Classification of time signals by CNN using spectrogram

Thanh Toan, Truong: tthanh@stud.fra-uas.de - 1185050

Hoang Hai, Pham: hhoang@stud.fra-uas.de - 1184763

Phan Bao Viet, Nguyen: viet.nguyenphanbao@stud.fra-uas.de - 1235188

Riyad-Ul-Islam: 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 are treated as images was achieved via the usage of a Convolutional Neural Network (CNN). In the end, the designed system can accomplish 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

I. Introduction

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

II. Literature Review

A. Gabor Transformation

Initially, all datasets provided for the subject are sets of analog signals in the time domain. However, the interest of this project lies in the change of frequency spectrum with respect to the time of the reflected signal from the object or the change of frequency spectrum in a 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 the time domain to time-frequency representations (TFRs).

In a quick overview, Gabor transforms first filters the signals with a Gaussian window. The remaining 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 transforms for each window with specific begin timestamps formed together with a TFRs. these TFRs can be presented in the form of a spectrogram. The following formula is the applied filter as discussed:

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

Fig. 1. Gabor Transform formular

From the formula, the Fourier Transform and the Gaussian filter can be observed in two different multiplicators, with the Gaussian filter is the exponential which contains $\boldsymbol{\tau}$. The shift $\boldsymbol{\tau}$ represents 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 of what is happening on the frequency characteristic (strength of sinusoidal frequency and phase) of a change signal in the time domain [1].

B. Convolutional Neural Network

The concept of a neural network or artificial neural network is commonly known as a network of different layers connected to each other to direct the transversal stream of information. Specifically, the task of an ANN involves calculating a vector of input from the input neuron along to the output neuron, which may function as a classifier. Biologically speaking, the neural network is a technique that mimics approximately how a brain functions. Each layer contains various nodes that act 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. The output of the network is also an array of neurons that can either do a classification, a regression function fitting, or signal generation.

Convolutional Neural Network (CNN) is a special implementation of ANN. CNN functions by convoluting through the whole data sample with a filter, then each output of the filter will then enter the next layer of CNN as input and undergo the same process until it reaches the smallest desired length of each array. These arrays are then added up to create a final representation, which will have different labels depends on the trained data. During this process, an Algorithm called Back Propagation was used in case the Data is Labeled to add/remove weights that connect adjacent layers, making the CNN adapt to the training data. CNN usually deals with data like Images, Pictures, Bit Arrays, Sound.

C. *Matlab:*

Matlab is a programming platform designed to make ease of use for engineer and student in scientific fields. The program comes with a variety of toolboxes for specific problems and fields, which saved a lot of time in development and testing.

This work took advantage of the available toolbox in Matlab that enables Deep Learning for its generation and usage of CNN in solving the problem. The developed work also uses the GUI API of Matlab to create an interactive testing environment for further use.

III. Detailed Approach and Implementation

A. *Overview*

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

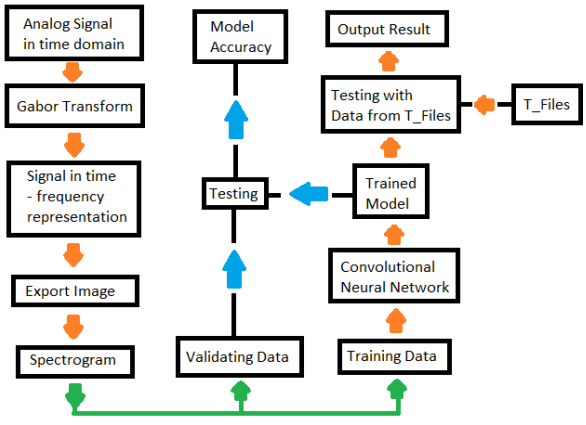


Fig. 2: Project pipeline

In Summary, the code for the project was divided into 3 stages. Stage one involves the 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.

B. 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.

- *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 file, 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 file:

TABLE I. NUMBER OF SAMPLES

Label	Number Of Samples
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 was 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 are 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 project focused more on developing a pipeline for dealing with reflected time-series signals of objects rather than reaching a good result of training.

	A	B	C	D	E	F	G	H
1	2	4	99	5795	23	3400	0,000384	0,00064
2	2	4	96	5532	23	3400	-0,00371	-0,00397
3	2	4	96	5512	23	3400	-0,00064	-0,00051
4	2	4	64	1843	23	3400	-0,00115	-0,0009
5	2	4	55	883	23	3400	-0,00218	-0,00192
6	2	4	51	400	23	3400	0,016895	0,011903

Fig. 3: Dataset samples excel view

- *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 a 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 as the name of the used trained network.

C. *Gabor Transform and Creation of Spectrogram*

To do have a mathematical 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 hold folders of spectrograms – Fig. 4.*

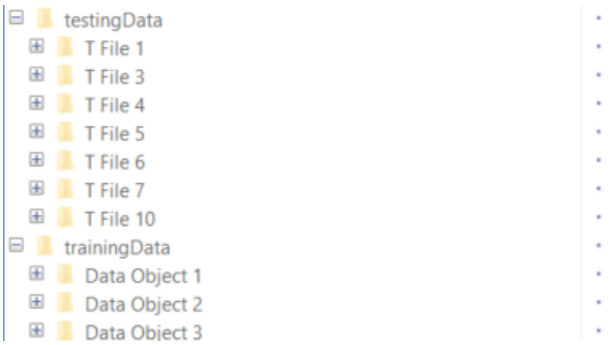


Fig. 4. Directory example of output spectrograms

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 interchangeably. It serves the purpose of transforming time-series signal samples into time-frequency representations (TFRs).

```
table = readtable("dataset/" + filename(i) + ".xlsx");
data_range = table(:,7:end); % rows 1 to the end and columns 7th to the end
```

In the above code snippet, 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

```
dataLabel = filename(i);
```

Because there is only one file 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 is 'Data Object1', 'Data Object 2', and 'Data Object 3'. So, at this state, we have a classification of multiple objects.

```
msgbox("Performing Gabor Transformation to
n = width(data_range); % Number of samples
t = (1:n); % Time vector 1:3400 samples
mkdir("trainingData/" + dataLabel);
```

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

```
for c = 1:height(data_range)
    close all
    S = table2array(data_range(c,1:width(data_range)));
    S = S./max(max(S),abs(min(S))); % normalizing
    L = 10; %Signal duration = 10s assumption
    k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
    Sgt_spec = [];
```

The code above demonstrates the iteration through all samples of one training file, which are now stored in variable **data_range**. Because the spectrogram was generated via exporting an image from a graph, multiple windows of the plot will be 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 to 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** uses **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_spec** 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 discusses the iteration that simulated Gabor transforms in this project. The instruction video of Mr. Nathan Kutz demonstrated the process beautifully. In his online course "Inferring structure of complex system" by the Department of Applied Mathematics at the University of Washington [2]. The code for Gabor Transform in this project is based on his Matlab demonstration.

```
tslide = 0:20:n; % Moving Gabor filter every 20 samples !
for j = 1:length(tslide)
```

In the above code, the iteration of the Transform is based on the index j, running from 1 to the end of **tslide**'s length.

tslide was used to take the indexes which belong 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 transforms the signal in each window respectively. In this experiment, the window moves 20 samples at a time.

