Gina Cody School of Engineering and Computer Science

Department of Electrical and Computer Engineering

Project Report on
Modulation Classification using Convolutional Neural Network based on Deep Learning

A report prepared to fulfill the requirements of
COEN6331
Neural Networks

Submitted to
Dr. Kash Khorasani

Prepared by
Pranav Kumar Jha
Student ID: 40081750

28th April, 2022

Abstract

Deep learning (DL) technique is widely used in application domains such as Natural Language Processing (NLP), automatic driving, medical devices, pattern recognition etc. However, in the wireless communication domain, it has not been well explored in many areas. Modulation classification is one of the domain in communications where we can use deep learning to improve the performance of the network while reducing the complexity, simultaneously. In this project, convolutional neural network (CNN) based DL model has been used for modulation classification where the channel-impaired waveforms have been generated. Further, the CNN based DL model has been trained and tested with the generated synthetic waveforms.

Contents

1 Introduction

Radio communications is a signal processing domain where modulation recognition of wireless signals can be explored using Deep learning (DL) methods. Traditional methods have been used so far which involved signal processing techniques and machine learning methods which have few hidden layers as compared to DL networks [1].

1.1 Signal Modulations

In this project work, a convolutional neural network (CNN) based DL model has been used to analyze the modulation classification of digital as well as analog modulations [2]. These signal modulations are

- **Digital Modulations**
  - Binary phase shift keying (BPSK)
  - Quadrature PSK (QPSK)
  - 8-ary PSK (8-PSK)
  - 16-ary Quadrature amplitude modulation (16-QAM)
  - 64-ary QAM (64-QAM)
  - 4-ary Pulse amplitude modulation (4-PAM)
  - Gaussian frequency shift keying (GFSK)
  - Continuous phase FSK (CPFSK)
- **Analog Modulations**
  - Broadcast FM (B-FM)
  - Double sideband amplitude modulation (DSB-AM)
  - Single sideband amplitude modulation (SSB-AM)

2 Problem Statement

Modulation classification has always been a problem to address in communication systems involving variety of applications such as, bandwidth management, interference recognition, digital cyber attacks etc. The problem of recognizing the modulations correctly is a challenging task. To study the performance of the radio signal classification, feature extraction of specific types of signals and their properties was an important and tedious task to fulfill. DL has given us the ability to feature pre-processing where we can design feature extractors and the information can be managed using support vector machines, decision trees etc. [3–5]. This work investigates the use of a CNN based DL network to classify the modulations of different types of signal waveforms.

3 CNN based DL Network

In the neural networks, the data need to labeled first and the we use the labeled data for training the CNN based DL network. The trained network can then be tested against the remaining data from the dataset or with a completely new data for modulation classifications.

3.1 Generation of the signal waveforms

The number of frames that has been generated for each of the modulation category are 10,000 where 70% is used for training, 15% is used for validation and the remaining 15% is used for testing. The network training phase consists of training and validation frames. The test frames provides the classification accuracy of the generated waveforms. The number of samples in each frame are 2^{10} with a sampling rate of 0.2 MHz. For digital modulation, the number of samples per frame are 8 and the center frequency is 900 MHz. For analog modulation, the center frequency is 100 MHz.

Each of the generated frame is passed through Additive white gaussian noise (AWGN) channel with Rician multipath fading and a clock offset. The AWGN channel adds an SNR of 25 dB. The communications toolbox of matlab is used to implement this into the network. The Rician multipath fading channel has a delay profile of [0 1.6 3.2] samples with their corresponding path gains of [0 -3 -9]. The K-factor value is considered as 4 and the maximum Doppler shift is considered as 3 Hz. The network is implemented with a clock offset of 4. Frequency offset and sampling rate offset have also been applied for every frame. The channel parameters are provided in the Fig. 3.1. In this network, the decision is entirely based on single frame unlike video data, where multiple frames are examined at once in order to make a decision.

```
channel =

  helperModClassTestChannel with properties:

                  SNR: 25
       CenterFrequency: 902000000
            SampleRate: 200000
            PathDelays: [0 8.0000e-06 1.6000e-05]
      AveragePathGains: [0 -3 -9]
               KFactor: 4
    MaximumDopplerShift: 3
     MaximumClockOffset: 4
```

(a)

```
chInfo =

  struct with fields:

              ChannelDelay: 6
    MaximumFrequencyOffset: 3608
  MaximumSampleRateOffset: 0.8000
```

(b)

