Classification of time signals by CNN using spectrogram

**Thanh Toan, Truong**: [tthanh@stud.fra-uas.de](mailto:tthanh@stud.fra-uas.de) - 1185050

**Hoang Hai, Pham**: [hhoang@stud.fra-uas.de](mailto:hhoang@stud.fra-uas.de) - 1184763

**Phan Bao Viet, Nguyen**: email

**Riyad-Ul-Islam**: [riyad-ul.islam@stud.fra-uas.de](mailto:riyad-ul.islam@stud.fra-uas.de) - 1324662

***Link to project***: https://github.com/Noath2302/Classification-of-time-signals-by-CNN-using-spectrogram

*Abstract —* In every aspect of modern technology, such as pattern recognition and artificial intelligence, the impact of time signal classification either theoretically or functionally is huge. Our novel approach within this paper, is to differentiate among three separate objects where the time-frequency dependency of their reflected acoustic wave signal readings has been analyzed. Initially, the respective time-frequency representation (TFR) of each echoed signal was computed using Gabor transformation to the specified time-series data. The spatial relation was considered for characteristic features extraction of the spectrogram in the subsequent steps. Classification of those TFRs which treated as images, was achieved via usage of a Convolutional Neural Network (CNN). In the end, the designed system can accomplish with great consistency in classifying three objects’ reflected signals within the MATLAB environment.

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

# Introduction

Convolutional neural network (CNN) is a deep learning algorithm used to process the data of image. It is commonly used in computer vision as a classification technique to distinguish different objects. On the other hand, spectrogram is a representation method used to present three-dimension measured signals in two-dimensional diagram.

Based on the dataset provided by Professor Pech in the module Computational Intelligence at Frankfurt University of Applied Sciences (FRA-UAS), the goal of this project is to classify the reflected signals of different objects using CNN and spectrogram.

# Literature Review

## Gabor Transformation

Initially, all dataset provided for the subject are sets of analog signals in time domain. However, the interest of this project lies on the change of frequency spectrum with respect to time of the reflected signal from the object, or the change of frequency spectrum in different projection from the sensor to the object. In the path to pursue this goal, Gabor Transform [1] was used to convert the signal in time domain to time-frequency representations (TFRs).

In a quick overview, Gabor transform first filters the signals with a Gaussian window. The remain part of the signal from the filtering process would then undergo Fourier Transform. The filtering window shifted through a fixed number of timestamps every cycle until it reaches the end of the input sample. After applying Gabor Transform to the sample (1 sample = 1 time series of analog signal), the output set of Fourier transform for each window with specific begin timestamps formed together a TFRs. these TFRs can be presented in the forms of spectrogram. The following formula is the applied filter as discussed:



*Figure 1: Gabor Transform formular*

From the formular, the Fourier Transform and the Gaussian filter can be observed in two different multiplicators, with the Gaussian filter is the exponential which contains **τ**. The shift **τ** represent the time step of the Gaussian filter in each iteration.

For a more rigorous definition, this method belongs to a family of short-time Fourier transforms and was named after Dennis Gabor upon his introduction of it. The goal of Gabor transforms is to give a clear view on what is happening on the frequency characteristic (strength of sinusoidal frequency and phase) of a change signal in time domain [2].

## Convolutional Neural Network

The concept of neural network or artificial neural network is commonly known as a combination of different layers connected to each other to make decisions based on different types of input. Biologically speaking, the neural network is a technique that mimics approximately how a brain function. Each layer contains various nodes acts as a system of neurons that can interconnect between layers. Besides, dependent on the importance of each specific neuron, or node, a factor called weight is introduced to bias for the purpose of the system. These layers are commonly known as the hidden layer.

....

On going...

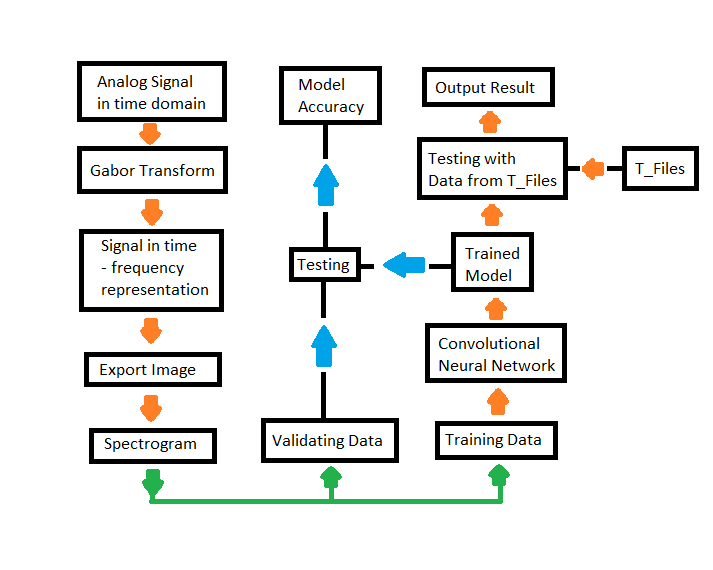
## Matlab

(Viet Nguyen)

# Methodology

## Overview

The problem of the project is how to classify time-series reflected signal of different objects. To solve this problem, a series of step was laid out in form of a project pipeline, which is presented above in this text.



In Summary, the code for the project was divided into 3 stages. Stage one involves processing of the analog time signal sample into spectrogram via Gabor Transform, see Literature Review for details of Gabor Transform. Stage two trains the Convolutional Neural Network (CNN) via adjusting the weights of the initial network to fit the training set. Stage two ends with validating the learning by feeding the validation set to the trained network. In stage three, the trained network was used to predict unlabeled data from the test file set T\_Files.

## About the dataset

The data was issued from Professor Pech in .xlsx files specifically for the Subject Computational Intelligence at Frankfurt University of Applied Sciences. There are two types of datasets in the provided files from the subject.

