3,1,3); stem(ks,abs(fftshift(Sgt))/max(abs(fftshift(Sgt)))); %normalizing
sb3.XLabel.String = 'frequency'; sb3.YLabel.String = 'power'; axis([-100 100 0 1])
title('Spectrum of the signal inside Gabor filter')
pause(0.01)
end
```




```
Sgt_spec=Sgt_spec';% transpose the Spectrogram for better representation
figure('Name',"Spectrogram of reading "+c+" of Object A")

pcolor(tslide,ks,Sgt_spec),shading interp
set(gca,'Ylim',[-171 171])
colormap hot
```

Figure 8: MATLAB code to realize Gabor transformation: (top) Sliding filter over the reflected signal and applying FFT in each window and stacking the results; (middle) The animation screenshot of the Gabor transform; (bottom) Plotting the final TFT or spectrogram code

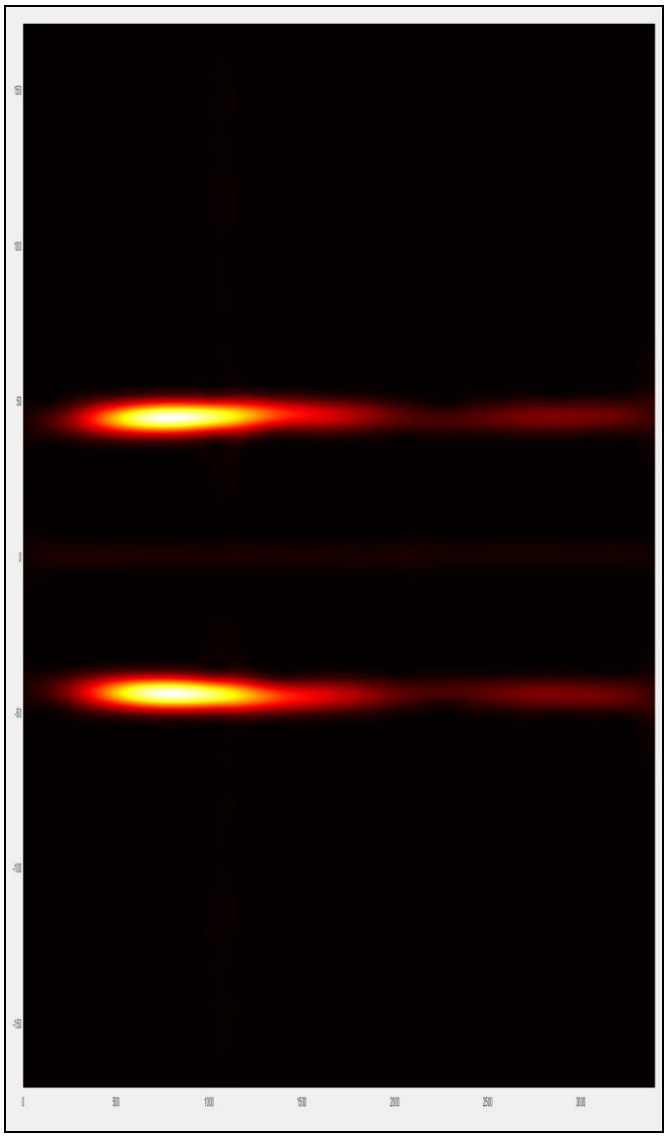


Figure 9: The complete spectrogram of the reflected wave from object A reading number one

D. Spectrogram exportation

It is clear to notice that the spectrogram is a mirrored image over the horizontal axis, thus it is unwise to directly use it to train the deep learning network such as CNN to classify. Furthermore, in both the low and high frequency regions in the spectrogram, there are nearly no contribution (nearly dark). These replicated and irrelevant pieces of spectrogram put a heavy burden during the later steps of training the network. Therefore, we decided to use only the region limited from 30 to 60 in the Y-direction, as it is shown in the Figure 10. The scaling of the new TFR might trick the human eyes, but the information stored is unchanged. It is feasible to conclude that the dynamic of the power density in both horizontal and vertical directions is the spatial key feature to distinct which source the data is coming from. From this point, we treated this new TFR as an image to train the CNN in later steps.

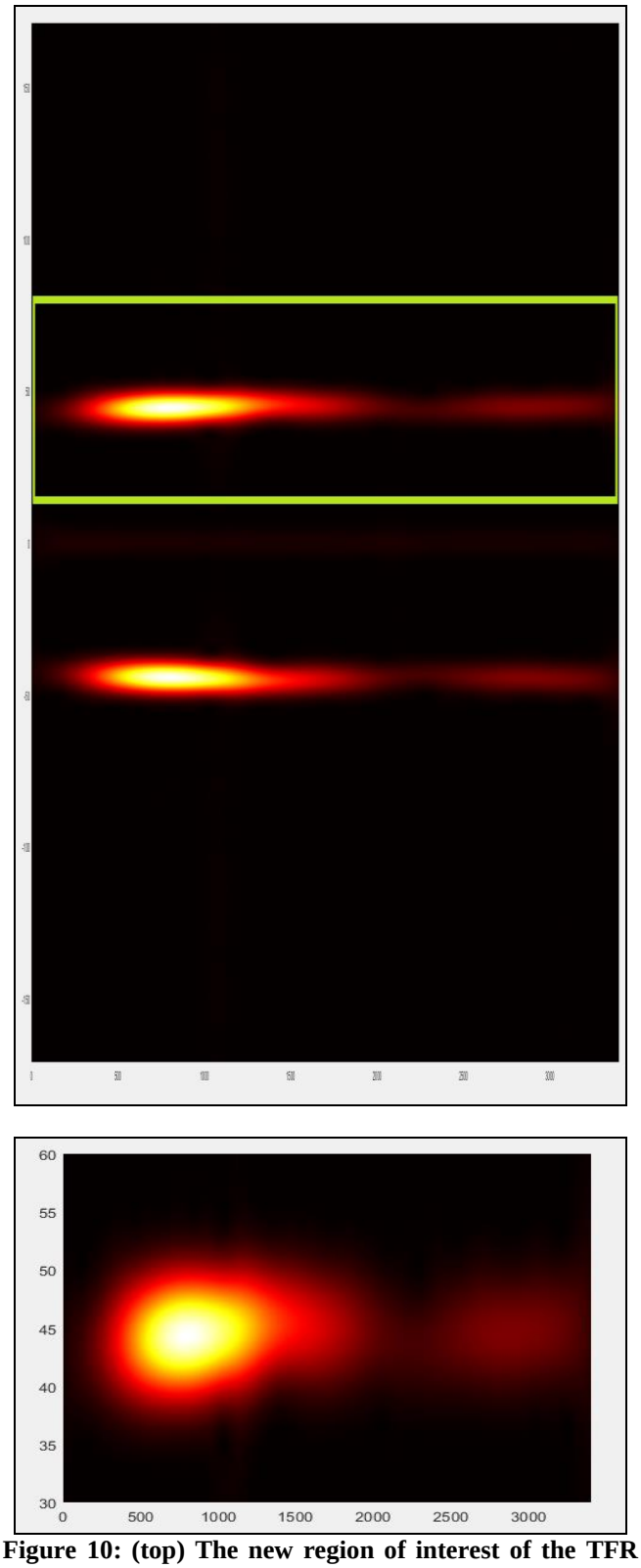


Figure 10: (top) The new region of interest of the TFR (inside the green box); (bottom) The new limited TFR (same as the one inside the green box)

At the final step, we tried to optimize by changing the three-channel spectrogram (RGB) to one-channel spectrogram (monotone) by switching the color code in colormap function to “gray” after few experiments and testing. Nevertheless, we set the coordinate values to be invisible since they are total unrelated to the later learning process. Finally, by the function “exportgraphics” in

MATLAB, we exported the spectrogram plots into image files (.jpg) with a specific name for each class A and B, as an example is depicted in the Figure 11.