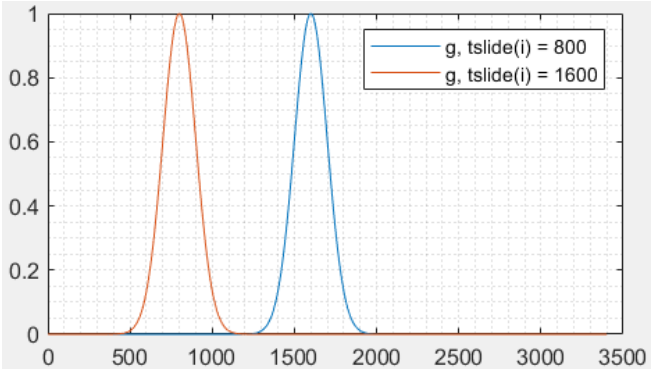


Fig. 5. Gabor Gaussian filter demonstration plot.

The code below shows the windowing function. **tslide** represents the shift in the time scale, the Fig. 5 shows a better representation of the function **g** when plotted with different **tslide** values.

```
g = exp(-5*1e-5*(t-tslide(j)).^2);
Sg = g.*S;
```

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

```
Sgt = fft(Sg);
Sgt_spec=[Sgt_spec; abs(fftshift(Sgt))];
```

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:

TABLE II. PARAMETERS USED IN PLOTTING SPECTROGRAMS

Parameter	Usage
tslide	Starting timestamp of each window, used for indexing in the 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 the color representation of pixels

```
Sgt_spec=Sgt_spec'; % transpose the Spectrogram
pcolor(tslide,ks,Sgt_spec);
shading(gca, 'interp');
set(gca,'Ylim',[30 60]);
colormap(gca,'gray');
set(gca, 'Visible', 'off');
```

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 Mathwork's official website [3].

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 Fig. 6 optimizes for the fastest training time at the moment. To illustrate this process of choosing the settings for the spectrogram, the first sample of 'Data Object 1' is utilized.

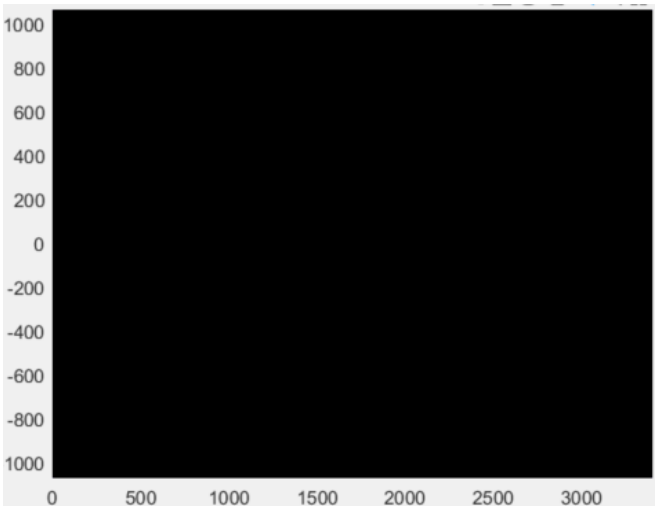


Fig. 6. Spectrogram before scaling.

Fig. 6 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 Fig. 7.

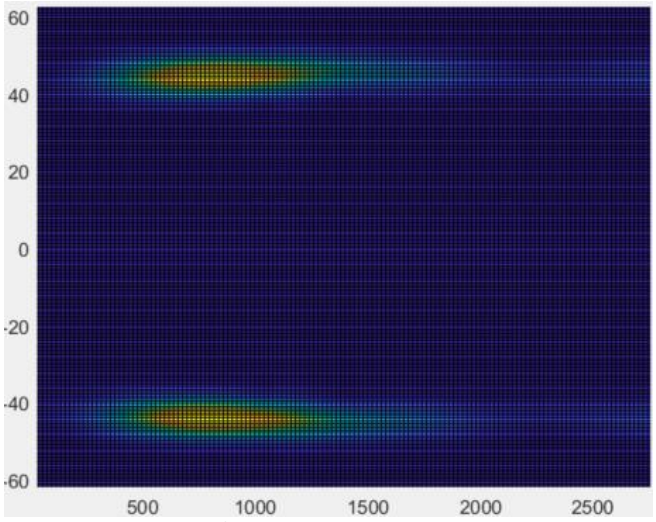


Fig. 7. Zoomed version of Fig. 6 on the graph.

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 explains the choice of the frequency range **ks** or **Y** here scaled to [30 60], the output can be seen in Fig. 8.

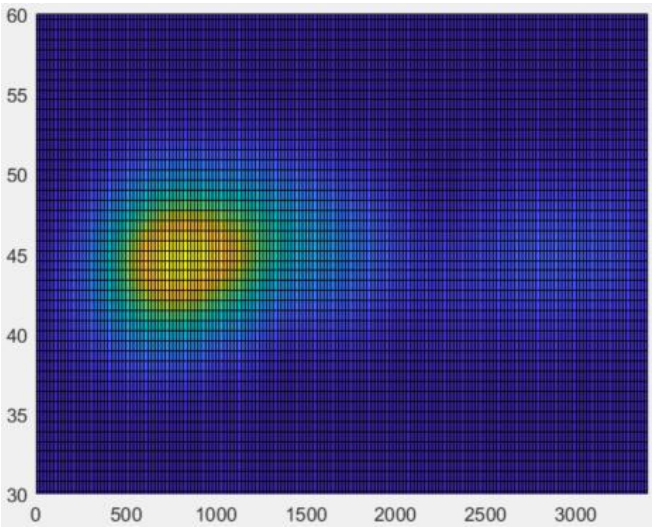


Fig. 8. Spectrogram with YAxis scaled to [30 60].

The Spectrogram at this point posed to have a 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, Fig. 9 shows the output spectrogram after using interpolated shading settings.

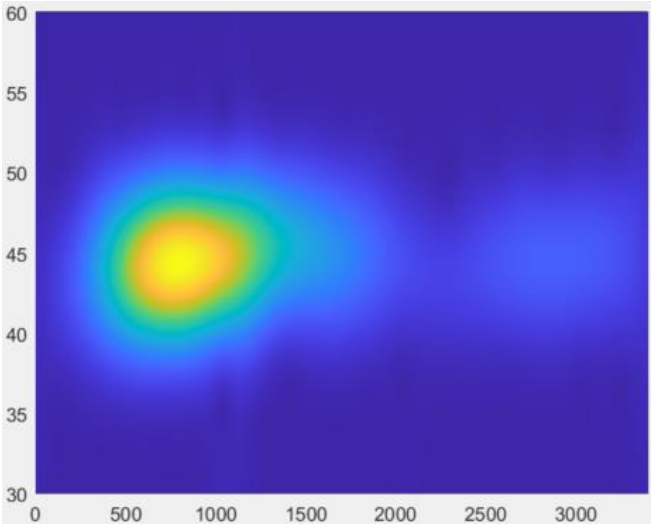


Fig. 9. Interpolated spectrogram.

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 is 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 Fig. 9 is the 'default' option, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14, Fig. 15 show some available options that might be of interest for further research about which map draws the best performance.