Figure 3.1: (a) Channel parameters, (b) Channel information

The trained network will have properties as shown in Fig. 3.6. The trained CNN based DL network takes 2^{10} samples and provides the prediction for each of the modulation types. 4-PAM frames have also been generated using Rician multipath fading, center frequency and a sampling time drift, and AWGN. Use following function to generate synthetic signals to test the CNN. Then use the CNN to predict the modulation type of the frames.

```
>> Mod_Class

trainedNet =

  SeriesNetwork with properties:

        Layers: [28×1 nnet.cnn.layer.Layer]
     InputNames: {'Input Layer'}
    OutputNames: {'Output'}
```

Figure 3.2: Trained CNN based DL Network

Now, to generate frames for each of the signal modulation, a loop has been created and the resulting frames data set have been stored with their corresponding labels in MAT files. These files can be used every time, without generating them again and again whenever we run the program. Few samples have been removed from the transient state to make the look more stable and to provide a random initialization point. The amplitude of the real and imaginary parts of the generated frames for each of the digital and analog modulation types are plotted against the sample number, and the corresponding spectrograms are also provided as shown below in following Figs. 3.3-3.5.

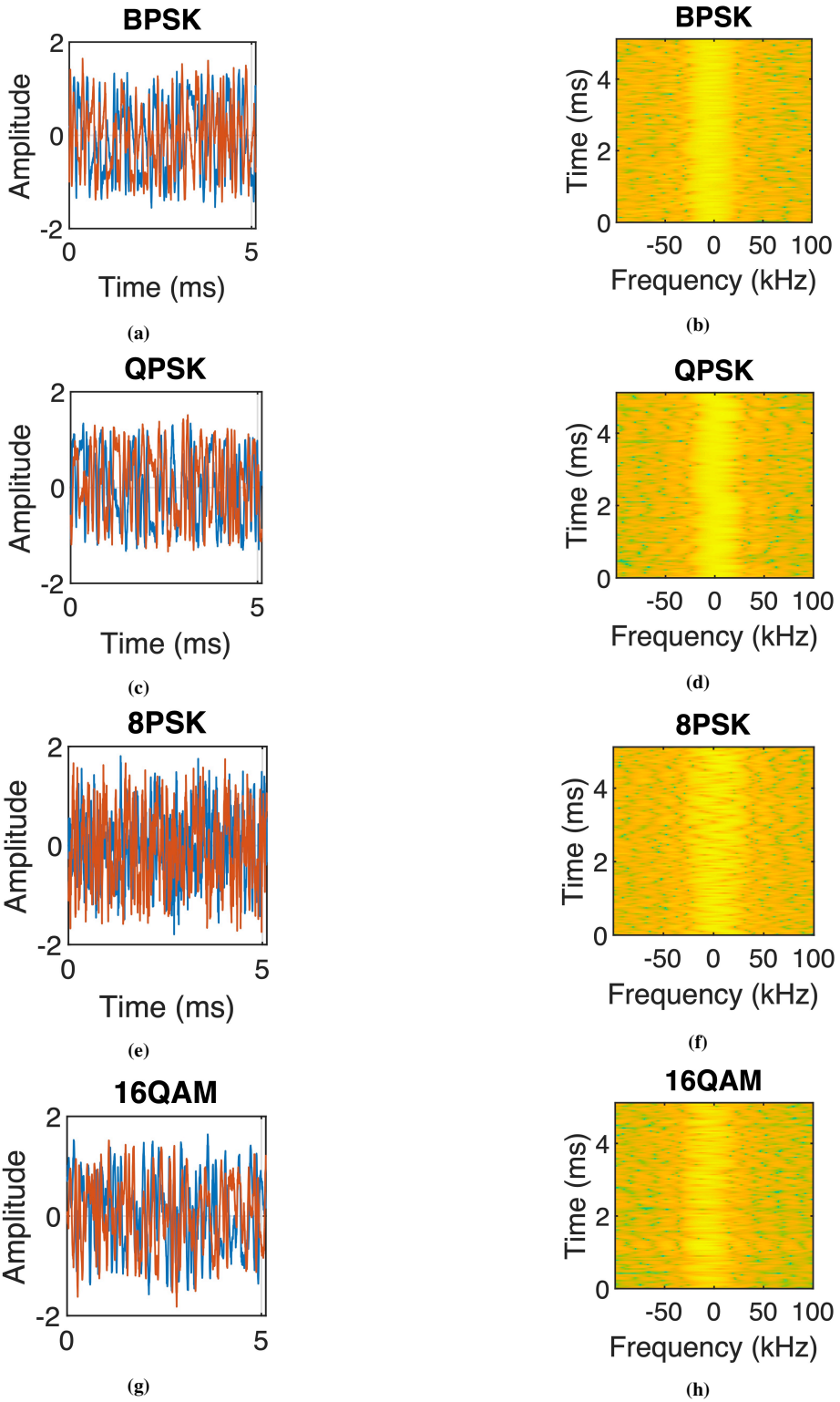

Figure 3.3: Time domain representations of BPSK, QPSK, 8PSK and 16QAM, and their corresponding spectrograms