```
pcolor(tslide,ks,Sgt_spec),shading interp
set(gca,'Ylim',[30 60])
colormap gray
set(gca, 'Visible', 'off');
exportgraphics(gca,"spectrogramA_"+c+".jpg",'Resolution',100)
```

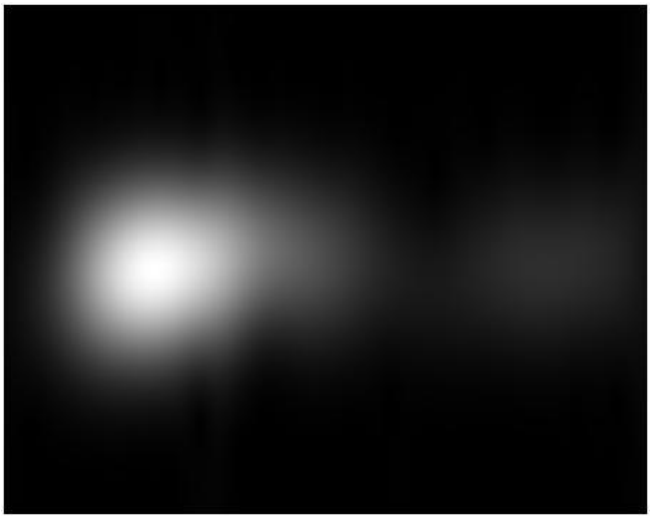


Figure 11: (top) The changes to export the new interested region of the TFR in Y-direction limits (b) The new color code "gray" for the TFR before giving it to CNN to train

In the summary, until this point, we have successfully applied the Gabor transformation to the given data from our professor (Excel data files .xlsx) to achieve the meaningful spectrograms (image files . jpg). There are few parameters that we can tweak to affect the processing time, the output spectrogram features as well as the image size that can be seen in Table 1:

Table 1: The parameters of Gabor transform of our implementation by MATLAB

Parameter name	Current value	Effect
tslide	0:20:3400	Progressing time, feature "sharpness"
Bell-shaped filter width	$5*10^{-5}$	Progressing time, feature "sharpness"
Limit of Y-direction in TFR	[40,60]	Spectrogram image size
Color code	"gray"	Spectrogram image size
Spectrogram resolution	100 dpi	Spectrogram image size

We then set up a loop function to go through every reading in the data bank to generate the new data set of spectrogram images of 315 readings and 200 readings of reflected wave from object A and B, respectively. Few of them are depicted as an example in the Figure 12:

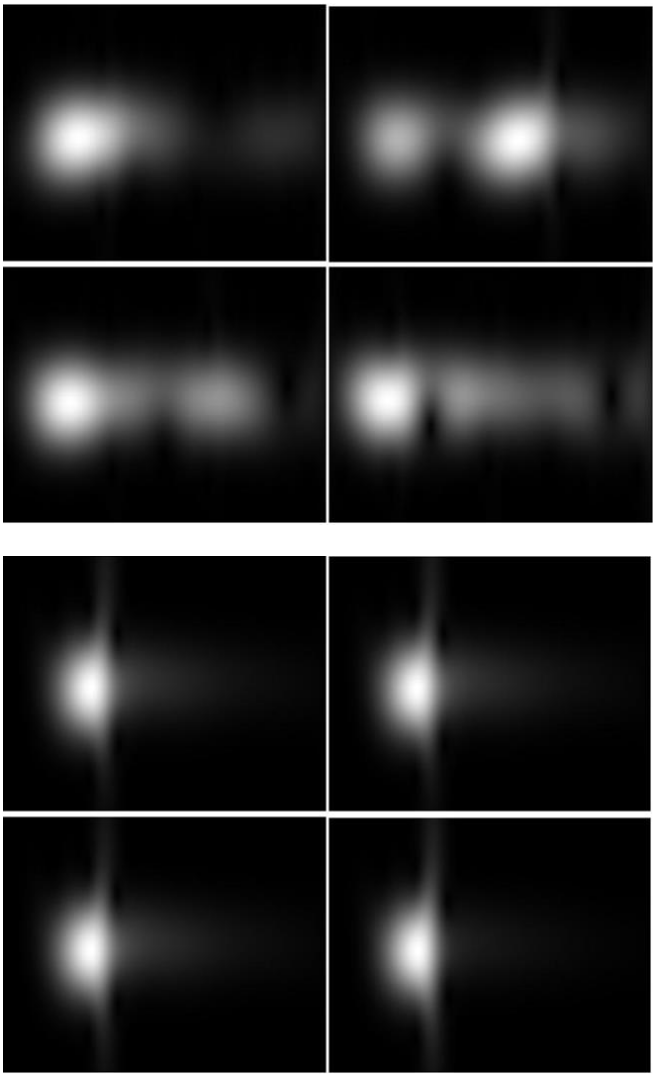


Figure 12: From left to right, top to bottom (top) spectrogram of reading number 1, 50, 100, 200 of object A; (bottom) spectrogram of reading number 1, 50, 100, 200 of object B

E. Create a simple CNN and train it as a binary classifier with the resulted spectrograms

This section provides step-by-step instructions on how to create and train a convolutional neural network for classification of this paper.[4]

- Load and explore image data.
- Define the network architecture.
- Specify training options.
- Train the network.
- Predict the labels of new data and calculate the classification accuracy.

1) Load and Explore Image Data

Load the spectrograms as an image datastore. *imageDatastore* automatically labels the images based on folder names and stores the data as an *ImageDatastore* object as demonstrated in Figure 13. An image datastore supports large image data, including data that does not fit in memory, and efficiently read batches of images during training of a convolutional neural network.

```
%% Specify the dataset directory
currentFolder = pwd;
imds = imageDatastore(fullfile(currentFolder, 'trainingData'), ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

Figure 13: Preparing steps for the dataset directory

Calculate the number of images in each category. "labelCount" is a table that contains the labels and the number of images having each label. The datastore contains 315 images for spectrogram type A and 200 images for spectrograms type B.

```
%% Getting number of labels
labelCount = countEachLabel(imds);
```

labelCount

2x2 table

1 Label	2 Count
spectrogramA	315
spectrogramB	200

Figure 14: (top) Labeling the data and (bottom) its location in the MATLAB workspace

Please note that the size of the images in the input layer of the network must be specified. It can be check by the following code as depicted in Figure 15:

```
%% Getting Image Size from first image in 'imds'
img = readimage(imds,1);
imageSize = size(img);
```

Figure 15: Code section to read the spectrograms as an image before feeding to the network

2) Specify Training and Validation Sets

The data are divided into training and validation data sets, in this case the training set contains 150 images for each category, and the validation set contains the remaining images. splitEachLabel splits the datastore into two new datastores, imdsTrain and imdsValidation as shown in Figure 16.

```
%% Specify the number of files used for training
filesToTrain = 150;
[imdsTrain,imdsValidation] = splitEachLabel(imds,filesToTrain,'randomize');
```