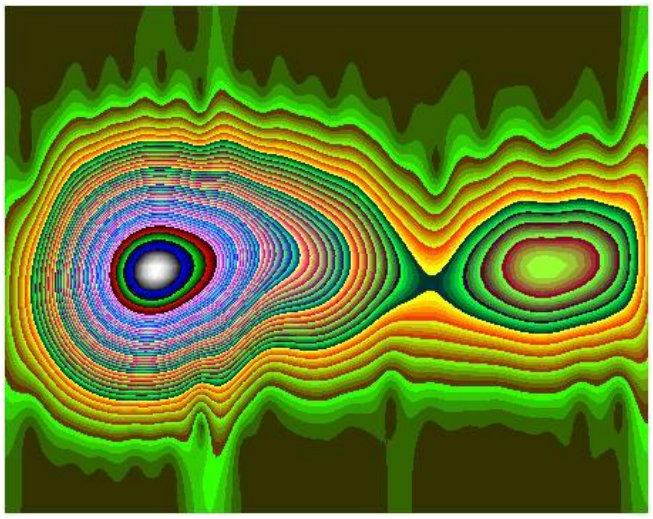


Fig. 10. colorMap 'colorcube'

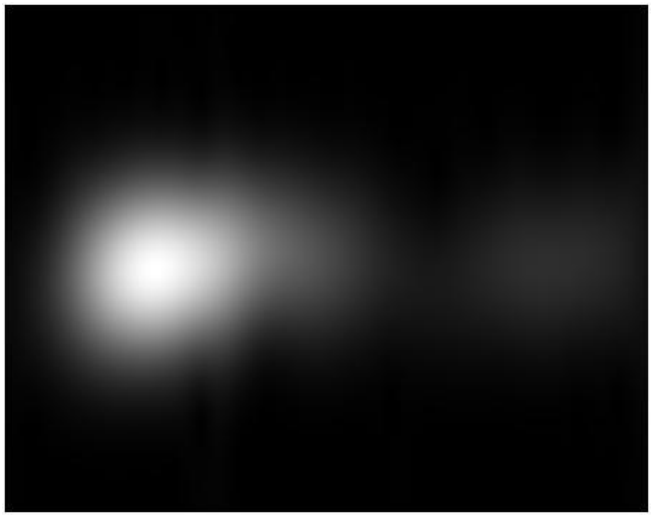


Fig. 11. colorMap 'gray'

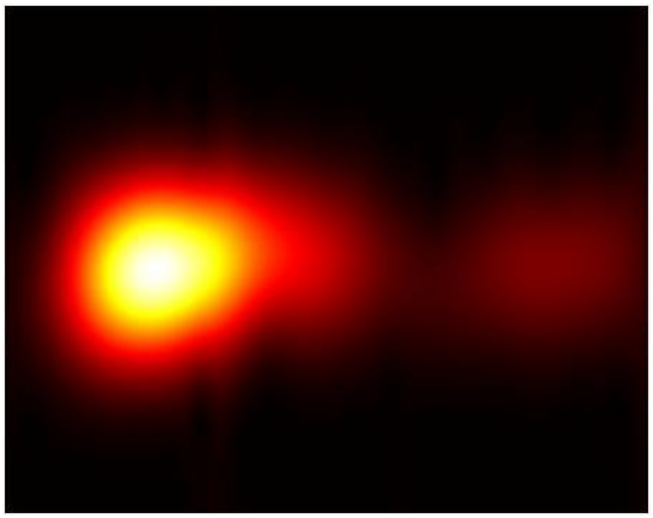


Fig. 12. colorMap 'hot'

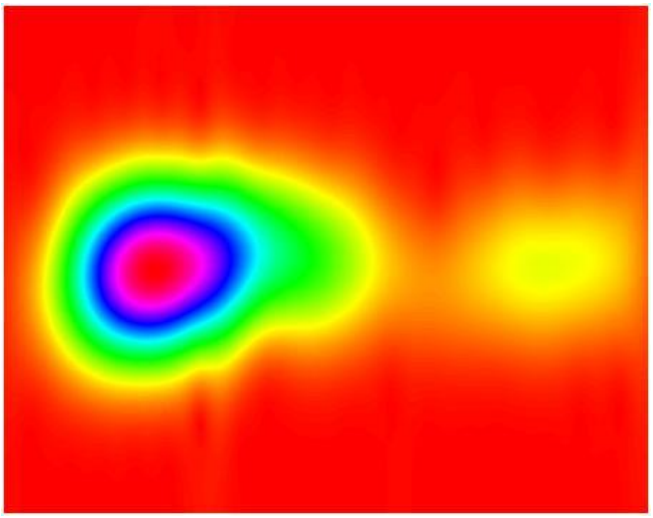


Fig. 13. colorMap 'hsv'

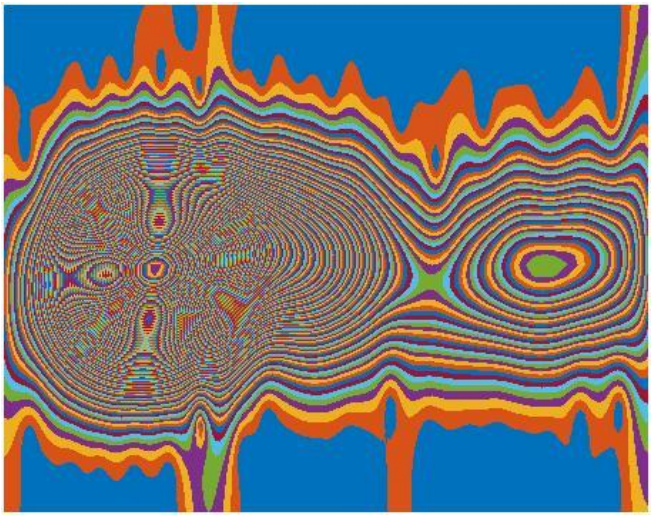


Fig. 14. colorMap 'lines'

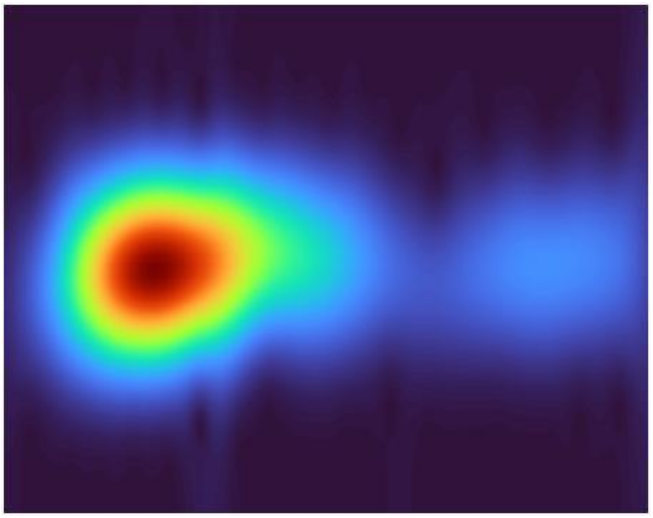


Fig. 15. colorMap 'turbo'

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

```
exportgraphics(gca,"testingData/" +  dataFile + "/" + ...
    dataFile + "_" + c + ".jpg", 'Resolution', 100)
```

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

TABLE III. DPI AND CORRESPONDING OUTPUT SPECTROGRAM SIZE

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

NOTICE: The output of the experiment are trained networks which are specific to a type of exported spectrogram. A network trained for one type of spectrogram can only be used to predict the spectrograms that share the 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.

D. Training the Convolutional Neural Network

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

```
% count number of subfolder in trainingData
app.currentFolder = pwd;
trainingFolder = dir(app.currentFolder + "\trainingData");
num_layer = length(trainingFolder)-2;
```

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 the above code snippet shows the process of getting the number of labels via the number of directories within the current directory.

```
% Initialize the network with fixed value before training
app.layers = [
    imageInputLayer(size(readimage(app.imds,1)))

    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(num_layer)
    softmaxLayer
    classificationLayer];
```

The code above represents a simple CNN model, which is used for the task of classifying different objects. The following section discusses some definitions of the layers and their functionalities in building the CNN model. These options are from Matlab ‘s Deep Learning Toolbox. The

documentation can be accessed on the Mathworks website for more clarity [4].