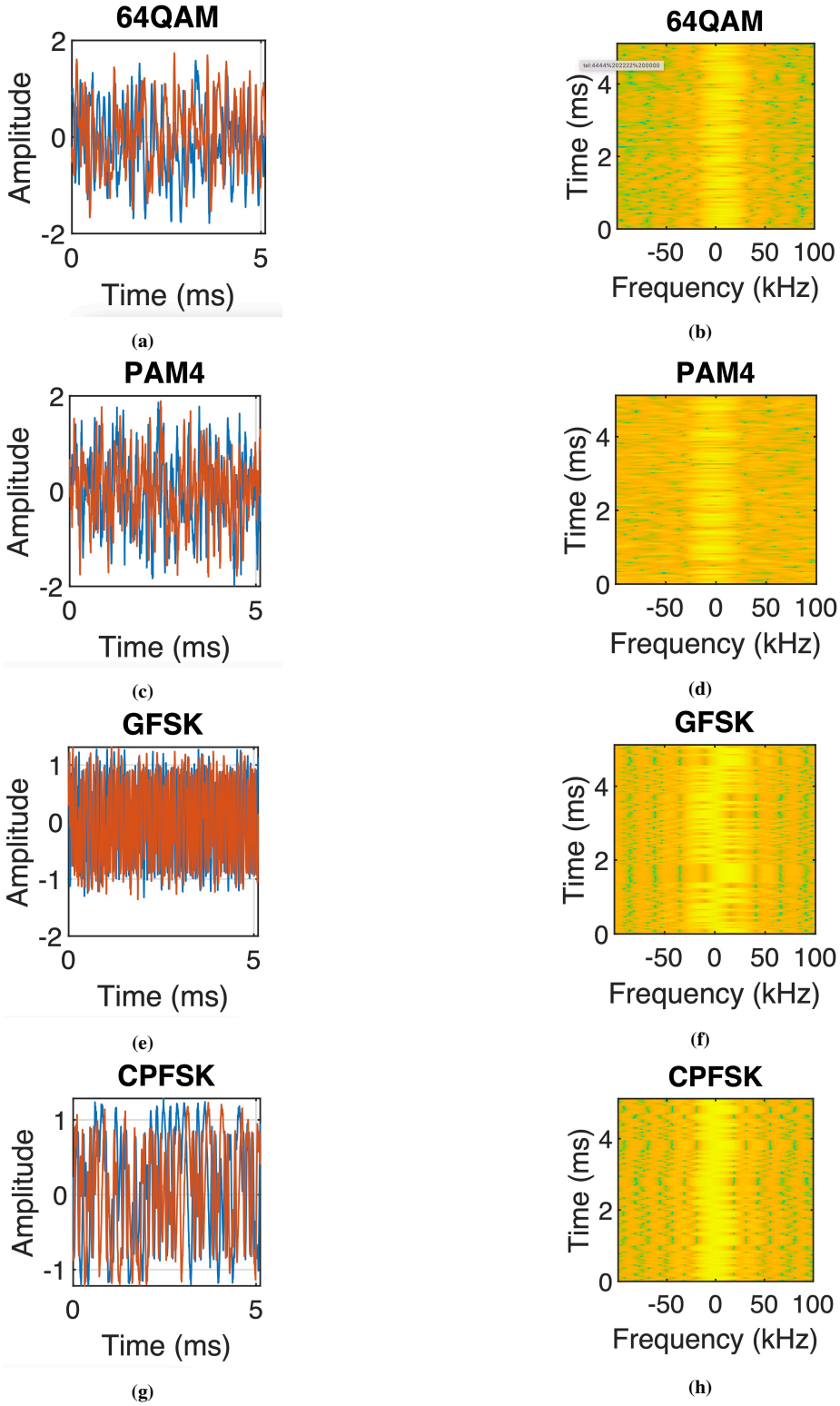

Figure 3.4: (a) Time domain representations of 64-QAM, 4-PAM, GFSK and CPFSK, and their corresponding spectrograms