Figure 16: Splitting the data into training and validation sets

3) Define Network Architecture

We defined the convolutional neural network architecture as demonstrated in Figure 17:

```
%% Declare the CNN network
layers = [
    imageInputLayer(size(img))

    convolution2dLayer(64,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(32,'Stride',8)

    convolution2dLayer(32,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(16,'Stride',16)

    convolution2dLayer(16,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

Figure 17: Code section to define network architecture

- **Image Input Layer**: as the name suggests, the image size is specified here. Function size returns height, width, and the channel size for imageInputLayer. The digit data consists of grayscale images, so the channel size (color channel) is 1. By default, the network shuffles the data at the beginning of training.

- **Convolutional Layer:** in the convolutional layer, there are two main arguments. The first one is filterSize, which is the height and width of the filters the training function uses while scanning along the images. The second argument is numFilters, this indicates the number of neurons that connect to the same region of the input and thus determines the number of feature maps. Also, 'Padding' is used to add padding to the input feature map and 'same' padding are used to ensure that the spatial output size is the same as the input size.

- **Batch Normalization Layer:** batch normalization layers normalize the activations and gradients propagating through a network, making network training an easier optimization problem. Use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers, to speed up network training and reduce the sensitivity to network initialization.

- **ReLU Layer:** one of the commonly used nonlinear activation function.

- **Max Pooling Layer:** used to down-sampling to reduce the spatial size of the feature map and remove redundant spatial information. This reduces the amount of computational power required and make it possible to increase the number of filters in following layers. The first argument poolSize specifies the maximum values of rectangular regions of inputs that the max pooling layer returns. The 'Stride' name-value pair argument specifies the step size that the training function takes as it scans along the input.

- **Fully Connected Layer:** following the convolutional and down-sampling layers, this is the layer where the neurons connect to all the neurons in the preceding layer. Therefore, all the features learned by previous layers can be combined to identify larger patterns. The OutputSize parameter in the last fully connected layer should be equal to the number of classes in the target data which is 2 in this case. Because the last fully connected layer combines the features to classify the images.

- **Softmax Layer:** used to create classification probabilities by the classification layer. The softmax activation function normalizes the output of the fully connected layer and its output are positive numbers that sum to one.

- **Classification Layer:** the final layer. After the softmax activation function returns the softmax activation function, this layer assigns the input to one of the mutually exclusive classes and compute the loss.

4) Specify Training Options

After defining the network structure, training options should be specified. In this case, the network uses stochastic gradient descent with momentum (SGDM) with an initial learning rate of 0.01. The number of epochs is 5. An epoch is a full training cycle on the entire training data set. validation data and validation frequency are used to monitor the network accuracy during training. Shuffle the data every epoch. The network trains on the training data and calculates the accuracy on the validation data at regular intervals during training. The validation data is only used for accuracy calculation and not for updating the network weights. The Figure 18 shows the details in our implementation in MATLAB environment.

```
%% Specify Training Options for the model
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',5, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'MiniBatchSize',2, ...
    'ExecutionEnvironment','gpu');
```

Figure 18: Specify training options for the network

5) Train Network Using Training Data

Training the network comes after defining the layers, the training data, and the training options. This project uses supported GPU device to train the network, but it is also possible to train it using CPU. By default, trainNetwork uses a GPU if one is available, otherwise, it uses a CPU. By default, the execution environment will use the GPU, but it can also be specified by using the 'ExecutionEnvironment' name-value pair argument of trainingOptions.

The training progress plot shows the mini-batch loss and accuracy and the validation loss and accuracy. The loss is the cross-entropy loss. The accuracy is the percentage of images that the network classifies correctly.

```
%% Train The CNN Network
net = trainNetwork(imdsTrain,layers,options);
```

Figure 19: Code section for training the network

6) Classify Validation Images and Compute Accuracy

YPred is the result after predicting the labels, while YValidation is the validation data. With these two datasets, accuracy, which is the fraction of labels that was correctly predicted, can be calculated.

A confusion matrix can be plotted using “plotconfusion” function provided by MATLAB.

Some other statistics are also calculated in this project. These are True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN). Consequently, with the values: Positive Predictive Value (PPV), False Discovery Rate (FDR), Negative Predictive Value (NPV), False Omission Rate (FOR), True Positive Rate (TPR), True Negative Rate (TNR), F-score (F1), False Positive Rate (FPR) can be calculated.

```
%% Checking Network Performance
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation);
plotconfusion(YPred,YValidation)
TP = 0; TN = 0; FP = 0; FN = 0;
for i=1:size(YPred)
    if (YValidation(i)=="spectrogramA")&&(YPred(i)=="spectrogramA")
        TP = TP+1;
    end
    if (YValidation(i)=="spectrogramB")&&(YPred(i)=="spectrogramB")
        TN = TN+1;
    end
    if (YValidation(i)=="spectrogramB")&&(YPred(i)=="spectrogramA")
        FP = FP+1;
    end
    if (YValidation(i)=="spectrogramA")&&(YPred(i)=="spectrogramB")
        FN = FN+1;
    end
end
```

```
PPV = TP/(FP+TP);
FDR = FP/(FP+TP);
NPV = TN/(TN+FN);
FOR = FN/(TN+FN);
TPR = TP/(TP+FN);
TNR = TN/(TN+FP);
F1 = 2*TP/(2*TP+FP+FN);

FPR = FP/(FP+TN);
%ROC plot
FPRm = [0 FPR 1];
TPRm = [0 TPR 1];
figure
plot(FPRm, TPRm)
grid
axis([0 1 0 1])
```

Figure 20: Checking the performance of the CNN in classifying object A and B spectrograms (top) Calculating overall accuracy, plotting the confusion matrix, checking TP, TN, FP, FN (bottom) Calculating further parameters and the ROC curve

7) Save the network

We decided to save the state of the network (mainly the network weights) for future usage of classifying the incoming data in future as depicted in Figure 21.