- **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 currently 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 are in the layer, this will be the number of output features that will be the inputs to the next layer. The '*Padding'* option *'same'* ensures 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 is 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 from the last layer and has at the 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 plays the normalizing role 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 belongs to in the pool of learned Labels.

After the initialization of the network structure, the learning behaviors of the network **app.options** are specified, for an overview see the code below.

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

There are a lot of options 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 runs 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 the 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 the 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 the below code.

```
app.net = trainNetwork(app.imdsTrain, app.layers, app.options);
app.nameOfCurrentTest=app.NetNameEditField.Value;
app.TrainedNetworkNameEditField.Value = app.nameOfCurrentTest;
net = app.net;
save([app.nameOfCurrentTest,'.mat'],'net');
```

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 by Prof Pech.

E. Validation of trained network

Validation of trained networks happens right after the training of the CNN. The current validating process uses 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 the validation stage is images (spectrograms), which are already created together with the training data. They were split from the original training set. The validation stage in this project includes the computation of the following parameters:

TABLE IV. ABBREVIATION FOR VALIDATION 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 based on '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.

F. 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 its validation result saved under /images/<netName>. This infrastructure separates the training process and the Testing process, which saves time when conducting and documenting multiple experiments. For details on the Testing procedure, see the description of steps in the GUI section.

The inputs at the testing stage are datasets that look 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 is 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 in which the 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.

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

IV. GRAPHICAL USER INTERFACE - GUI

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

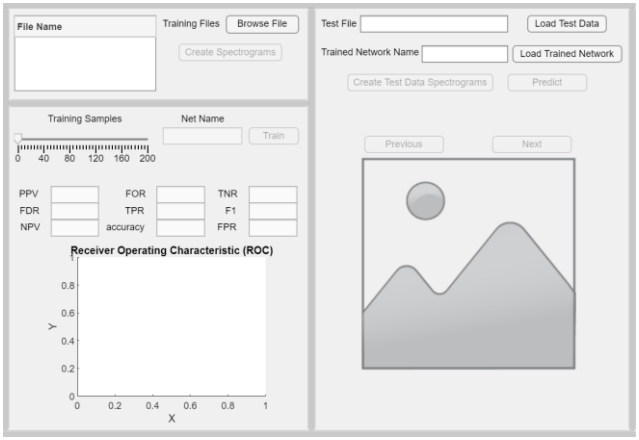


Fig. 16. Graphical User Interface (GUI)

Fig. 16 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, the users need to load the training files by pressing the button "Browse File" as shown in Fig. **17**. Currently, the training dataset includes three files "Data Object 1", "Data Object 2", "Data Object 3" which are located inside folder "dataset". Users can choose one file, two files, or three file files at the same time for training purposes. After choosing training files, the names of all chosen files are illustrated in the table to help the user can check them again.

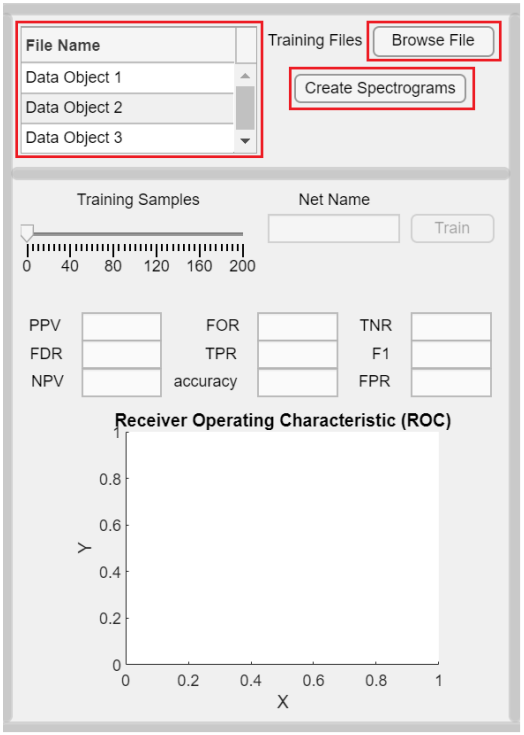


Fig. 17. Choosing the training files

- ***Step 2***: Press the button "Create Spectrogram" to generate all spectrograms from the "Data Object" files. All the images are stored in the folder "trainingData", which is created automatically by Matlab. There is a LED in the lower area of the "Create Spectrogram" button to illustrate the status of the process. As depicted in Fig. **18**, the LED shows orange and the status indicates "Processing" while the program is running. At this stage, the "Net Name" field in section 2 is disabled.

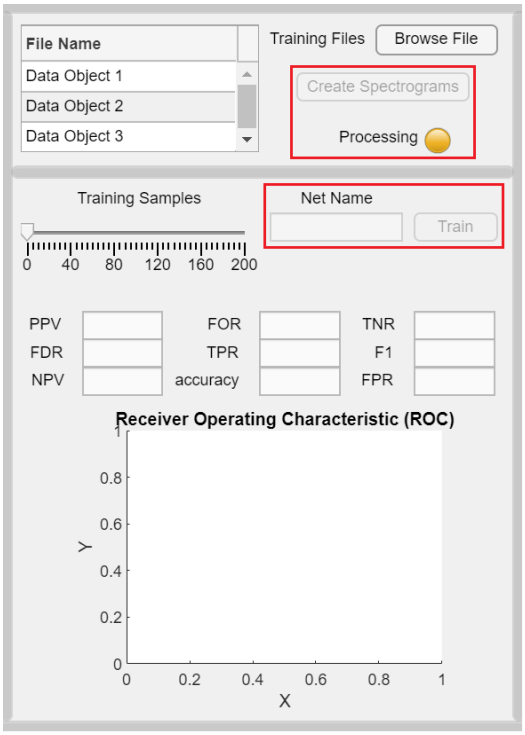


Fig. 18. Creating spectrograms for training data

- ***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, users must fill in the "Net Name" field as well as specify the "Training Samples" value using the slider as indicated in Fig. **19**. A second orange LED with the status "Processing" is also displayed.

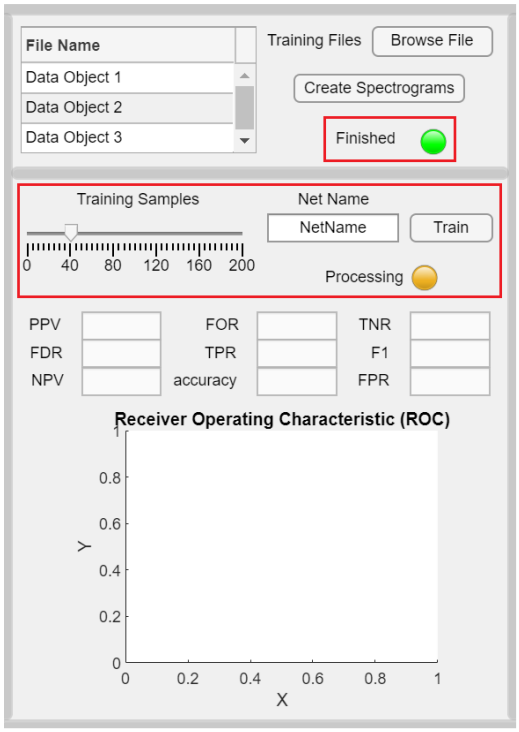


Fig. 19. Training Process

- ***Step 4***: The training process is done when the LED is green, and the status is "Finished". Fig. **20** 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 an appropriate resolution with respect to the images.