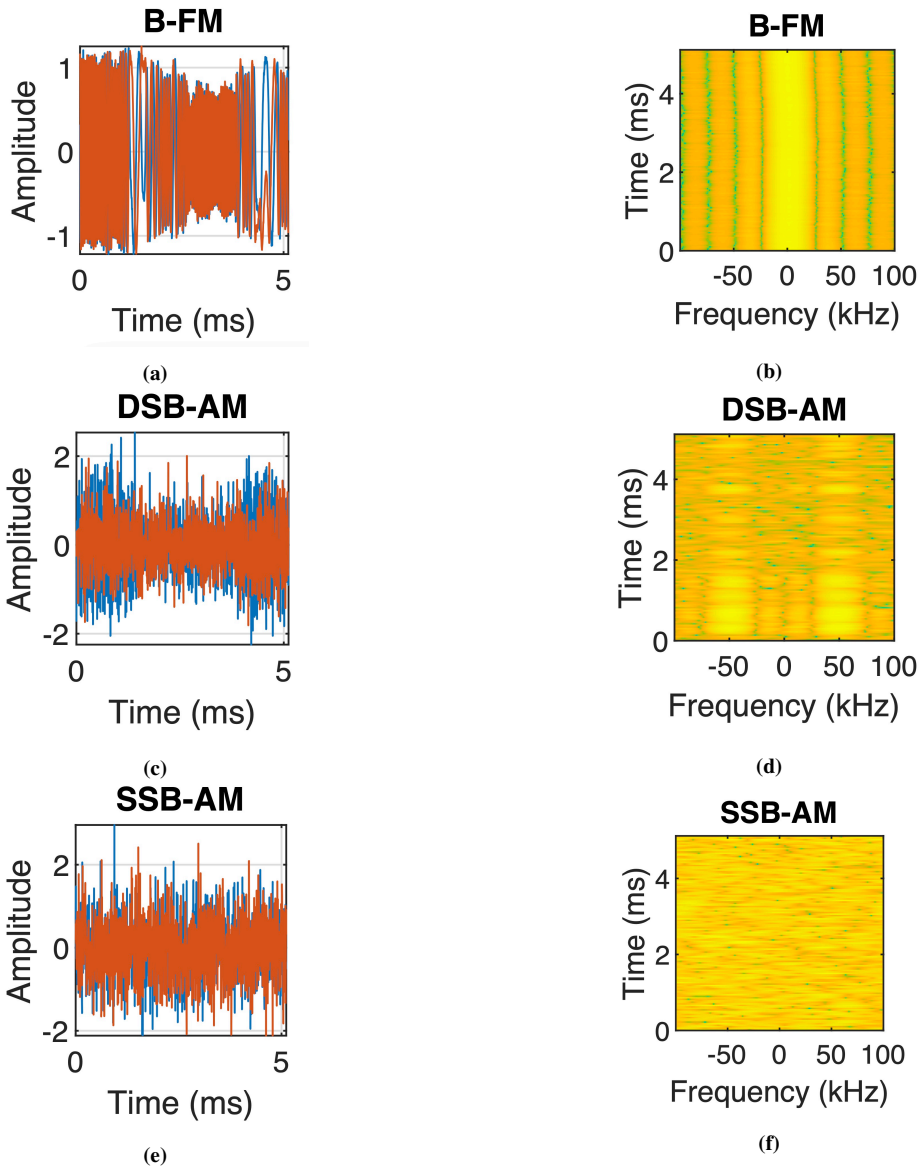

Figure 3.5: (a) Time domain representations of B-FM, DSB-AM and SSB-AM, and their corresponding spectrograms

3.2 Training of the CNN

The CNN consists of six convolution layers and one fully connected layer where each convolution layer except the last layer is followed by a batch normalization layer, rectified linear unit (ReLU) activation layer, and max pooling layer. In the last convolution layer, the max pooling layer is replaced with an average pooling layer. The output layer has softmax activation [2].

