Internship Project Report and Documentation: RF Signal Classification using Deep Learning (06 July 2020 to 21 Aug 2020)

Yoke Kai Wen

August 19, 2020

Contents

1 Intent

1.1 Motivation

The aim of this project is to classify RF signals by their modulation type using deep learning models. Modulation recognition is an important component of spectrum management and an essential first step towards realising the vision of cognitive radios to better ensure access to radio frequencies required for military and civilian communications. Traditional methods of modulation recognition involve feature extraction and design by domain experts which takes a lot of time and effort, whereas deep learning methods have been shown to be capable of extracting features automatically from raw I/Q data and surpassing the performance of traditional methods. Therefore, in this project, I focus on using deep learning methods for modulation classification.

1.2 Project organisation and main questions to answer

My project can be split into two parts. In the first part, I work with the 2016.10A radioML dataset [1] (described in the next section), and try to reproduce previous work on modulation classification described in Table 1. The main questions I want to answer in this part are: *What is the best neural architecture and input feature type for the task of modulation classification, and why do they work?* In the second part, I create another RF dataset that simulates more realistic and harsher channel conditions than the radioML dataset, and test the models that performed well in the previous part on this new and more difficult dataset. The main questions I want to answer here are: *How well can DL models adapt to RF signals from different channel conditions, and do the conclusions drawn from the first part apply to this dataset as well?*

2 Literature Studies

Several neural architectures (employing various combinations of convolutional and recurrent layers) and RF signal data representations (I/Q time-series, amplitude-phase time-series and constellation diagrams) have been proposed for the task of modulation recognition. In this section, I highlight works that I have heavily relied on, as summarised in Table 1.

Architecture	**Input feature**	**Dataset** [1]	**Source**
Basic CNN	2x128 I/Q time-series	2016.04C,	[2]
		2016.10A radioML	[3]
	2x128 amp-phase time-series	2016.10A radioML	[4]
Inception	2x128 I/Q time-series	2016.10A radioML	[3]
ResNet	2x128 I/Q time-series	2016.10A radioML	[3]
	2x1024 I/Q time-series	2018.01A radioML	[5]
CLDNN	2x128 I/Q time-series	2016.10A radioML	[3]
LSTM	2x128 amp-phase time-series	2016.10A radioML	[6]
AlexNet	Grayscale constellation image (227x227x1)	unknown	[7]
	Coloured constellation image (128x128x3), (227x227x3)	unknown	[8] [9]

Table 1: Table summarizing neural architectures and signal feature types focused on in this project

2.1 DeepSig and radioML datasets

DeepSig Inc. (https://www.deepsig.ai/) and its researchers are the pioneers in the field of RF signal processing using machine learning, having written many papers on this topic [2] [3] [5] and published several RF datasets [1]. The three RF radioML datasets are available here: https://www.deepsig.ai/datasets. The 2016.04C and 2016.10A datasets contain 11 types of modulation schemes ranging across SNRs from -20dB to 18dB, with each data sample being an I/Q time-series with 128 time-steps, represented as a 2x128 array. Realistic channel imperfections such as moderate LO drift, light multipath fading and AWGN are included in the datasets (generated by GNU Radio) and the detailed process for dataset generation can be found in O'Shea et al.'s paper [1]. There are 8 digital modulation classes `(BPSK, QPSK, 8PSK, PAM4, QAM16, QAM64, GFSK, CPFSK)` and 3 analogue modulation classes `(WBFM, AM-DSB, AM-SSB)` (see Figure 1). The work from DeepSig forms the basis for this project.

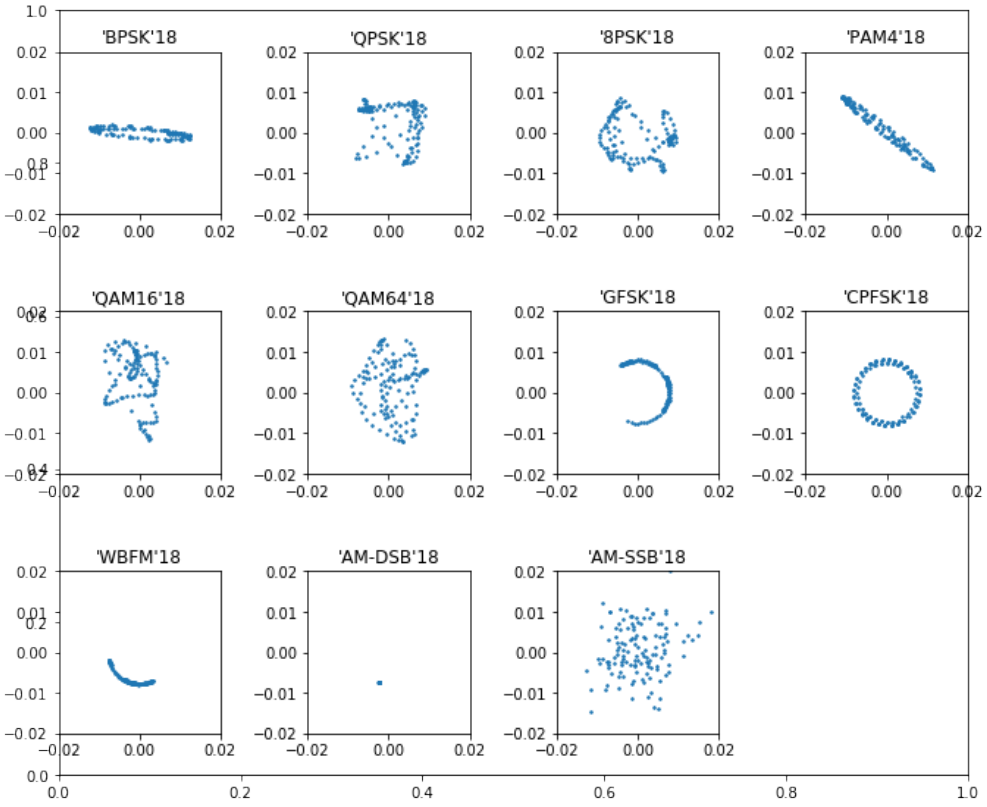

Figure 1: Constellation diagrams of 11 modulation schemes at SNR=18dB of the 2016.10A radioML dataset [1]