1. *Training Files*

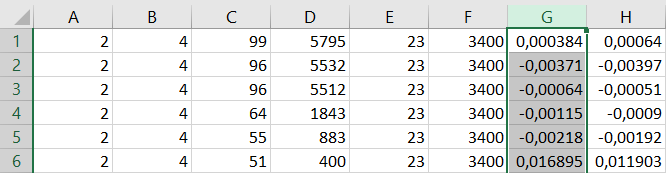
The training set includes 3 analog time readings samples of 3 different Objects: ‘Data Object1’, ‘Data Object 2’, ‘Data Object 3’. They are located under Matlab/dataset/. Each Row, from the 7th column in each files represents a time-series sample of the object. So, when the samples are read from the training data, the rows are read from the 7th column to the end of each row.

The samples in each training files:

|  |  |
| --- | --- |
| Data Object 1 | 315 Samples |
| Data Object 2 | 200 Samples |
| Data Object 3 | 400 Samples |

During the training, the number of samples used for training and validation were chosen via a split of data from the original training files. The current approach used the same number of training samples for each type of Object. Therefore, the number of validation samples left from ‘Data Object 2’ will be less than that of the others. This is the reason that the training data samples is currently set to 200, the maximum number of samples from ‘Data Object 2’.

Due to a current lack of training data, the scope of this Projects focused more on developing a pipeline for dealing with reflected time series signal of objects rather than reaching a good result of training.



1. *Test Files*

Test Files are structured the same way as the training Files, but they have no label, there are twelve test files in total, each has 50 data samples.

For the scope of this project, these test files were used as result of the model. They were fed into the trained model after the training session ends for the prediction of the associated label that belongs to the test file. Tests File are located under Matlab/dataset/ and share the same form as “T File <No>” with 12 >= No >= 1.

The documented output of the test file ‘s label is written under Matlab/Result/<netName> with netName is the name of the used trained network.

## Gabor Transform and Creation of Spectrogram

To do have a mathematically description of Gabor Transform, please refer to the literature review above. The creation of Spectrograms involves plotting and export the potted graph into .jpg, so during the creation of the pictures, a plot window will pop up and close for each data sample. The result of this process is the folders *‘trainingData’* and *‘testingData’, which holds folders of spectrograms - FIGURE*.



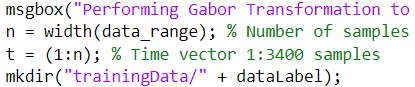
Transformation in 2 stages of the Experiment. One at the creation of the training data, reading the training set and one at the preprocessing of the test files. Consequently, in this section ‘*testingData’* and ‘*trainingData’* are used interchangeable. It serves the purpose of transforming time-series signal samples into time frequency representations (TFRs).



In Figure <>, the data samples, in time series form of the data was read as a table. In the second row, the data\_range extract the table for only the data from column 7th to end for each data samples



Because there are only one files for one label, dataLabel for the learning was chosen as the filename. In this work, the file names are ‘Data Object1’, ‘Data Object 2’ and ‘Data Object 3’. Consequently, the respective dataLabel are ‘Data Object1’, ‘Data Object 2’ and ‘Data Object 3’. So at this stage we have classification of multiple objects.



We then proceed to read the length of data\_range (extracted data from the original files, column 7th –> end) which is 3400 timestamps for every data samples. A variable ‘t’ was used here for the indexing of the timestamps for upcoming iteration. Then a folder was made in trainingData/<dataLabel> to save the output spectrogram. The same process happens when dealing with testing data, whereas a folder of name testingData/<dataLabel> is made.

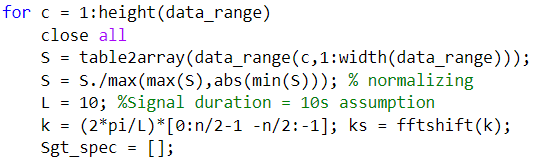


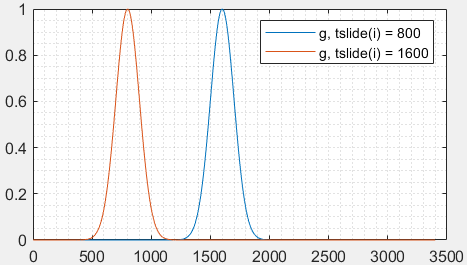
Figure **<>** demonstrates the iteration through all samples of one training files, which are now stored in variable **data\_range**. Because the spectrogram was generated via exporting image from a graph, multiple windows of plot will present while executing the loop. To cope with this, ‘*close all’* simplifies the process by closing the previous graph after each cycle. **S** is the normalized signal, in table forms, this variable has the value of a normalized sample from **data\_range** at index **c**. Here normalization means take the values of each timestamp in a sample and divide it by the max absolute value of all timestamps in that sample. **L** serves here as the period in seconds of a sample for calculation of the frequency spectrum, but as only spatial relation/ shape of each TFR/ Spectrogram is of concern of the Convolutional Neural Networks, it can be of any value, here it is set at 10s. The designed CNNs for this work focuses more on the graphical rather than the numerical features of the samples. **K** use **t** to generate the frequency scale of the graph. **ks** is a shifted version of **k**, which will be used to plot the frequency dimension of the spectrogram. **Sgt\_spect** is the strength of the frequencies or the spectrum density in each window, this will be presented as color in the spectrogram.

The Next session discuss about the iteration that simulated Gabor transform in this project. The instruction video of **<insert 45’ guy youtube name>** demonstrated the process beautifully. The code for Gabor Transform in this project is based on his Matlab demonstration.



**tslide** was used to take the indexes which belongs to the start of each window used for Gabor transform. To recall, Gabor transform makes use of a sliding window in the time domain, then Fourier transform the signal in each window respectively. In this experiment, the window moves 20 samples at a time.

In figure **<No>**The iteration of the Transform based on the index j, running from 1 to the end of tslide’s length.



The code in Figure **<>** shows the windowing function. **tslide** represents the shift in the time scale, he figure **<>** shows a better representation of the function **g** when plotted with different **tslide** values.