```
%% Save the Network for future validation
save net;
```

Figure 21: Code section to save the trained network for future usage

F. Network Performance Analysis

In this section, the performance of the input different spectrograms would be tested and evaluate to choose the

most appropriate spectrograms. Recall that the spectrograms were created with RGB colors but later changed to gray scale. Also, the resolution of these spectrograms should be considered. Because these factors influence the speed, accuracy, and the integrity of the training process. Of course, to keep the consistency of the results, the layers of the network are not changed, only the spectrograms are changed.

Please note that the following results may only apply to the current hardware and may be different on other hardware. Given that the network runs on GPU, for your information, the GPU that was used to run this experiment is a RTX 2060 (laptop version) with 6GB of VRAM.

With the chosen spectrograms' resolution of 682x540 RGB scale (color code "default"; DPI = 150). The network achieves 100% accuracy with the total training and validation time of 22 minutes 53 seconds. However, running with this resolution repeatedly several times and it was noticed that sometimes there was error about GPU running out of memory. This prevented the network from further prediction and evaluation. This experiment is conducted, and its record is shown in the Figure A in the Appendix. An example of the used spectrograms is depicted in the Figure 22:

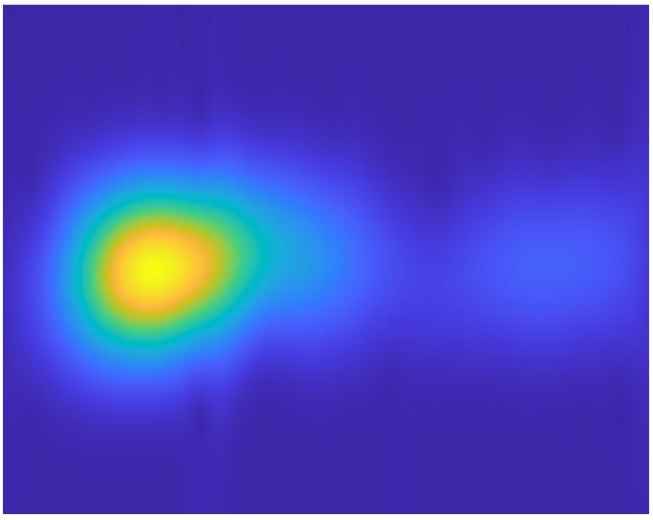


Figure 22: RGB Spectrogram #1 color code "default", image resolution 682x540 (DPI = 150)

As such, the input data were changed to gray scale but with the same resolution as mentioned above (color code "gray"; DPI = 150). The result was 99.53% accuracy which is well within the margin of error and with a similar total running time of 22 minutes, 53 seconds. The error, however, did not reappear. This experiment is conducted, and its record is shown in the Figure B in the Appendix. An example of the used spectrograms is depicted in the Figure 23:

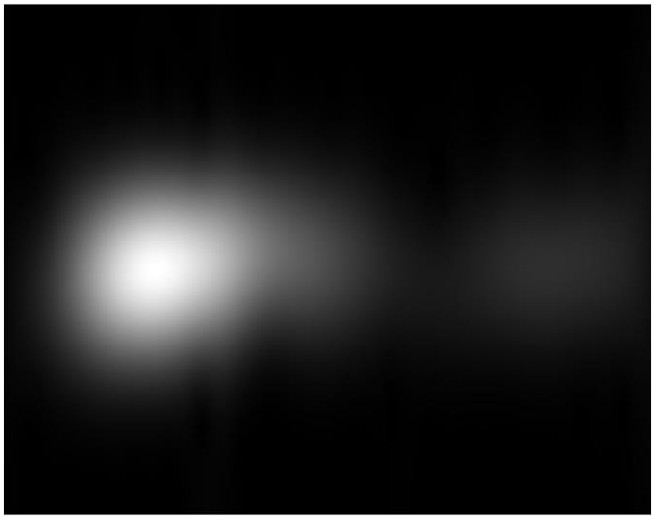


Figure 23: Gray-scale Spectrogram #1 color code "gray", image resolution 682x540 (DPI = 150)

Taking another step, the input data were reduced to 456x361 gray scale (color code "gray"; DPI = 100). This results in around 98.60% accuracy which is, again, within the margin of error but with half the total running of only 10 minutes and 43 seconds. Again, the GPU error did not reappear in this case. This experiment is conducted, and its record is shown in the Figure C in the Appendix. An example of the used spectrograms is depicted in the Figure 24:

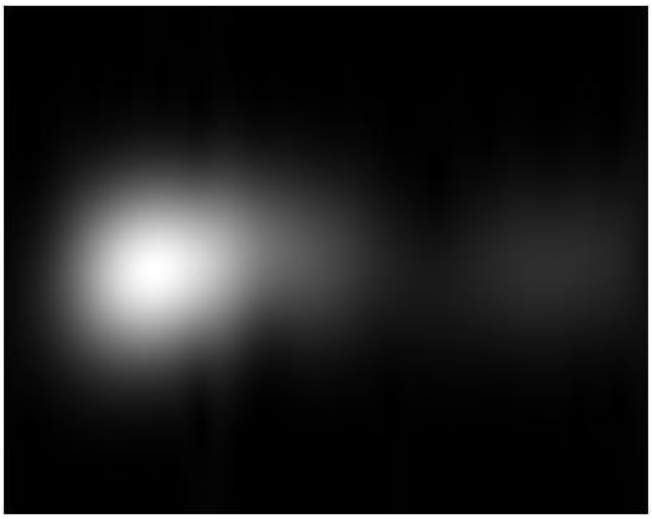


Figure 24: Gray-scale Spectrogram #1 color code "gray", image resolution 456x361 (DPI = 100)

Of course, with the results that were represented so far, and with a stronger hardware, 682x540 RGB scale may be preferable with its 100% accuracy so far (assuming that no GPU error are presented). However, it is important to notice that using 456x361 gray scale resolution as input data introduce no GPU error, have almost the same accuracy but with only half the running time. If the dataset is bigger, the running time can become a problem.

G. Generate GUI for user-friendly operation

We have designed a user-friendly GUI for our MATLAB implementation of Gabor transform and CNN

classifier. The app is call GUI.mlapp located in our repository, which is shown in the Figure 25:

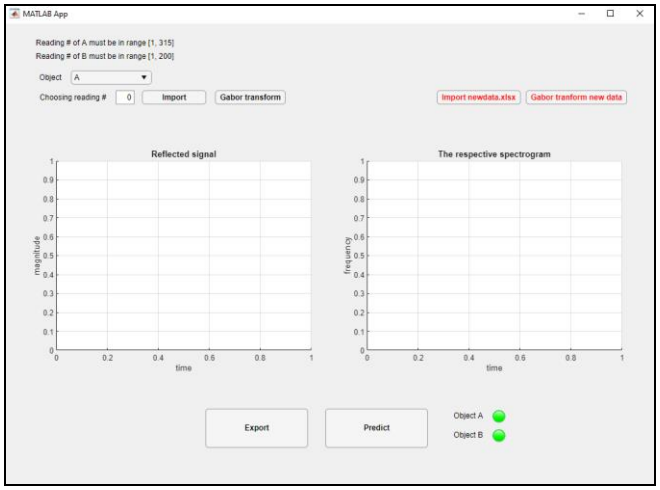


Figure 25: The GUI of our implementation

In the following parts, we would like to introduce a short user manual for this simple application:

Step 1: If the user wants to load the current data bank of both Object A and B, please use the upper left corner area of this app. To select which objects the user want, please use the drop-down option as shown in Figure 26. There is a note about the range of data of individual one, thus please ensure to enter the correct value of sensor reading number before pressing the "Import" button.

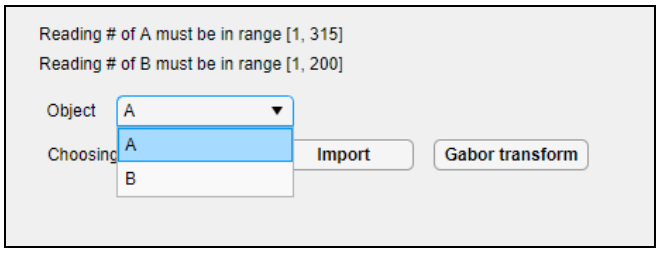


Figure 26: Drop-down option to choose between A and B