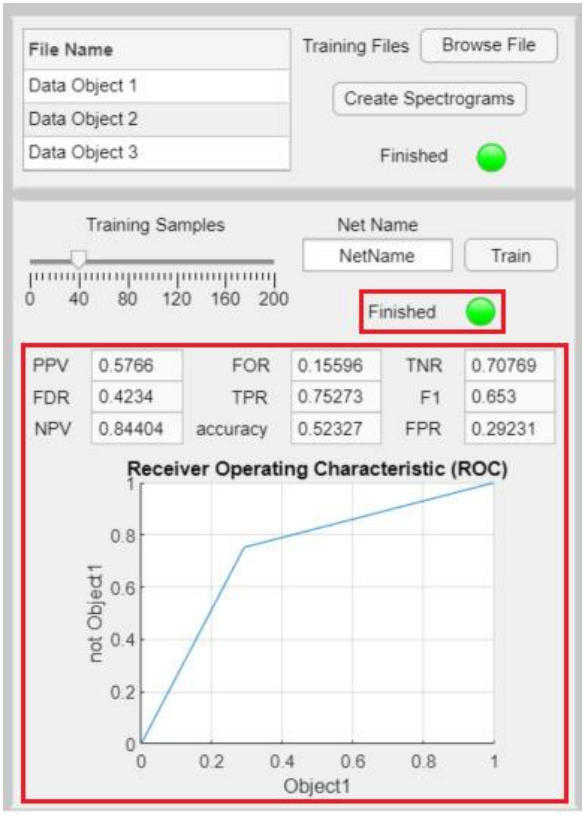


Fig. 20. Training results

- ***Step 5***: Fig. 21 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. In 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.

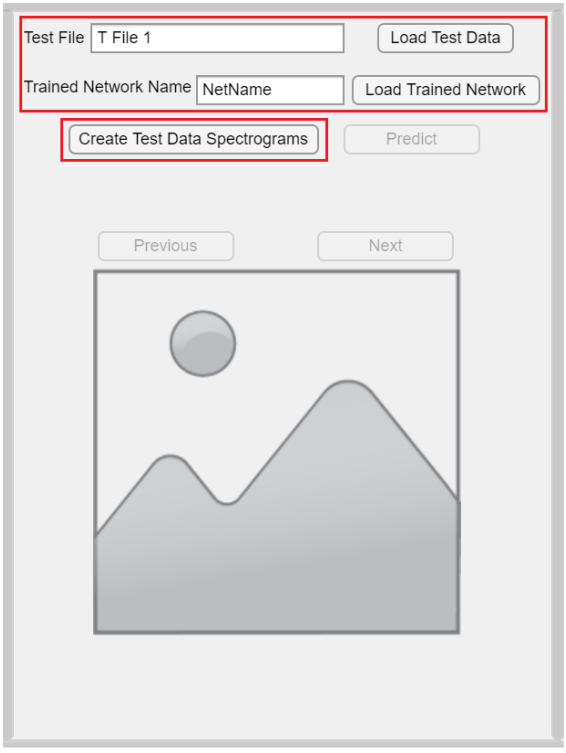


Fig. 21. Choosing a test file for the prediction process

- ***Step 6***: The data of the test file is read into images under the "testingData" folder, this data is then be loaded into the program. Eventually, the "Predict" button is able to be pressed. The prediction process starts after the button is pressed and the third orange LED appears to show the "Processing" status as depicted in Fig. 22.

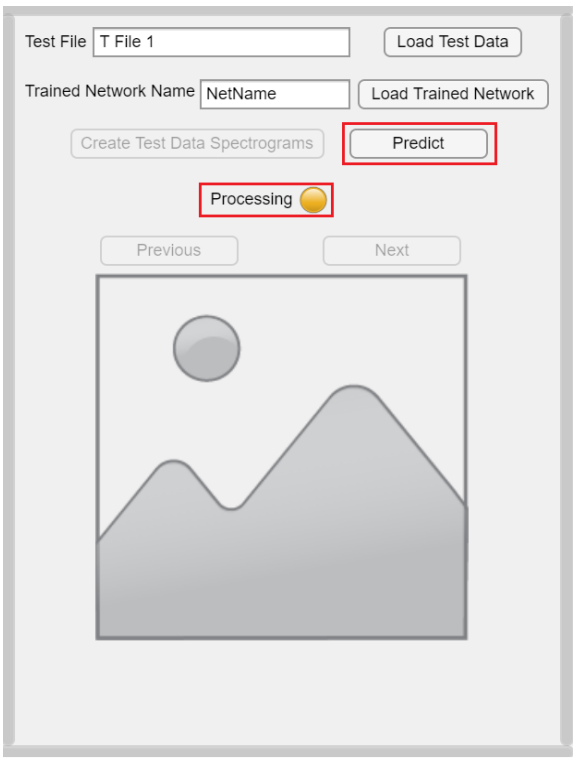


Fig. 22. Press the Predict button

- ***Step 7***: Fig. 23 depicts the result after the prediction process is done. The LED turns to green and the status is "Finished". The result pictures as well as the predicted output are displayed in the lower area of the LED. The pictures can be browsed using the 2 buttons "Previous" and "Next".

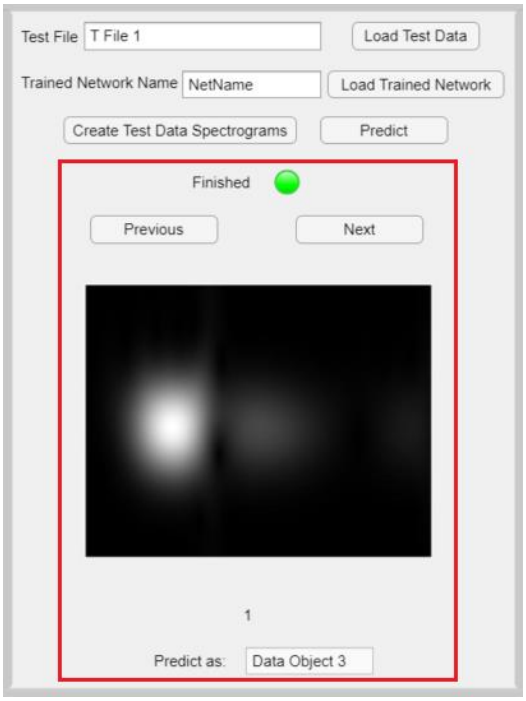


Fig. 23. Result of the prediction process

NOTE: If the user already has the trained network, step 1 to step 4 can be skipped and users 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.

V. Experiment

A. *Experiment's result*

Following the performance analysis, further statistics such as the confusion matrix, 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) and accuracy are provided for each resolution mentioned above in the following Figures.