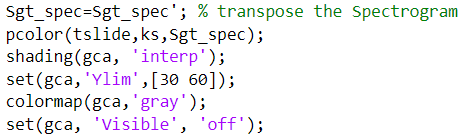


Then **tslide** is multiplied with **S** to get the corresponding time signal after filtered out by the filter function **g**.



The next step is calculating the Fourier Transform of **Sg** and add it to **Sgt\_spec**. After ending the Gabor Transform of a sample, three variables are now of interest to plot the final Spectrogram:

|  |  |
| --- | --- |
| **tslide** | Starting timestamp of each window, used for indexing in time axis |
| **ks** | Frequency scale of the signal, used for allocating the spectrum/ frequency axis |
| **Sgt\_spec** | Matrix of frequency strength for each window, used for color representation of pixels |



**pcolor** or Pseudocolor plot, is the function used for plotting a color graph with X axis (**tslide** - time), Y axis (**ks** – frequency spectrum) and color matrix of each pixel **C** (**Sgt\_spec** – strength of signal at each frequency). More about **pcolor** can be found on Mathworks official website **[reference]**.

The option that to change the spectrogram output extends this project with a greater source of inputs for the CNN. However, the current settings shown in figure **<>** optimizes for the fastest training time at the moments. To illustrate this process of choosing the settings for the spectrogram, the first sample of ‘Data Object 1’ is utilized.

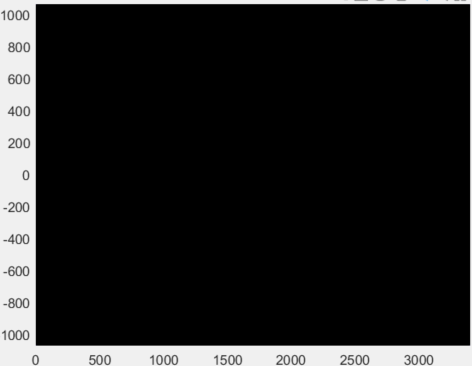
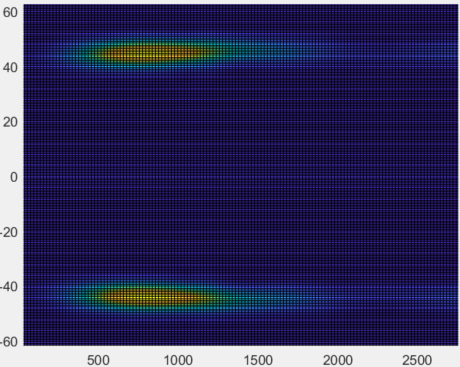
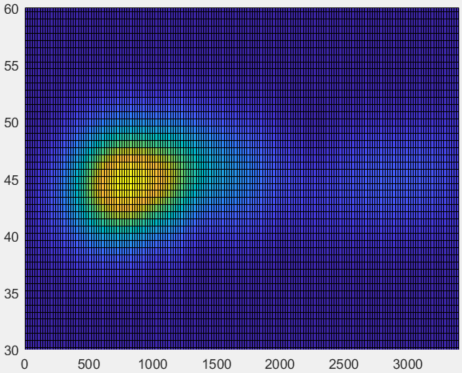


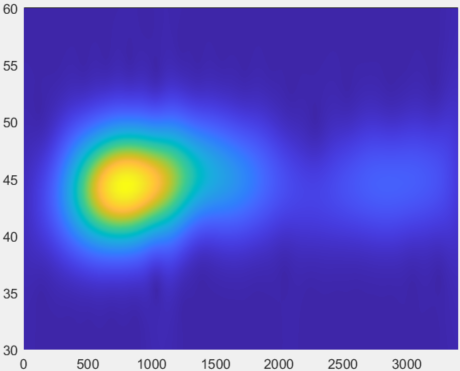
Figure **<>** shows the raw graph which is the output of only **pcolor**. It was considered at the start that this represented errors during the implementation. The Spectrogram only reveals itself when zoomed in**,** the zoomed version of the spectrogram is shown in the following figure **<>**.



Due to the symmetric nature of the Fourier transform/ Gabor transform, it is assumed in this work that half of the spectrogram has enough information of the object’s reflected signal readings. This explain the choice of the frequency range **ks** or **Y** here scaled to [30 60], the output can be seen in figure **<>**.