Next, the stochastic gradient descent with momentum (SGDM) solver has been configured using "TrainingOptionsSGDM" with a mini-batch size of 256. The maximum number of epochs

is set to 12 which is sufficient to provide stability into the CNN network. By default, the "ExecutionEnvironment" property is set to 'auto', where the "trainNetwork" function uses a GPU if one is available or uses the CPU, if not. The initial learning rate is set to 2×10^{-2} and for every 9 epochs, the learning rate decreases by a factor of 10. The network on CPU takes approximately 19 minutes to train as shown in Fig. 3.6. As the plot of the training progress shows, the network converges in about 12 epochs and the accuracy is 92.88% as shown in Fig. 3.9.

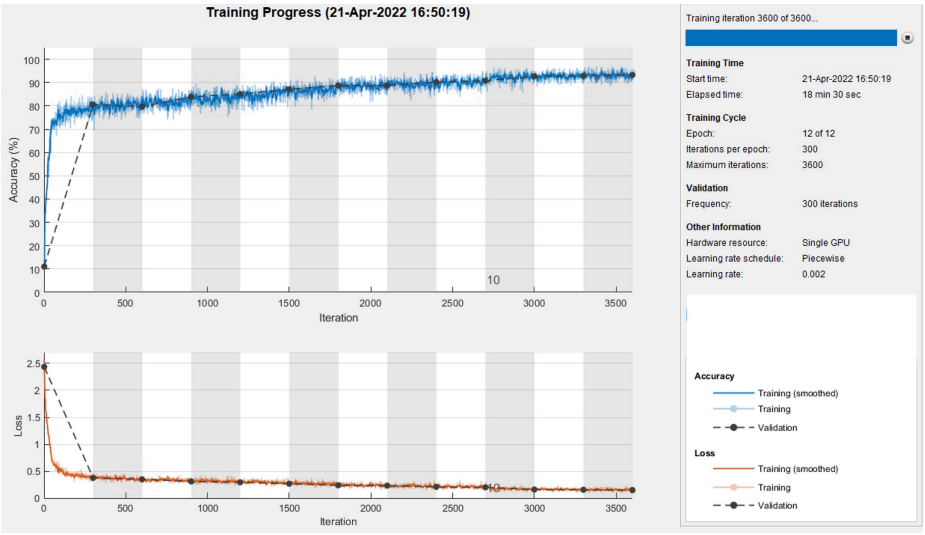

Figure 3.6: Training of the CNN

The confusion matrix is provided for the training data in Fig. 3.7, which is not required but it's been included to watch the behaviour of the CNN network when the training process completes.

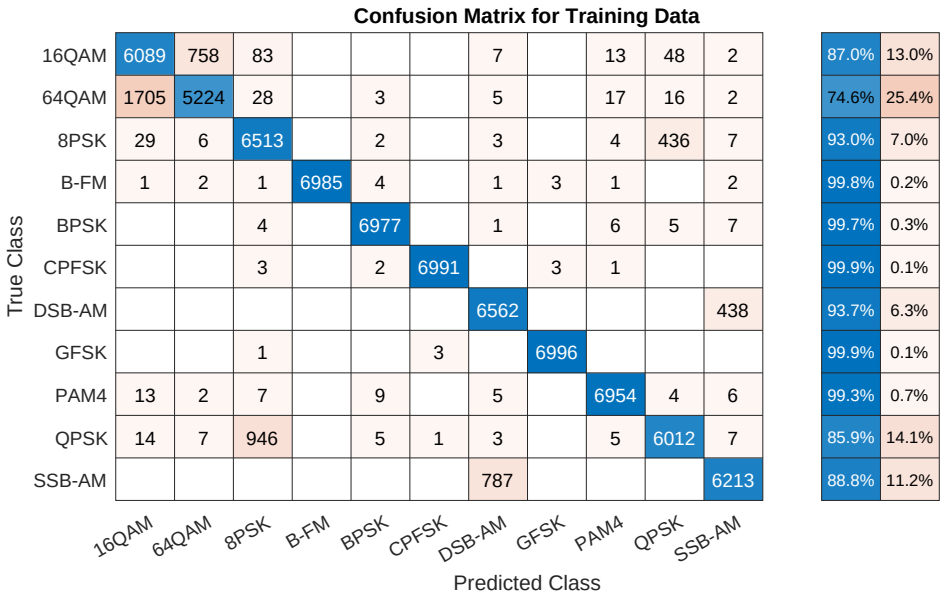

Figure 3.7: Confusion matrix for training data

3.3 Testing of the CNN

The confusion matrix has been plotted for the test frames of the signal waveforms. As the matrix shows, the network confuses 16-QAM and 64-QAM frames. This problem is expected since each frame carries only 128 symbols and 16-QAM is a subset of 64-QAM. The network also confuses QPSK and 8-PSK frames, since the constellations of these modulation types look similar once phase-rotated due to the fading channel and frequency offset. The CNN network also confuses between DSB-AM and SSB-AM which is based on the fact that DSB-SC has the same BW as AM and SSB has half of the bandwidth as AM where AM bandwidth is approximately twice the highest frequency that has energy in the modulating waveform.