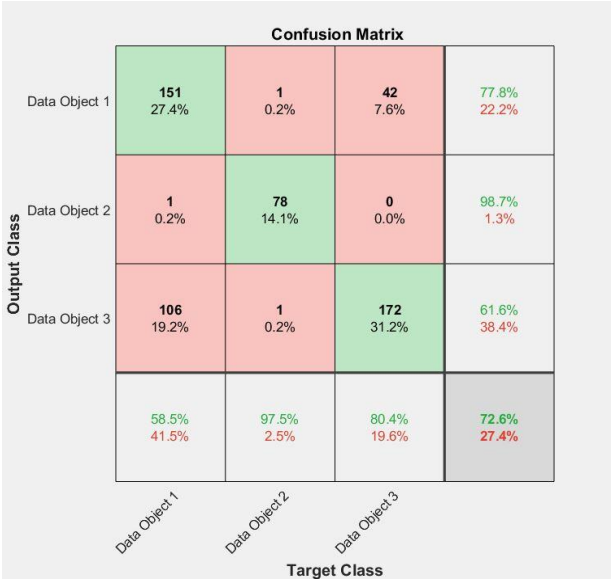


Fig. 24. Confusion Matrix of test_net network experiment.

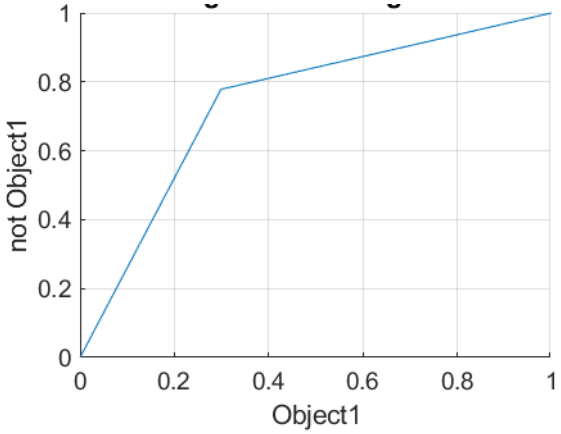


Fig. 25. Receiver Operating Characteristic (ROC) of test_net network experiment.

TABLE V. TEST_NET NETWORK EXPERIMENT FURTHER STATISTICS

Statistic	Value
Accuracy	72.65%
PPV	0.59
FDR	0.414
NPV	0.85
FOR	0.15
TPR	0.78
TNR	0.7
F1	0.67
FPR	0.3

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

For the second experiment with the network net50, the confusion matrix and other statistical results are shown.

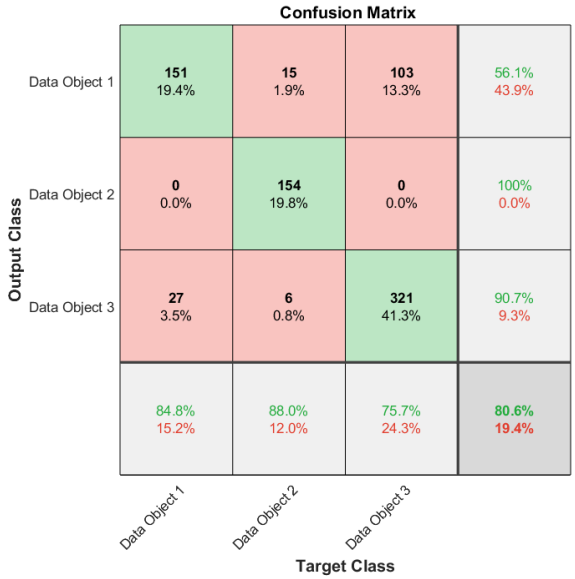


Fig. 26. Confusion Matrix of net50 network experiment.

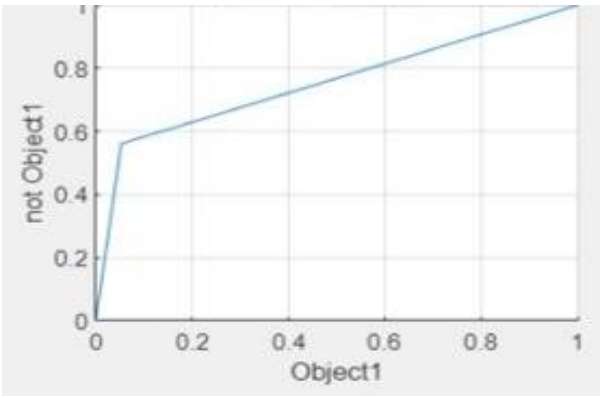


Fig. 27. Receiver Operating Characteristic (ROC) of net50 network experiment.

TABLE VI. NET50 NETWORK EXPERIMENT FURTHER STATISTICS

Statistic	Value
Accuracy	80.56%
PPV	0.85
FDR	0.152
NPV	0.8
FOR	0.2
TPR	0.56
TNR	0.95
F1	0.68
FPR	0.05

The given Data is for the third training experiment with network net80_20ep done with a sampling of 80.

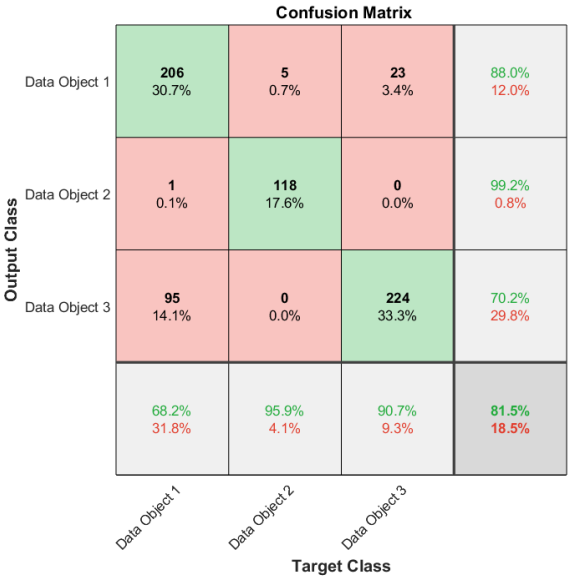


Fig. 28. Confusion Matrix of net80_20epoch network experiment.

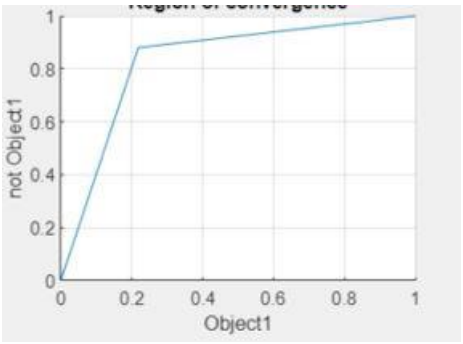


Fig. 29. Receiver Operating Characteristic (ROC) of net80_20epoch network experiment.

TABLE VII. NET80_20EPOCH NETWORK EXPERIMENT FURTHER STATISTICS

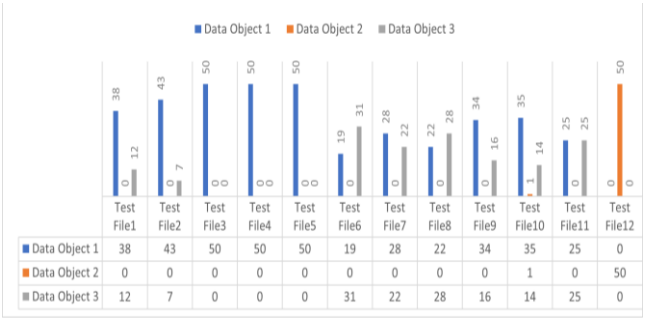
Statistic	Value
Accuracy	81.58%
PPV	0.68
FDR	0.32
NPV	0.92
FOR	0.08
TPR	0.88
TNR	0.78
F1	0.77
FPR	0.21

Overall, the network accomplished nearly perfect performance. Better 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. The Training Run of the networks can be found at the end of this paper.