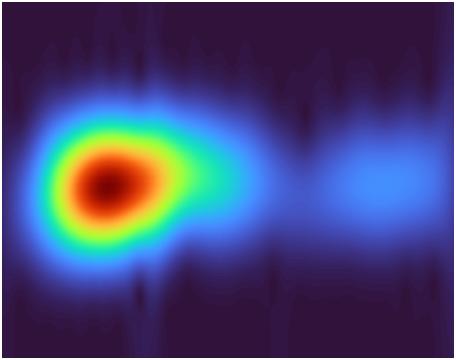
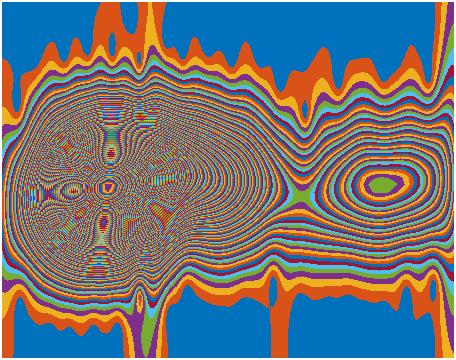
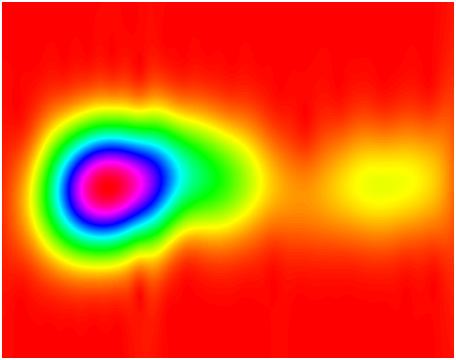
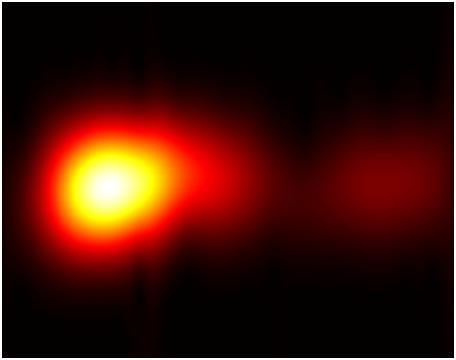
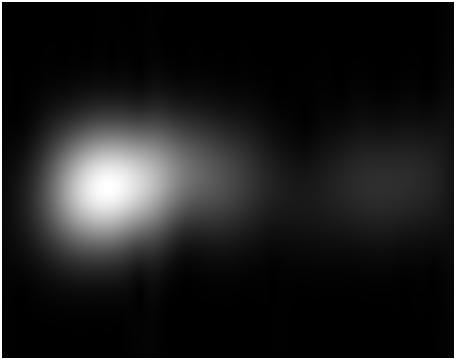
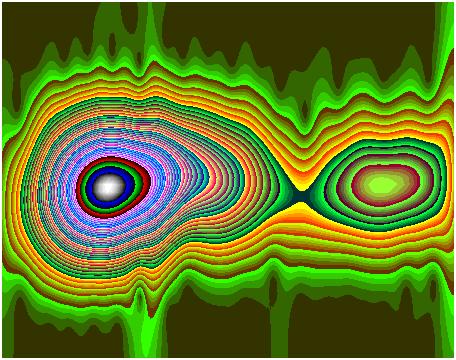


The Spectrogram at this point posed to have good quality of information extracted from the sample. Thus, the plot possesses unwanted table borders that are the same for all plotted spectrograms. To remove these lines, the option Interpolated shading of the plot was switched, figure **<>** shows the output spectrogram after using interpolated shading settings.



The spectrogram was thought to be enough for the training at this point. However, after exporting, the image appears to come with the axes, which is not of concern for the learning also due to its similarity in all exported spectrograms. To remove these axes, the “*Visible”* option of *gca* was set to *“off”*.

Training at this point is considered sufficient after all the data cleaning done. This work however took a step further in setting the spectrogram to grayscale by changing the colormap option to ‘gray’. The colormap setting in figure <> is the ‘default’ option, the figures <>, <>, <>, <> shows some available options that might be of interests for further research about which map draws the best performance.



After tweaking with the settings of the plot, a spectrogram is outputted. The last parameters to put in are the file name and the dimension.



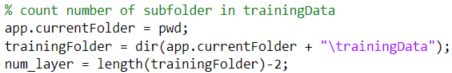
The dimension unit is dots per inch, the higher the number, the more details the exported spectrogram gets. Tables **<>** demonstrates the output spectrogram with different settings of dpi.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **dpi** | 10 | 50 | 100 | 200 |
| **size** | 42x25 | 230x183 | 456x361 | 908x719 |

*NOTICE:* The output of the experiment are trained networks which is specific to a type of exported spectrogram. A network trained for one type of spectrogram can only be used to predict the spectrograms that shares same properties with the training set. Unless the Input spectrogram is converted through further preprocessing, Matlab will raise an error when the input spectrogram is not compatible. It is recommended to name the trained network more specifically with the settings of the exported spectrograms.

## Training the Convolutional Neural Network

The modeling of the CNN consisted of 2 components: layers and options. The variable **app.layers** holds information about the neuronal structure of the networks whereas **app.options** describes the behaviors of the training process.



The initiation of both network components could be found in the **networkInit** function. The current generation of training Data described in section **C** above, the number of folders represents the number of output labels. The first code chunk in figure **<>** shows the process of getting the number of labels via the number of directories within the current directory.

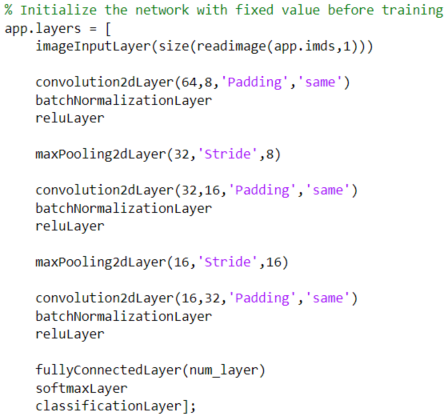
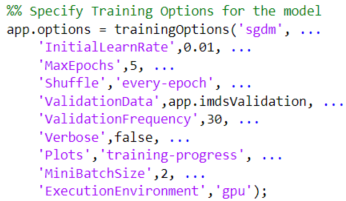


Figure **<>** represents a simple CNN model, which is used for the task of classifying different objects. The following section discuss about some definition of the layers and their functionalities in building the CNN model. These options are from the Matlab ‘s Deep Learning Toolbox. The documentation can be accessed on Mathworks website for more clarity **[ref]**.