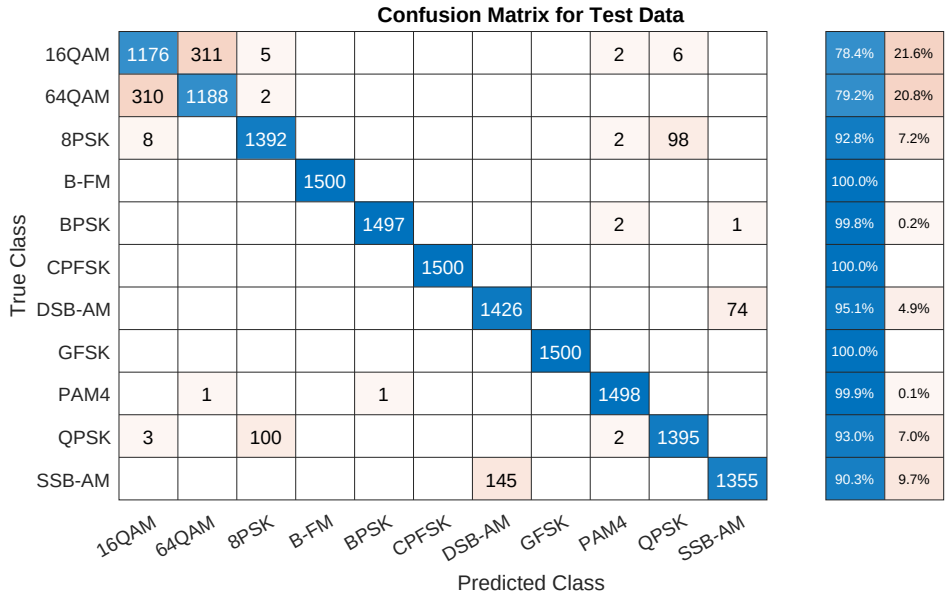

Figure 3.8: Confusion matrix for test data

Test accuracy: 93.5%.

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 2: Completed in 4 min 49 sec
- Pass 2 of 2: Completed in 5 min 32 sec
Evaluation completed in 10 min 22 sec
Training accuracy: 92.8779%
00:20:12 - Training the network
00:39:33 - Classifying test frames
Test accuracy: 93.497%
IdleTimeout has been reached.
Parallel pool using the 'local' profile is shutting down.
```

Figure 3.9: Training and testing accuracy

4 Discussion

The CNN based DL network is tested against the 15% data that has been reserved from the original dataset. To test the performance of the trained network with over-the-air signals using the “helperModClassSDRTest” function, we must have dedicated SDRs for transmission and reception. We must install communications toolbox support package for analog radio devices which is unavailable in our lab computers.

5 Conclusion

This project is used a CNN based DL model to classify modulations. Several waveforms have been generated and used for training and testing purposes. As compared to conventional and ML-based modulation classification algorithms, CNN based DL approach avoids manual feature selections and provide a higher classification accuracy.The CNN based DL network is successful in classifying modulations by 93.5% of accuracy.

References

[1] S. Peng, H. Jiang, H. Wang, H. Alwageed, Y. Zhou, M. M. Sebdani, and Y.-D. Yao, "Modulation classification based on signal constellation diagrams and deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 3, pp. 718–727, 2018.

[2] MATLAB. "Modulation classification with deep learning," *The MathWorks, Inc.* [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/modulation-classification-with-deep-learning.html

[3] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[4] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *International conference on engineering applications of neural networks*. Springer, 2016, pp. 213–226.

[5] X. Liu, D. Yang, and A. El Gamal, "Deep neural network architectures for modulation classification," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 915–919.

Appendix

MATLAB Code

The MATLAB code [2] used to generate results in the project is given below:

```
modulationTypes = categorical(["BPSK", "QPSK", "8PSK", ...
  "16QAM", "64QAM", "PAM4", "GFSK", "CPFSK", ...
  "B-FM", "DSB-AM", "SSB-AM"]);

load trainedModulationClassificationNetwork
trainedNet

% Set the random number generator to a known state to be able to regenerate
% the same frames every time the simulation is run
rng(123456)
% Random bits
d = randi([0 3], 1024, 1);
% PAM4 modulation
syms = pammod(d,4);
% Square-root raised cosine filter
filterCoeffs = rcosdesign(0.35,4,8);
tx = filter(filterCoeffs,1,upsample(syms,8));