Step 2: After pressing the "Import" button, the corresponding plot of the sensor reading of reflected signal should be shown in the left-side plotting area, which is denoted as "Reflected signal". By pressing the "Gabor transform", the second plot of the respective spectrogram or namely TFR with the intended region of interest will be plotted in the right-side plotting area, which is denoted as "The respective spectrogram".

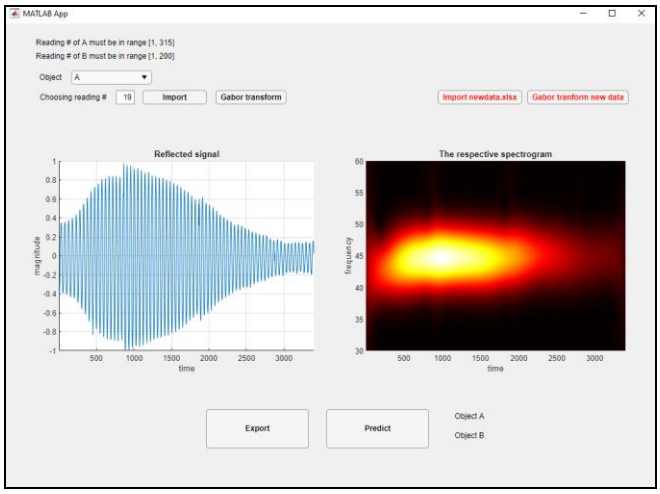


Figure 27: An example of using the GUI.mlapp

Step 3: After the steps of Gabor transformation, if the user wants to continue to classify this shown spectrogram to either belong to object A or B, please press the "Export" button in the lower area of this app. This step is successful done if the user can see the spectrogram in "hot" color code (as depicted with red orange yellow and black colors) has been changed into "gray" color code with a lower resolution (a little blurrier) as illustrated in the Figure 28. Additionally, the user can notice that an image file, named as "spectrogram.jpg" has been created in the same directory of this app.

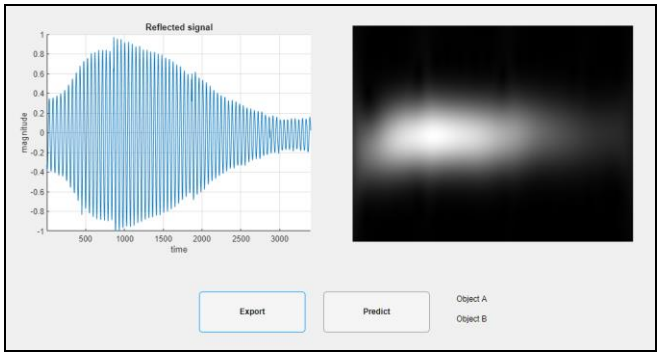


Figure 28: Spectrogram exportation is done by our app

Step 4: Finally, by pressing "Predict" button, the app will notice the user by the red light. If the spectrogram belongs to object A, its light will be turned on and same for object B as shown in Figure 29. It is unexplainable for some internal execution of MATLAB app designer that we recommend pressing the "Predict" button multiple times to get the correct prediction.

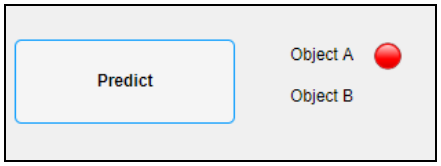


Figure 29: Our app gives the final prediction

If the user wants to use a new data for our implementation, please ensure these data is given in Excel format (.xlsx) and located in the same directory of this app

under the name “newdata.xlsx”. By design, we only allow to accept one line of reflected signal data per run. If the user wants to load multiple sensor readings at the same time, please consider the usage of our old GUI, but it is far more complicated. The additional user manual for the old GUI will be included at the end of the Appendices.

To start loading the new data and applying Gabor transformation, please use the red marked buttons in the upper right area of our app, namely “Import newdata.xlsx” and “Gabor transform new data” as shown in the Figure 25. The corresponding plots will be illustrated as same as the step one and two of using old data. To export and classify, users can follow the step three and four.

V. Testings and Results

Following the performance analysis, further statistics such as the confusion matrix, True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN), Positive Predictive Value (PPV), False Discovery Rate (FDR), Negative Predictive Value (NPV), False Omission Rate (FOR), True Positive Rate (TPR), True Negative Rate (TNR), F-score (F1), False Positive Rate (FPR) are provided for each resolution mentioned above in the following figures. We have randomly used 300 reading data for training and 215 read data for validation, respectively.

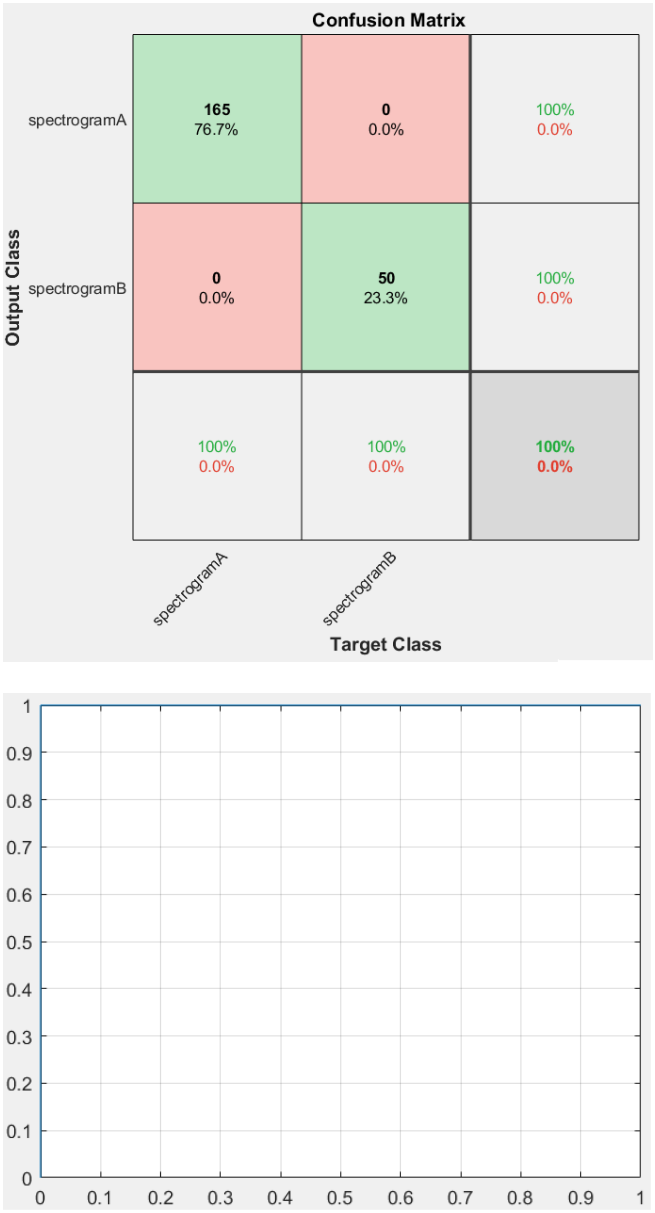
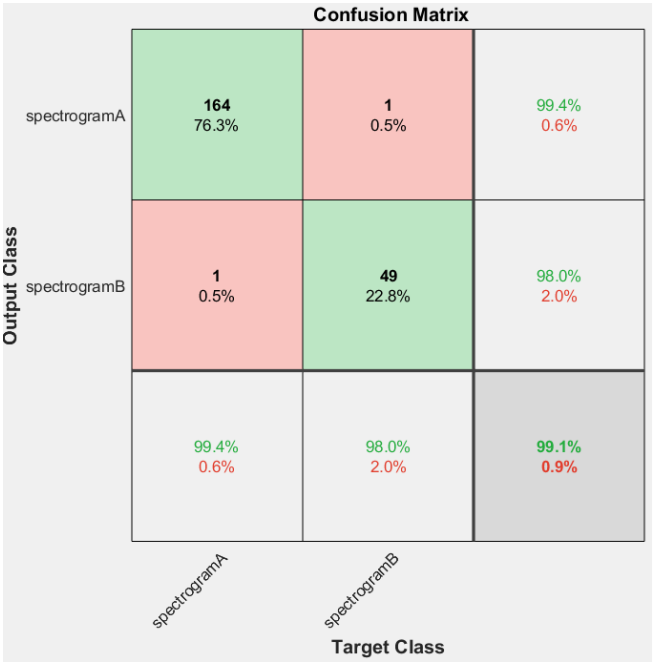


Figure 30: (top) First experiment results the confusion matrix and (bottom) the ROC curve for 682x540 (150 DPI) RGB spectrograms

Table 2: First experiment further statistics

Accuracy	100%
TP	165
TN	50
FP	0
FN	0
PPV	1
FDR	0
NPV	1
FOR	0
TPR	1
TNR	1
F1	1
FPR	0



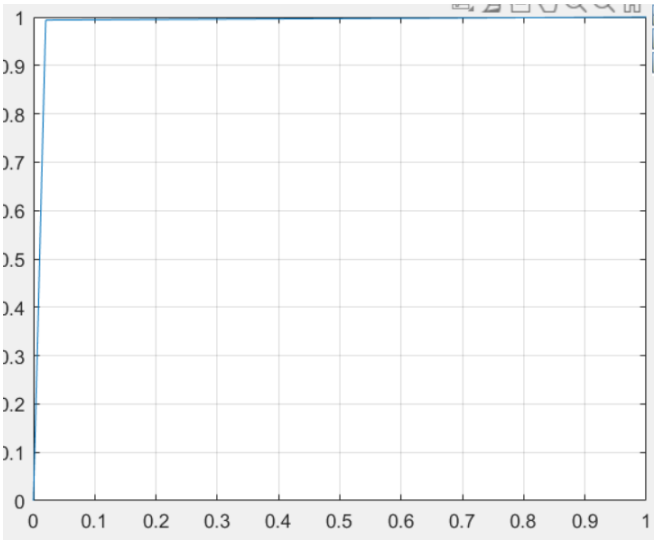


Figure 31: (top) Second experiment results the confusion matrix and (bottom) the ROC curve for 682x540 (150 DPI) grayscale spectrograms

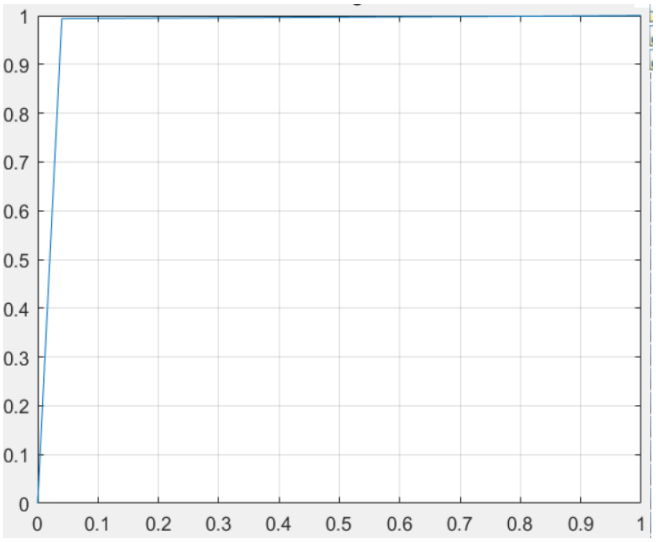


Figure 32: (top) Third experiment results the confusion matrix and (bottom) the ROC curve for 456x361 (100 DPI) grayscale spectrograms

Table 3: Second experiment further statistics

Accuracy	99.07%
TP	164
TN	49
FP	1
FN	1
PPV	0.9939
FDR	0.0061
NPV	0.9800
FOR	0.0200
TPR	0.9939
TNR	0.9800
F1	0.9939
FPR	0.0200

Table 4: Third experiment further statistics

Accuracy	98.60%
TP	164
TN	48
FP	2
FN	1
PPV	0.9880
FDR	0.0120
NPV	0.9796
FOR	0.0204
TPR	0.9939
TNR	0.9600
F1	0.9909
FPR	0.0400

Overall, the network achieved near perfect or even perfect performance. Higher resolution is preferable for accuracy but slows down the training and validation time, and depending on the running hardware, the computational requirement may be too much for it to handle.

VI. Conclusions and Future works