* **ImageInputLayer**: This is the outer input layer of the CNN. So, the number of inputs here depends on the chosen number of pixels from the dataset’s image. The inner part of the code took the size of the first spectrogram from the original image dataset **imds**. The current used size of the spectrograms is 456x361 at 100dpi.
* **Convolution2dLayer**: This layer creates the backbone and the core logic of CNN. The first parameters in this layer config describe the convolutional filter size, the convolutional filter in this case is a square due to the one scalar passing. The filter specifies the size of local pixels which are included in one calculation. The next parameter describes the number of neurons that is in the layer, this will be the number of output features that will be the inputs to the next layer. The ‘*Padding’* option *‘same’* ensure that the output from this layer and the input have the same dimension.
* **bachtNormalizationLayer**: This layer normalizes a mini-batch of data independently across all observations in each channel. Training CNN faster via decreasing network initialization’s sensitivity was achieved by using batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers. After normalization, the layer scales the input with a learnable scale factor γ and shifts it by a learnable offset β.
* **reluLayer**: or Rectified Linear Unit (ReLU) is a common activation function that are used in machine learning. In short, it converts all values that are less than a threshold (usually 0) to become 0.
* **maxPooling2dLayer**: A 2-D max pooling layer performs down-sampling in its input. The layer first divides the input into rectangular pooling regions. Then it computes the maximum value in each divided region.
* **fullyConnectedLayer**: A fully connected layer multiplies the input by a weight matrix and then adds a bias vector. This layer takes the whole input form the last layer and have at output the same number of input labels. Its parameter is also the number of Labels/outputs.
* **softmaxLayer:** A softmax layer applies a softmax function to the input. The softmax function is known as a normalized exponential function. This play the normalizing roles for the classification.
* **classificationLayer**: *From Mathworks – “*A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes. The layer infers the number of classes from the output size of the previous layer. For example, to specify the number of classes K of the network, you can include a fully connected layer with output size K and a softmax layer before the classification layer”. Basically, it calculates what the output to belongs to in the pool of learned Label.

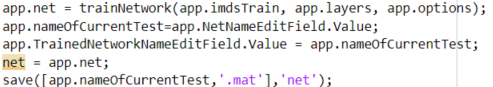
After the initialization of the network structure, the learning behaviors of the network **app.options** is specified, for an overview see figure **<>**.



There are a lot of option available in the configuration of **app.options** before training the network. ‘*sgdm*’ is an option that enables Stochastic Gradient Descent with momentum, which is a method of assigning cost function to the weights of the network. ‘*InitialLearnRate’* means the learning rate of the weights, it is a scalar which is multiplied into the change of weights to ensure that the learning can slowly reach its stable point. *‘MaxEpochs’* was set to 5 based on experimental proof. With a lot of observation on the learning of the dataset, most of the run reach maximum accuracy at epoch 5 or 6. So, the number was set to 5 to reduce the training time of the model. ‘*Shuffle’* option shuffles the order of the inputs, so that the training can reach a more general solution rather than a local minimum of errors. *‘ValidationData’* specifies the dataset used for Validation Process. *‘ValidationFrequency’* defines how much the network is validated in each epoch, the default was 50, so a smaller value resulted in a faster training session. To display the training progress, *‘Verbose’* was set to false and *‘Plot’* was set to *‘training-progress’*. A mini-batch is a subset of the training set that is used to evaluate the gradient of the loss function and update the weights - MathWorks. The parameter used by the *‘MiniBatchSize’* specifies its size.

*‘ExecutionEnvironment’* was set here to use *‘gpu’.* However, it is recommended to be set as *‘auto’* as some computers are not available with a GPU. If this happened, the training would use CPU as the main power source for training the networks.

After tweaking up the structure and the learning parameter of the network, the training start with **trainNetwork** – see figure **<>**.



The code after the learning saves the trained network to a mat file. This mat file will later be used to predict the testing data, which are provided from Prof Pech.

## Validation of trained network

Validation of trained networks happens right after the training of the CNN. The current validating process use the Validation dataset split from the first step from the Training Dataset. The data are fed into the trained networks to drawn out the labels of the data sample.

The input data at validation stage is images (spectrograms), which are already created together with the training data. They were split from the original training set. Validation stage in this project includes the computation of the following parameters:

|  |  |
| --- | --- |
| **PPV** | Positive Predictive Value |
| **FDR** | False Discovery Rate |
| **NPV** | Negative Predictive Value |
| **FOR** | False Omission Rate |
| **TPR** | True Positive Rate |
| **F1** | F-score |
| **Accuracy** | Accuracy |
| **FPR** | True Positive Rate |
| **TNR** | True Negative Rate |
| **ROC** | Receiver Operating Characteristic |

The following attributes, **accuracy**, **Receiver Operating Characteristic** (**ROC)** and the **Confusion Matrix** are calculated based on all three Labels. The others are calculated on the basis as ‘Data Object 1’ or ‘not Data Object 1’, this is not only for the simplification of the problem but also to fit with the requirements of the given project on classifying reflected time signal of Objects.

The Confusion matrix, the output of the training, and the training progress of 5 pre-trained networks can be found in the Experiment section.

## Testing of Data from T Files

Testing of trained networks can take place whenever there is a trained network in the same directory of the GUI, there are several trained networks submitted together with the project, each has their own validation result saved under /images/<netName>. This infrastructure separates the training process and the Testing process, which save time when conducting and documenting multiple experiments. For details on the Testing procedure, see the description of steps on the GUI section.