% Channel
SNR = 25;
maxOffset = 4;
fc = 900e6;
fs = 200e3;
multipathChannel = comm.RicianChannel(...
  'SampleRate', fs, ...
  'PathDelays', [0 1.6 3.2] / 200e3, ...
  'AveragePathGains', [0 -3 -9], ...
  'KFactor', 4, ...
  'MaximumDopplerShift', 3);

frequencyShifter = comm.PhaseFrequencyOffset(...
  'SampleRate', fs);

% Apply an independent multipath channel
reset(multipathChannel)
outMultipathChan = multipathChannel(tx);

% Determine clock offset factor
```

```
clockOffset = (rand() * 2*maxOffset) - maxOffset;
C = 1 + clockOffset / 1e6;

% Add frequency offset
frequencyShifter.FrequencyOffset = -(C-1)*fc;
outFreqShifter = frequencyShifter(outMultipathChan);

% Add sampling time drift
t = (0:length(tx)-1)' / fs;
newFs = fs * C;
tp = (0:length(tx)-1)' / newFs;
outTimeDrift = interp1(t, outFreqShifter, tp);

% Add noise
rx = awgn(outTimeDrift,SNR,0);

% Frame generation for classification
unknownFrames = helperModClassGetNNFrames(rx);

% Classification
[prediction1,score1] = classify(trainedNet,unknownFrames);

helperModClassPlotScores(score1,modulationTypes)

trainNow = true
if trainNow == true
  numFramesPerModType = 10000;
else
  numFramesPerModType = 200;
end
percentTrainingSamples = 70;
percentValidationSamples = 15;
percentTestSamples = 15;

sps = 8;                % Samples per symbol
spf = 1024;             % Samples per frame
symbolsPerFrame = spf / sps;
fs = 200e3;             % Sample rate
fc = [900e6 100e6];     % Center frequencies

maxDeltaOff = 5;
deltaOff = (rand()*2*maxDeltaOff) - maxDeltaOff;
C = 1 + (deltaOff/1e6);

```

```
channel = helperModClassTestChannel(...
  'SampleRate', fs, ...
  'SNR', SNR, ...
  'PathDelays', [0 1.6 3.2] / fs, ...
  'AveragePathGains', [0 -3 -9], ...
  'KFactor', 4, ...
  'MaximumDopplerShift', 3, ...
  'MaximumClockOffset', 4, ...
  'CenterFrequency', 900e6)

chInfo = info(channel)

% Set the random number generator to a known state to be able to regenerate
% the same frames every time the simulation is run
rng(1235)
tic

numModulationTypes = length(modulationTypes);

channelInfo = info(channel);
transDelay = 50;
dataDirectory = fullfile(tempdir,"ModClassDataFiles");
disp("Data file directory is " + dataDirectory)

fileNameRoot = "frame";

% Check if data files exist
dataFilesExist = false;
if exist(dataDirectory,'dir')
  files = dir(fullfile(dataDirectory,sprintf("%s*",fileNameRoot)));
  if length(files) == numModulationTypes*numFramesPerModType
    dataFilesExist = true;
  end
end

if ~dataFilesExist
  disp("Generating data and saving in data files...")
  [success,msg,msgID] = mkdir(dataDirectory);
  if ~success
    error(msgID,msg)
  end
  for modType = 1:numModulationTypes
    elapsedTime = seconds(toc);
    elapsedTime.Format = 'hh:mm:ss';
```