As the result of the project, it is claimable to conclude that our approach, in applying Gabor transform and CNN classifier to discriminate sensor readings from two objects A and B, is successfully done. The workflow received the given data in xlsx format, converted them into enriched-information TFRs or namely the spectrograms as images, feed them to CNN model to learn to classify and give highly meaningful predictions at the end. With the testing data shown in the previous part V, our pipeline developed a strongly reliable classifier, proving the effectiveness of CNN in image-like categorizing.

However, if considering the other possibilities to keep improving this project, there are a few points that we would like to call attention to as bellowing:

- **Clarifying the ambiguous signal length:** We have assumed that the signal length which consists 3400 data points lasts for ten seconds at the beginning of our project. As we have mentioned that assumption might not affect the final predictions, but if we want to have a deeper understanding in analyzing the TFRs, this information must be clarified from our professor.

- **Customizing the TFRs or namely spectrograms properties:** We have conducted several rules of thumb to choose the values for the properties of Gabor transform as shown in the previous Table 1. It is feasible to test with different values to evaluate which the best settings can be chosen to optimize the project in the future. Choosing the optimal region of interest in the TFRs is also curious topic.

- **Pre-processing the TFRs before feeding into CNN:** There are numerous techniques done by audio analysis researchers to pre-process the spectrograms namely: min-max scaling, z-score scaling, log-scaling that we can apply to test to keep improving the accuracy of CNN in later steps. An example of pre-processed TFR can be seen in the Figure 33:

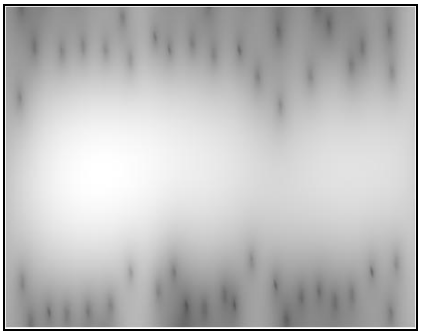


Figure 33: Log-scaled TFR of reading number 1 of Object A

- **Customizing the CNN architecture:** We can upgrade our simple CNN model into more sophisticated one if the workflow is required to discriminate more complicated objects.

- **Customizing the CNN learning options:** We can test the CNN with different learning options to find the most suitable setting to fit the current running hardware.

- **Expanding the dataset:** Our dataset is limited (515 readings in total), therefore we highly recommend expanding it in the future. By adding more readings in both objects as well as new objects, this project can be more practical. Unless, in our opinions, we can do the noise injection to 515 readings to expand the data, since the spectrogram augmentation for CNN can not be realized regards the spatial relation in the TFRs.

- **Optimizing the GUI:** There are plenty rooms for improvement in our new GUI. For example, in Code View supported by MATLAB, user can see that every call back instance of each button when pressed is an individual function (private) and they only communicate when read a value from drop down menu or editable text as input and plotting axes UI as output. A public function would be nice to have, therefore there is no need to add two different buttons for old and new data.

Acknowledgment

We would like to express deeply our thanks of gratitude to our professor Prof. Dr. Andreas Pech, who gave us the golden opportunity to explore his fascinating project on the topic "Discrimination of reflected sound signals by applying Gabor transform and CNN classifier" which have intensively broadened our knowledge and practical experience in this field.

Secondly, we would also like to cherish our parents and friends who encouraged us a lot during this project to finish it within the limited time frame.

References

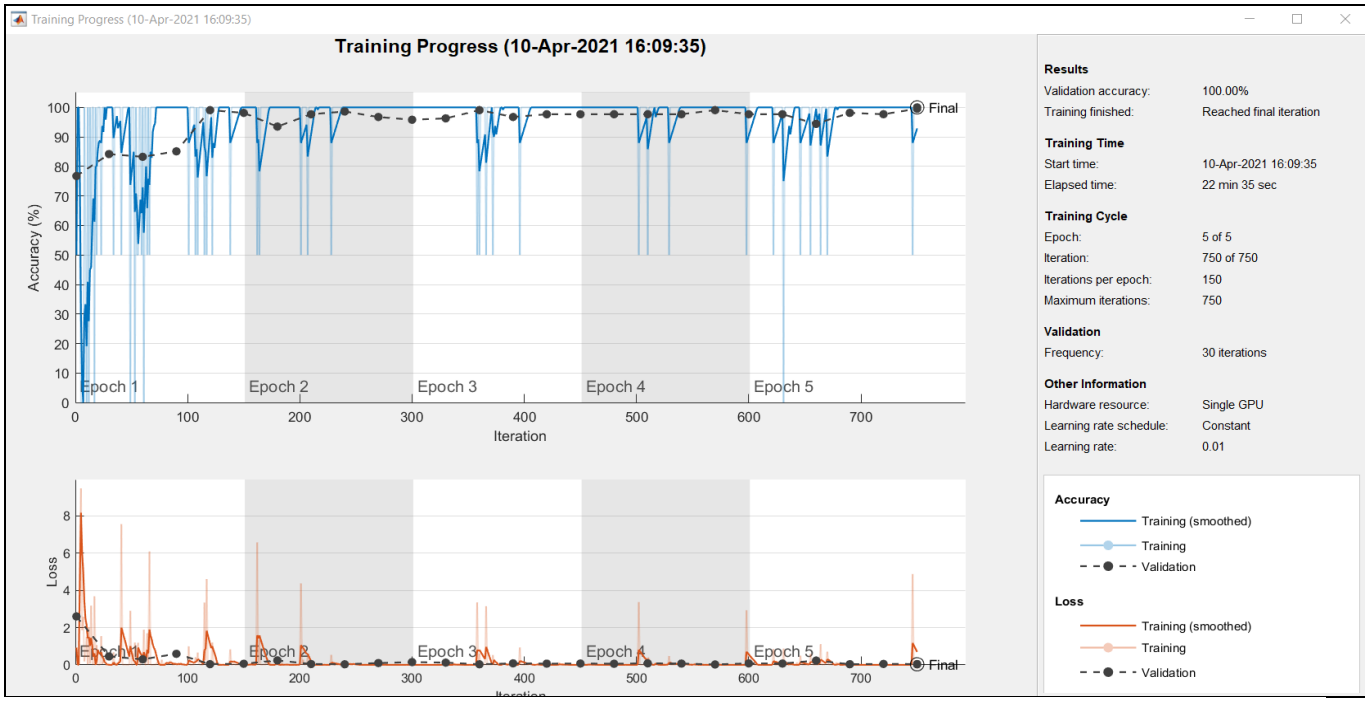
[1] D. Gabor, "Theory of communication. Part 1: The analysis of information", *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 429-441, 1946. Available: 10.1049/ji-3-2.1946.0074.

[2] Thanh Duy, Nguyen, " Computational Intelligence: Audio Classification: CNN Network vs. LSTM Network – A Comparison"

[3] Hoang Long, Nguyen, " Computational Intelligence: Audio Classification: CNN Network vs. LSTM Network – A Comparison"

[4] MATLAB Documentation [Online] available at https://www.mathworks.com/help/matlab/

[5] Nathan Kutz, "Time Frequency Analysis & Gabor Transforms" belongs the online course "Inferring Structure of Complex System" lecture [Online] available at https://www.youtube.com/watch?v=4WWvvMkFTw0