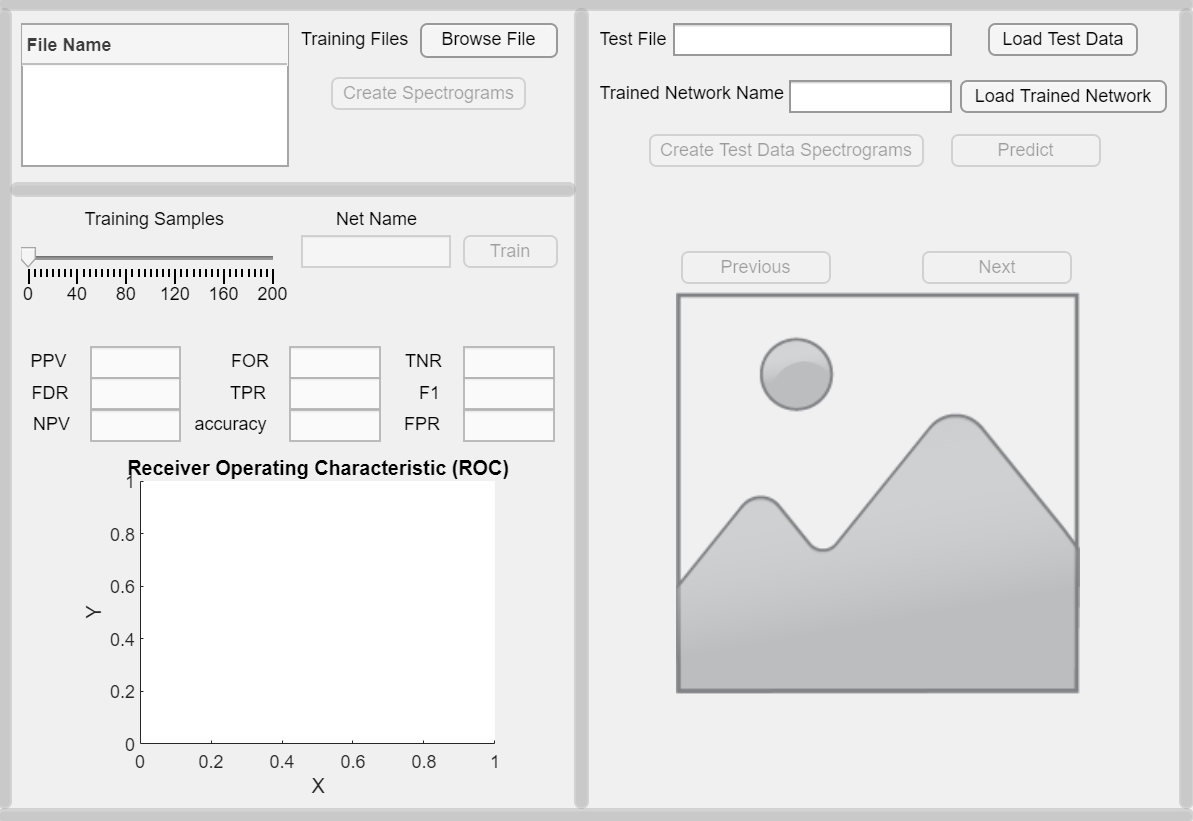
The inputs at testing stage are datasets which looks like the original training dataset, except they have no label. At this point, the data must be converted again into TFRs and then exported into spectrograms. The process of converting the dataset into TFRs are the same as that when dealing with the training dataset.

The output spectrograms of the testing dataset are saved under *‘testingData/’*, the folders in *‘testingData/’* are named after the testing dataset’s name. To prepare for the prediction, a trained network must also be loaded. In the normal workflows which training network takes place, the name of the trained network will be presented in the field. Predict in this case outputs first show an interactive graph where the navigation of sample indexes can be performed. Then the result from the prediction is saved in a file under the directory *‘Result/<netName>/’.* The Result folder was then analyzed statistically to see the differences between the trained net, it also helps to determined which T Files is showing which Data Object.

In the end of the testing stage, the T Files are predicted into the label pool by describing how much % of the testing file is of which Label (‘Data Object 1’,’ Data Object 2’,’ Data Object 3’).

## Graphical User Interface GUI

A user-friendly GUI was created to implement the experiment of Gabor transform as well as CNN classifier. The app is named “GUI.mlapp”. The figure below shows the GUI of this project.



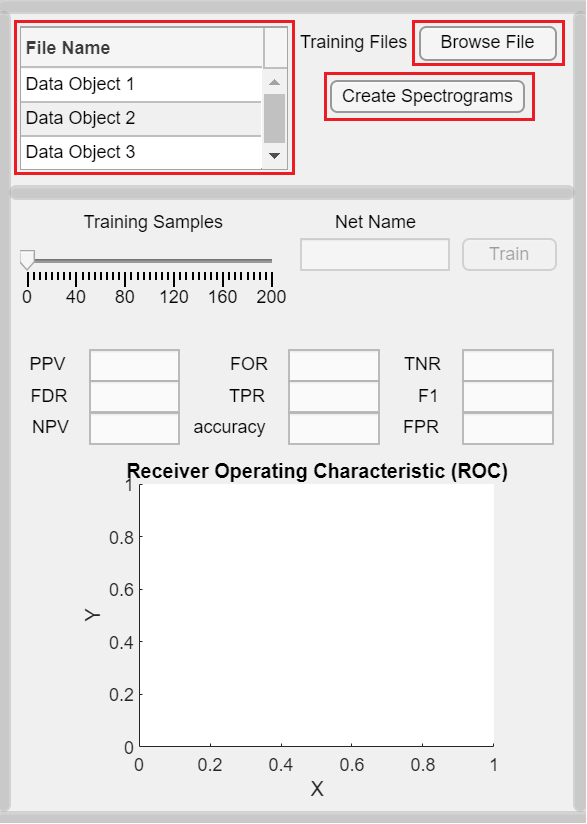
*Figure 2: GUI*

Figure 2 shows the GUI is divided into three sections:

* The first section is located at the top left corner of the frame. This one is used for loading training data files and create spectrograms for each sample in the training data.
* The section which is used for training and validating the data is right under the first section.
* The last section is situated at the left panel of the GUI. This section indicates the predicted results of test data files.

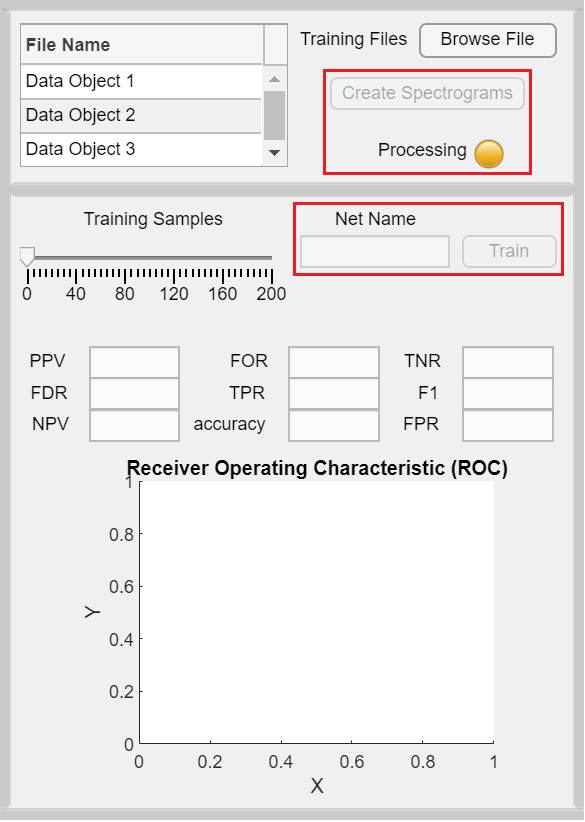
The following is a short manual instruction in order to run the GUI application:

* ***Step 1***: First and foremost, user needs to load the training files by pressing the button “Browse File” as shown in Figure 3. Currently, the training dataset includes three files “Data Object 1”, “Data Object 2”, “Data Object 3” which are located inside folder “dataset”. User can choose one file, two files or three file files at the same time for training purpose. After choosing training files, the names of all chosen files are illustrated in the table to help the user can check it again.



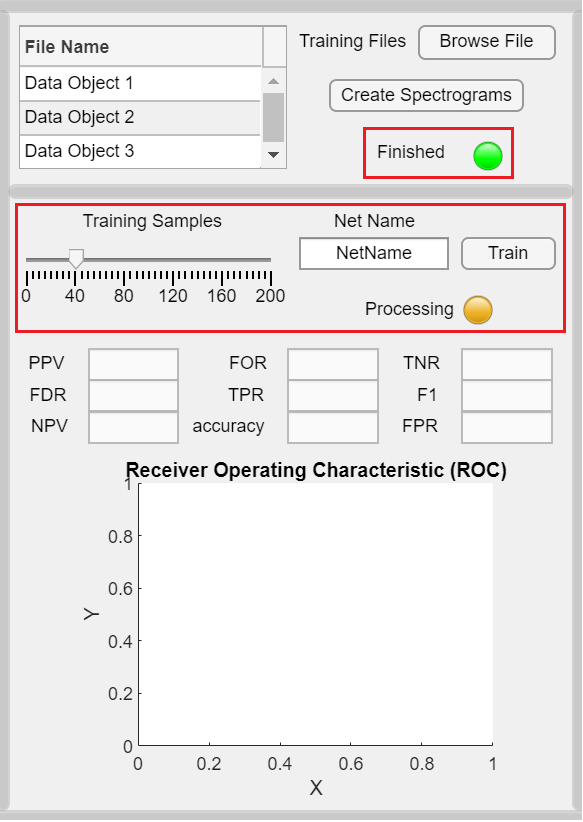
*Figure 3: Choosing the training files*

* ***Step 2***: Press the button “Create Spectrogram” to generate all spectrograms from the “Data Object” files. All the images are stored in folder “trainingData”, which is created automatically by Matlab. There is a LED in the lower area of the “Create Spectrogram” button to illustrate status of the process. As depicted in the Figure 4, the LED shows orange and the status indicates “Processing” while the program is running. At this state, the “Net Name” field in section 2 is disabled.



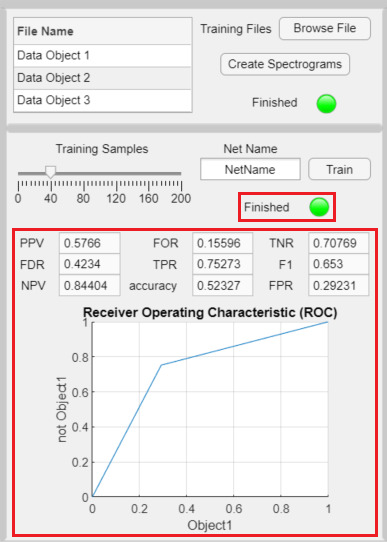
*Figure 4: Creating spectrograms for training data*

* ***Step 3***: After generating spectrograms, the LED changes to green and the status also alternates to “Finished”. There is also a notification to show that the process of creating images is finished. Then, the user moves to the second section for the training data purpose. Before pressing the “Train” button to start the training process, user must fill in the “Net Name” field as well as specify the “Training Samples” value using the slider as indicated in the Figure 5. A second orange LED with status “Processing” are also displayed.



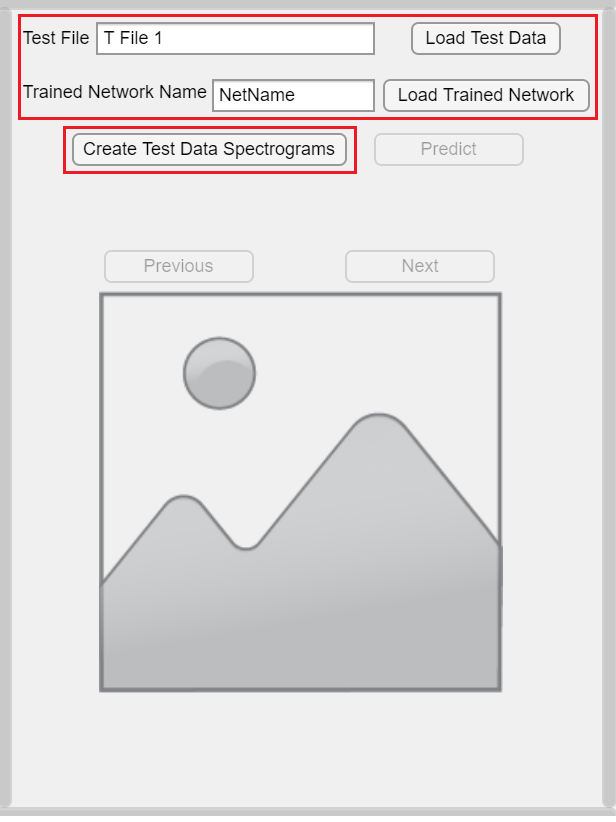
*Figure 5: Training Process*

* ***Step 4***: The training process is done when the LED is green, and the status is “Finished”. The Figure 6 illustrates the statistic validation as well as the graph receiver operating characteristic after the training process. The validation process also compels the data to have its label to generate the confusion matrix (pop up window) and the Receiver Operating Characteristic (ROC). Validation needs a loaded CNN with appropriate resolution with respect to the images.