```
127       fprintf('%s - Generating %s frames\n', ...
128         elapsedTime, modulationTypes(modType))
129
130       label = modulationTypes(modType);
131       numSymbols = (numFramesPerModType / sps);
132       dataSrc = helperModClassGetSource(modulationTypes(modType), sps, 2*spf, fs);
133       modulator = helperModClassGetModulator(modulationTypes(modType), sps, fs);
134       if contains(char(modulationTypes(modType)), {'B-FM','DSB-AM','SSB-AM'})
135         % Analog modulation types use a center frequency of 100 MHz
136         channel.CenterFrequency = 100e6;
137       else
138         % Digital modulation types use a center frequency of 900 MHz
139         channel.CenterFrequency = 900e6;
140       end
141
142       for p=1:numFramesPerModType
143         % Generate random data
144         x = dataSrc();
145
146         % Modulate
147         y = modulator(x);
148
149         % Pass through independent channels
150         rxSamples = channel(y);
151
152         % Remove transients from the beginning, trim to size, and normalize
153         frame = helperModClassFrameGenerator(rxSamples, spf, spf, transDelay, sps);
154
155         % Save data file
156         fileName = fullfile(dataDirectory,...
157           sprintf("%s%s%03d",fileNameRoot,modulationTypes(modType),p));
158         save(fileName,"frame","label")
159       end
160     end
161   else
162     disp("Data files exist. Skip data generation.")
163   end
164
165   % Plot the amplitude of the real and imaginary parts of the example frames
166   % against the sample number
167   figure(2)
168   helperModClassPlotTimeDomain(dataDirectory,modulationTypes,fs)
169
170   figure(3)
```

```
% Plot the spectrogram of the example frames
helperModClassPlotSpectrogram(dataDirectory,modulationTypes,fs,sps)

% Create a datastore
frameDS = signalDatastore(dataDirectory,'SignalVariableNames',["frame","label"]);

frameDSTrans = transform(frameDS,@helperModClassIQAsPages);

splitPercentages = [percentTrainingSamples,percentValidationSamples,percentTestSamples];
[trainDSTrans,validDSTrans,testDSTrans] = helperModClassSplitData(frameDSTrans,splitPercentages)
    ;

% Read the training and validation frames into the memory
pctExists = parallelComputingLicenseExists();
trainFrames = transform(trainDSTrans, @helperModClassReadFrame);
rxTrainFrames = readall(trainFrames);%,"UseParallel",pctExists);
rxTrainFrames = cat(4, rxTrainFrames{:});
validFrames = transform(validDSTrans, @helperModClassReadFrame);
rxValidFrames = readall(validFrames);%,"UseParallel",pctExists);
rxValidFrames = cat(4, rxValidFrames{:});

% Read the training and validation labels into the memory
trainLabels = transform(trainDSTrans, @helperModClassReadLabel);
rxTrainLabels = readall(trainLabels);%,"UseParallel",pctExists);
validLabels = transform(validDSTrans, @helperModClassReadLabel);
rxValidLabels = readall(validLabels);%,"UseParallel",pctExists);


rxTrainPred = classify(trainedNet,rxTrainFrames);
trainAccuracy = mean(rxTrainPred == rxTrainLabels);
disp("Training accuracy: " + trainAccuracy*100 + "%")


figure(4)
cm = confusionchart(rxTrainLabels, rxTrainPred);
cm.Title = 'Confusion Matrix for Training Data';
cm.RowSummary = 'row-normalized';
cm.Parent.Position = [cm.Parent.Position(1:2) 740 424];


modClassNet = helperModClassCNN(modulationTypes,sps,spf);

maxEpochs = 12;
miniBatchSize = 256;
```

```
options = helperModClassTrainingOptions(maxEpochs,miniBatchSize,...
  numel(rxTrainLabels),rxValidFrames,rxValidLabels);

if trainNow == true
  elapsedTime = seconds(toc);
  elapsedTime.Format = 'hh:mm:ss';
  fprintf('%s - Training the network\n', elapsedTime)
  trainedNet = trainNetwork(rxTrainFrames,rxTrainLabels,modClassNet,options);
else
  load trainedModulationClassificationNetwork
end

elapsedTime = seconds(toc);
elapsedTime.Format = 'hh:mm:ss';
fprintf('%s - Classifying test frames\n', elapsedTime)
% Read the test frames into the memory
testFrames = transform(testDSTrans, @helperModClassReadFrame);
rxTestFrames = readall(testFrames);%,"UseParallel",pctExists);
rxTestFrames = cat(4, rxTestFrames{:});

% Read the test labels into the memory
testLabels = transform(testDSTrans, @helperModClassReadLabel);
rxTestLabels = readall(testLabels);%,"UseParallel",pctExists);

rxTestPred = classify(trainedNet,rxTestFrames);
testAccuracy = mean(rxTestPred == rxTestLabels);
disp("Test accuracy: " + testAccuracy*100 + "%")

figure(5)
cm = confusionchart(rxTestLabels, rxTestPred);
cm.Title = 'Confusion Matrix for Test Data';
cm.RowSummary = 'row-normalized';
cm.Parent.Position = [cm.Parent.Position(1:2) 740 424];
```