```
Error using nnet.internal.cnngpu.convolveBiasReluForward2D
Out of memory on device. To view more detail about available memory on the GPU, use 'gpuDevice()'. If the
problem persists, reset the GPU by calling 'gpuDevice(1)'.
```

Figure A: First experiment with the color code "default" spectrograms, high image resolution resulting in astonishing accuracy, significant training time and exceeding hardware training resource

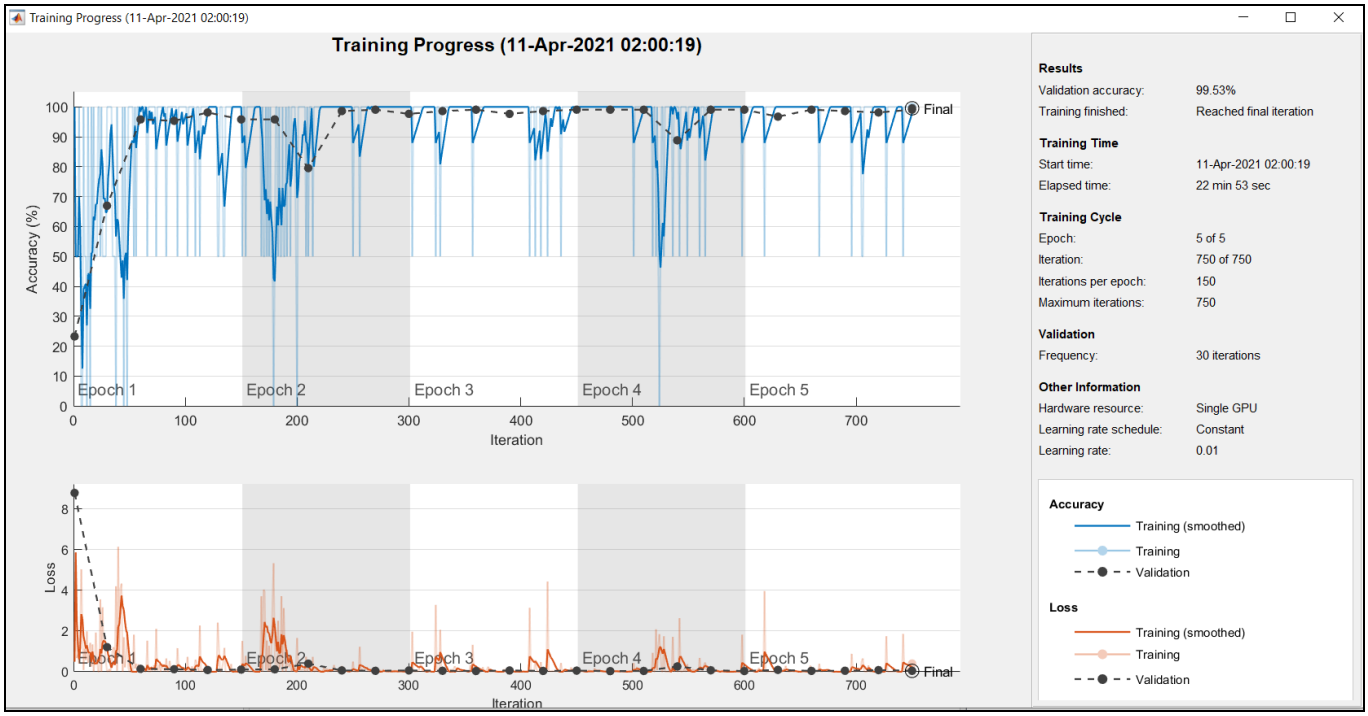


Figure B: : Second experiment with the color code "gray" spectrograms, high image resolution resulting in reduced accuracy, same significant training time but not exceeding hardware training resource

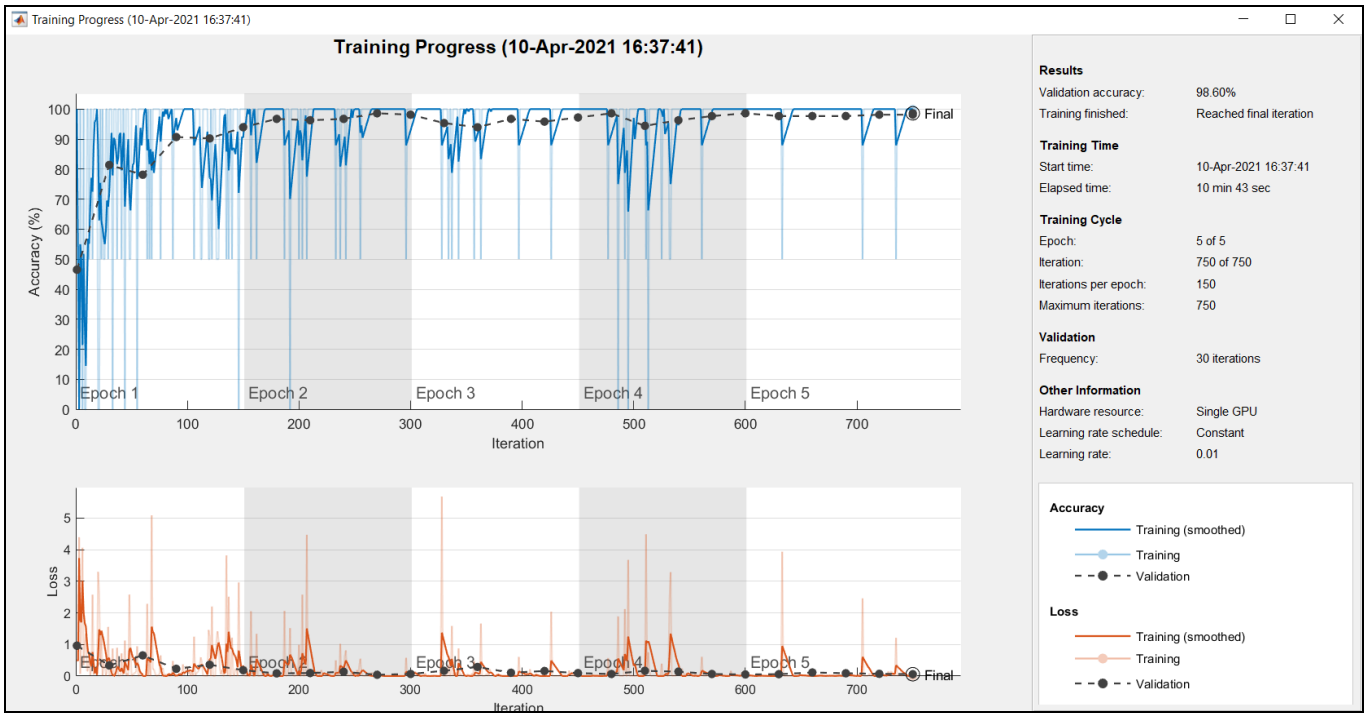


Figure C : Final experiment with the color code "gray" spectrograms, reduced image resolution resulting in reduced accuracy, sufficient training time

Warning: the trained network only works with its specified images dimension the testing process use the Resolution in the code when export images to manipulate this. For known experiment, Resolution 100: 456x361, 150: 682x540, and so on. Problem also occurs when exporting images as it also includes the toolbar of the graph. Because of this the testing process will crash. Running loading images again will fix the problem eventually. The dimension of the images can also be check outside before running the test to avoid crashing.

A. GUIApp publish version:

The GUIapp, created using app designer comes as a handy tool to test the trained network after training.

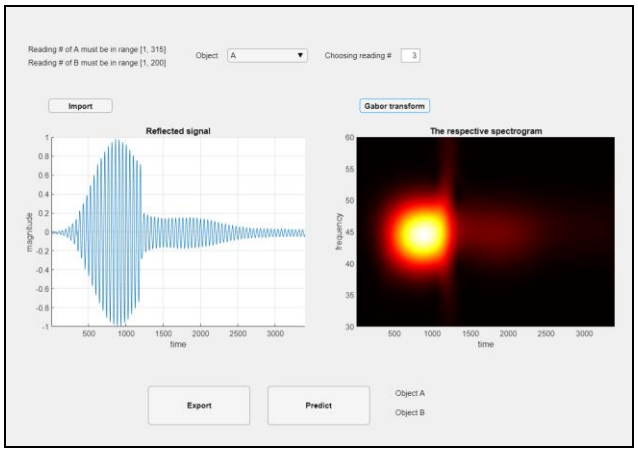


Figure 34: Testing GUI interface

After the network is trained, from the Interface, a reading from either of the dataset: *spectrogramB* and *spectrogramA* can be chosen by selecting the label (Object A or B) and the No# of reading.

Import button read the time series signal of the reading and plot it out on the left figure. To do a Gabor Transformation on the time signal, the button Gabor Transform should be clicked. The output will be shown on the right figure. Export Button export the Gabor output into a grayscale counterpart of it, this by experiment saves a lot of time conducting the test. Predict run the classify function to check whether the loaded reading is either an object A or object B. The result is indicated as a dot after the field *Object A* or *Object B*.

B. GUIApp old version:

An older version of the GUI existed with a more complete set of procedure used in the development of the project. This, however put a more complex approach compared to the publish version. A crude manual is laid here for further testing.

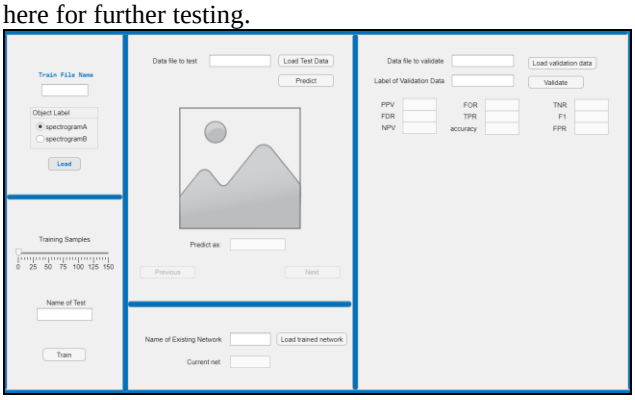


Figure 35: Layout of old GUI app

Figure 35 shows the whole interface of GUI old; it is divided into 5 sections: Load Training File, Load Test File (to predict), Load Validation File - validate, Train and save network, and load a saved network.

Figure 36 shows the Load Data to Train function. The .xlsx file must be inputted without the file prefix e.g., combinedataforA.xlsx should be inputted as combinedataforA in the Train File Name. Available data labels are “spectrogramA” and “spectrogramB”, as the requirement only includes the classification of 2 objects.

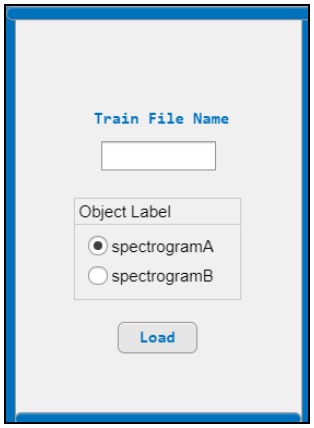


Figure 36: Load Train data from xlsx file

The Training Section trains the network, then save the net under the name <Name of Test>.mat in the current directory. The training sample number involved in training the CNN are specified using the slider. The train button starts the training process, which will open another window like ones above (Figure A, B, C).

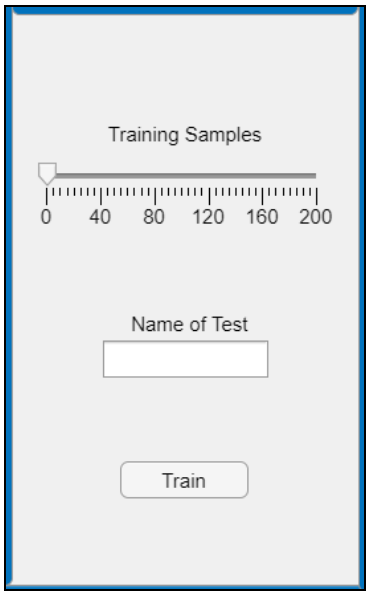


Figure 37: Training section

Figure 38 shows the Load network section. In this section, the current active network appears in the Current net text field. To load a trained network from the current directory, the name of the file .mat should be inputted.

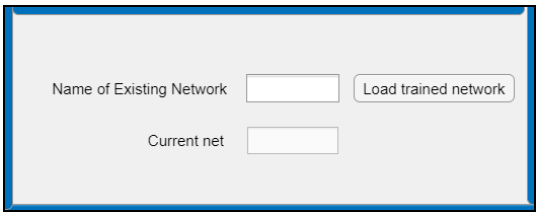


Figure 38: Load Network Section

Figure 37 shows the Predict section. The Data File to test field enquire the Users to input the test xlsx file. The data will then be read into images under testingData/, this data will then be loaded into the program. Eventually, the *Predict button* will used the current active CNN to predict the newly inputted data. The pictures can be browsed using the 2 buttons *Previous* and *Next*. The predict output is written in the predict as field.

Figure 39: Predict Section.

Figure 40 shows the Validation function of the GUI. The validation requires the data also in xlsx file format. The validation process also compels the data to have its label (*spectrogramA* or *spectrogramB*) to generate the confusion matrix (pop up window) and the Receiver operating characteristic curve (ROC). Validation needs a loaded CNN with appropriate resolution with respect to the images.

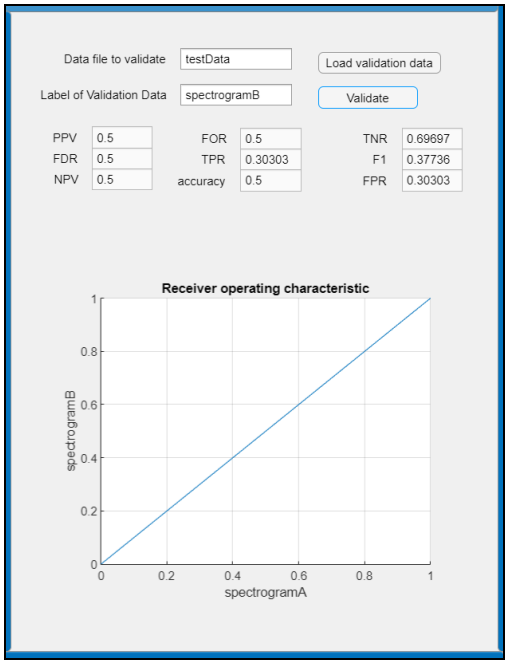


Figure 40: Validation Section