B. The result on Given Test data

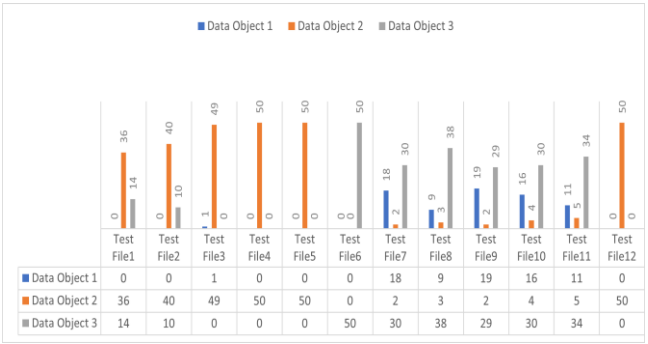
After applying the given test data on the trained network, the classifier predicts 3 data objects samples in Fig. 30 Fig. 31, and Fig. 32. The results are inferred from the three labels. From the graphs, we can see all the trained networks agree that T File 12 is Data Object 2. T File 6 to T File 11 seems to have an ambiguity between Data Object 1 and Data Object 3. T File 1 to 5 is predicted to be not Data Object 3. However,

they don't distinguish between Data Object 1 and Data Object 2.



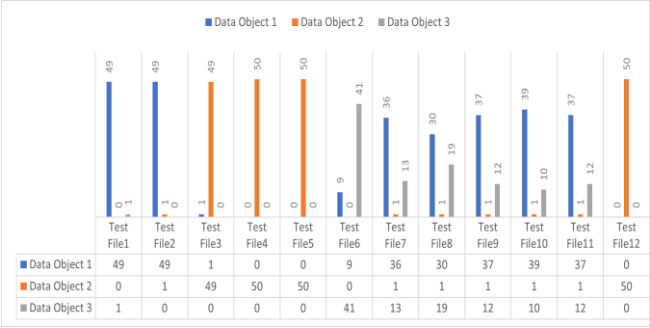
	Test File1	Test File2	Test File3	Test File4	Test File5	Test File6	Test File7	Test File8	Test File9	Test File10	Test File11	Test File12
Data Object 1	38	43	50	50	50	19	28	22	34	35	25	0
Data Object 2	0	0	0	0	0	0	0	0	0	1	0	50
Data Object 3	12	7	0	0	0	31	22	28	16	14	25	0

Fig. 30. Data Object prediction from testing data for test_net network experiment.



	Test File1	Test File2	Test File3	Test File4	Test File5	Test File6	Test File7	Test File8	Test File9	Test File10	Test File11	Test File12
Data Object 1	0	0	1	0	0	0	18	9	19	16	11	0
Data Object 2	36	40	49	50	50	0	2	3	2	4	5	50
Data Object 3	14	10	0	0	0	50	30	38	29	30	34	0

Fig. 31. Data Object prediction from testing data for net50 network experiment.



	Test File1	Test File2	Test File3	Test File4	Test File5	Test File6	Test File7	Test File8	Test File9	Test File10	Test File11	Test File12
Data Object 1	49	49	1	0	0	9	36	30	37	39	37	0
Data Object 2	0	1	49	50	50	0	1	1	1	1	1	50
Data Object 3	1	0	0	0	0	41	13	19	12	10	12	0

Fig. 32. Data Object prediction from testing data for net80_20epoch network experiment.

VI. Conclusion:

The project has created a pipeline for classifying reflected signals of Data Objects. Through this project, the formulation of the spectrogram had been implemented in a Matlab environment. A CNN model was used for the classification of the samples. The project also ended up creating a GUI for an interactive session when working with the time series of object reflected signals. The resulting output of this project didn't reach a high accuracy, it stopped about 79-81%. The similarity in the testing dataset T Files also raised inconsistency in the prediction of the Labels using the trained sample network.

The project possesses great potential in training/ classifying object reflected signals. The work has already discussed some options that are available for tweaking the CNN before the learning. With more time, computing power, and experiments, the current model is able to achieve even more accuracy in predicting Object reflected signals.

VII. Further Development

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

A. *Different types of Spectrogram*

With changes added to the output settings of each spectrogram, 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 sizes may also draw a better result in classification.

B. *Streaming Networks*

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

Notice here is that CNN may perform with lesser accuracy than the state-of-the-art technology in predicting real-time streams such as LSTM or RNN. Which is also an open path for development.

C. *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 of the system would draw better results in designing a network for that system.

References

[1] 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)

[2] 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)

[3] [2]"pcolor", *Pseudocolor plot*, 2021. [Online]. Available: https://de.mathworks.com/help/matlab/ref/pcolor.html. [Accessed: 30- Sep- 2021].

[4] [3]"Deep Learning Toolbox", *Deep Learning Toolbox Documentation*, 2021. [Online]. Available: https://de.mathworks.com/help/deeplearning/index.html?searchHighlight=Deep%20Learning&s_tid=srchtitle. [Accessed: 30- Sep- 2021].

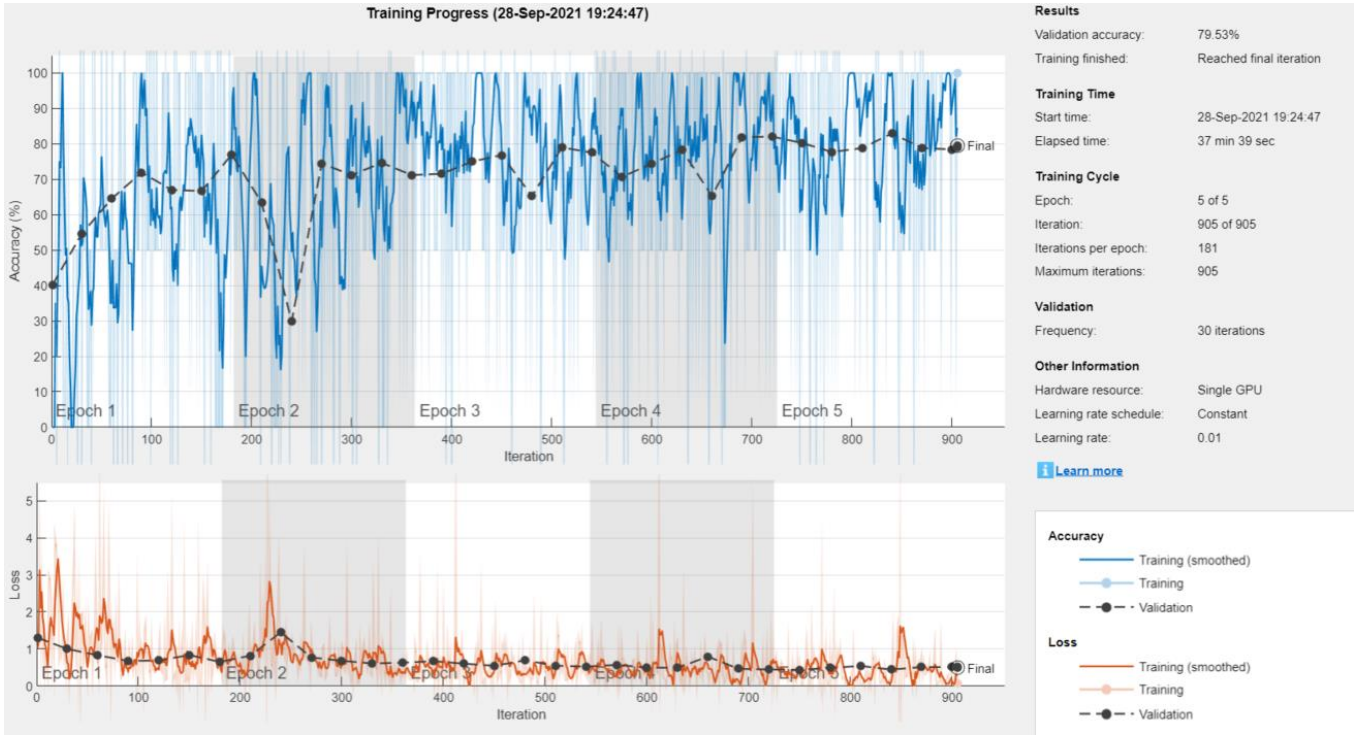


Fig. 33. Traing Run with test_net.