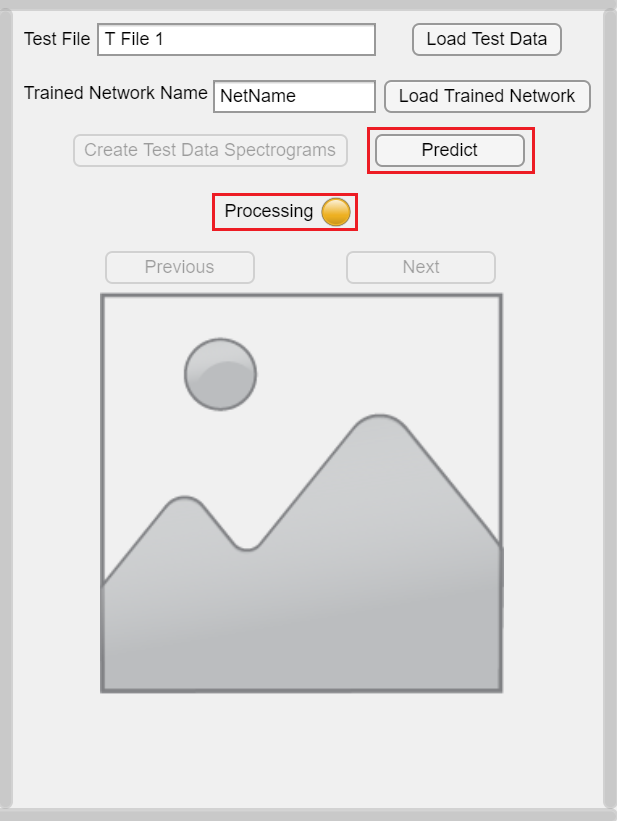
*Figure 6: Training results*

* ***Step 5***: shows the third section of the GUI. The “Test File” field requires the user to input the test .xlsx file, which is located in the “dataset” folder, by pressing the “Load Test Data” button. Then, the user presses the “Load Trained Network” button to load a trained network from the current directory, the name of the file “.mat” should be inputted. At the beginning, the “Create Test Data Spectrograms” is unable to press because there is no test data file. Therefore, the “Test File” and the “Trained Network Name” must be fulfilled to be able to proceed to the next step. Press the “Create Test Data Spectrograms” button when it is enabled.



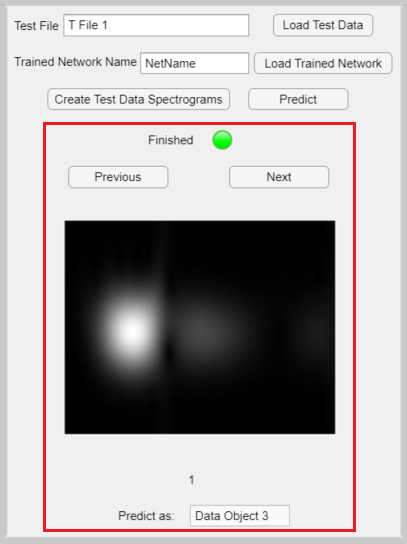
*Figure 7: Choosing test file for prediction process*

* ***Step 6***: The data of test file is read into images under “testingData” folder, this data is then be loaded into the program. Eventually, the “Predict” button is able to be pressed. The predict process starts after the button is pressed and the third orange LED appears to show the “Processing” status as depicted in Figure 7.



*Figure 8: Press the Predict button*

* ***Step 7***: Figure 8 depicts the result after the predict process is done. The LED turns to green and the status is “Finished”. The result pictures as well as the predict output are displayed in the lower area of the LED. The pictures can be browsed using the 2 buttons “Previous” and “Next”.



*Figure 9: Result of the prediction process*

*NOTE*: If the user already has the trained network, step 1 to step 4 can be skipped and user can start directly at step 5 for the prediction purpose only. If the user only wants to train a new network, then please perform only step 1 to step 4 and ignore step 5, step 6 and step 7.

# Experiment

## Experiment’s result

(Riyad)

## Result on Given Test data

(Riyad)

# Conclusion

# Further Development

This Project has opened a variety of development to proceed due to its modular arrangement in code and design.

## Different types of Spectrogram

With changes added to the output settings of each Spectrograms, the provided system can access a variety of different output networks. Instruction for the changes can be found in the above section.

It is also worth mentioning here that CNN can work with multiple representations of a signal. Aka. It can take 2 or more spectrograms as input for 1 label. This method also includes modification in the export of spectrogram. However, the step for including this code is not yet applied to the current state of the project.

Additionally, adjusting the output images in different size may also draw a better result in classification.

## Streaming Networks

The current model CNN works on static data that was recorded and predict a static test dataset. This can be extended to predicting a real time reflected signal reading, with delays in multiples of window length. The real time graphing and predicting may be a possible achievement in an upcoming project from this state.

A notice here is that CNN may performs in lesser accuracy than the state-of-the-art technology in predicting real-time stream such as LSTM or RNN. Which is also a open path for development.

## Sufficient dataset

Experimentally, the project was found to be lack of training data. The provided data (315 for ‘Data Object 1’, 200 for ‘Data Object 2’ and 400 for ‘Data Object 3’) is not clean and large enough for the sufficient training of the network.

The process of taking the data was not available from the authors at the time of this work. Creating a network with an understanding about the system would draw better results in designing a network for that system.

##### References

1. "Gabor transform - Wikipedia", *En.wikipedia.org*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Gabor\_transform#cite\_note-1. [Accessed: 29- Sep- 2021].
2. D. Gabor, Theory of Communication, Part 1, J. Inst. of Elect. Eng. Part III, Radio and Communication, vol 93, p. 429 1946 (<http://genesis.eecg.toronto.edu/gabor1946.pdf>)