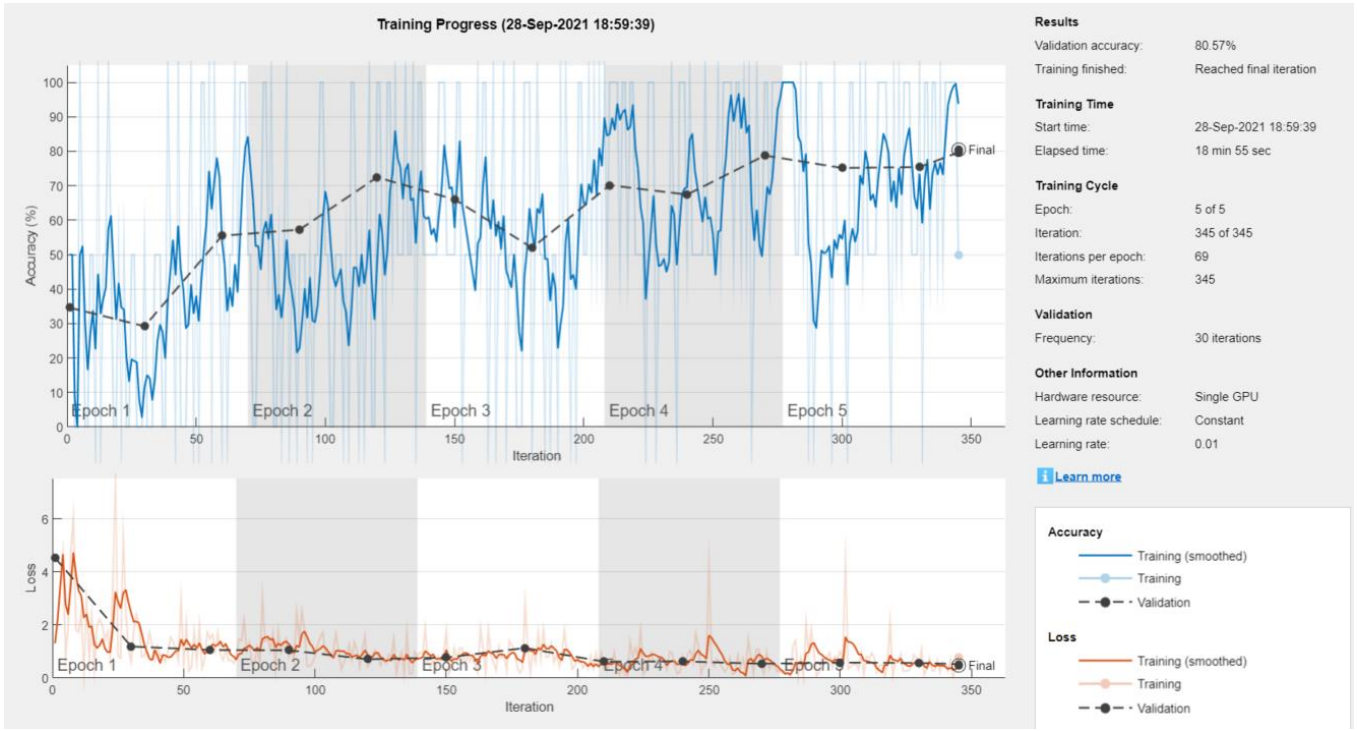


Fig. 34. Training Run with net50.

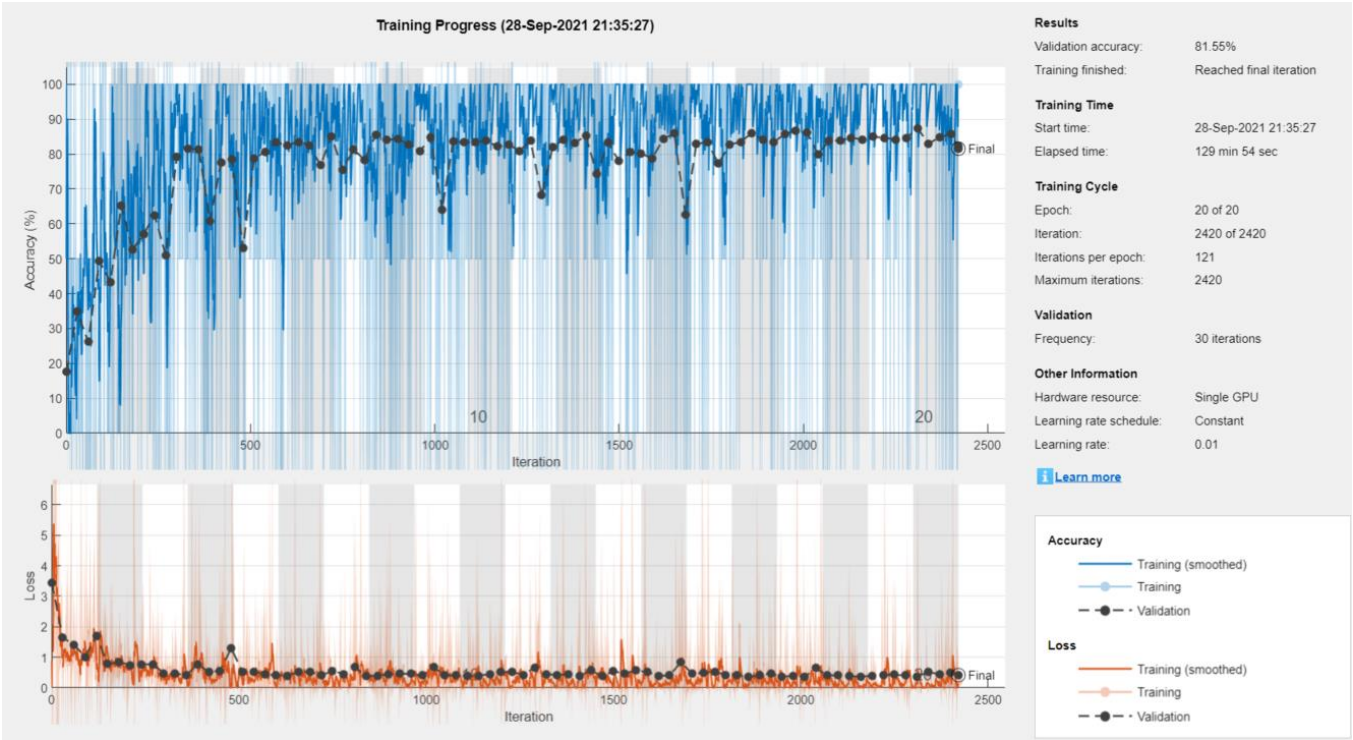


Fig. 35. Training Run of net80_20ep shows the saturation of accuracy in epochs 5,6.