The 2016 radioML datasets have been used by other papers as a benchmark and therefore I also use them in the project. I explore all three radioML datasets in more detail in my `dataset_visualisation.ipynb` notebook, and have found flaws in the 2016.04C and 2018.01A datasets that make them unusable, hence only the 2016.10A radioML dataset is used in this project. The description of each dataset is presented in Table 2.

	2016.04C	2016.10A	2018.01A
num_classes	11		24
Class types	Digital: BPSK, QPSK, 8PSK, QAM16, QAM64, CPFSK, GFSK, PAM4 Analogue: WBFM, AM-SSB, AM-DSB		Digital: OOK, 4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 32PSK, 16APSK, 32APSK, 64APSK, 128APSK, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM, GMSK, OQPSK Analogue: AM-SSB-WC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM
SNR range	-20dB to 18dB		
num_samps per (mod, snr)	705	1000	>4000
samp format	2x128		2x1024
Problems (analogue modulation)	Almost impossible to differentiate between analogue modulations because of pauses in the voice recording of the source dataset that analogue modulations were generated from.		
Other problems	very noisy, not normalised	-	Class labels are wrong

Table 2: Table summarizing descriptions of radioML datasets

2.2 Using CNN to extract features from I/Q time-series

O'Shea et al. [2] [3] [5] propose several variations of Convolutional Neural Networks for the task of modulation classification. 1D convolutional layers have proven to be helpful for time-series analysis in tasks such as human activity recognition and financial time-series forecasting, so it makes sense that 1D convolutional layers would also be helpful for extracting features from the I/Q time-series. In [2] and [3], O'Shea et al. suggest that convolutional filters could be analogous to matched filters at the receiver which help to maximise the SNR of the received signal at specific points. I attempt to visualise the filters and feature maps of the CNN layers but have not managed to figure out if they do actually work like matched filters. Nonetheless, the CNN architectures are quite successful in modulation classification, and below I describe four variations of CNN from [2] [3] [5] that have been inspired by architectures applied in computer vision, such as ResNet and Inception.

2.2.1 Basic CNN [2]

In 2016, O'Shea et al. [2] designed a basic CNN with two convolutional layers followed by two dense layers (architecture shown in Figure 2) to prove the point that even a simple neural architecture like this outperforms traditional expert feature based methods, which consequently led to more research in using deep learning methods for modulation classification. The choice of filter sizes (1x3, 2x3) and filter numbers in each layer were determined through trial and error.

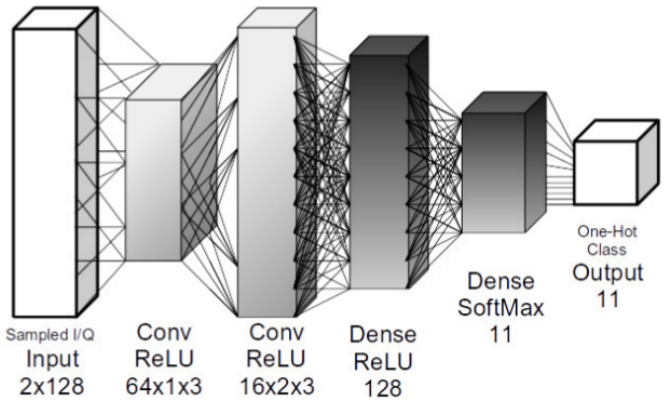

Figure 2: Architecture of basic CNN with two convolutional layers and two dense layers [2]

2.2.2 CNN based on Inception [3]

The inception module contains filters of varying sizes in each layer, allowing processing of spatial information at various scales, which are then aggregated when the filter outputs are concatenated together. O'Shea et al. [3] adopts the same idea, but varies the filter sizes used in the inception module (see Figure 3). In [3], they mention that by expert knowledge they hypothesize 8-tap filters to be the most optimal, but I was still not very sure why and I was thinking it could be related to rule-of-thumbs for determining FIR filter order based on desired filter attributes such as bandwidth, roll-off, attenuation. [3] use the 2016.10A dataset and find the Inception modules not significantly helpful in improving classification accuracy.

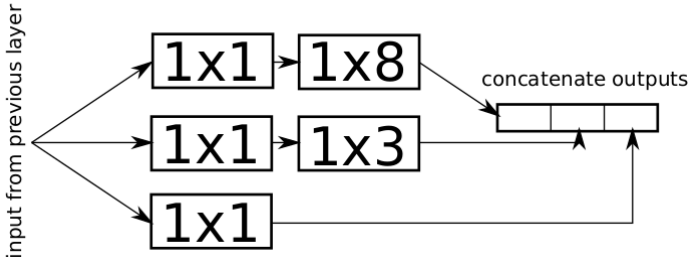

Figure 3: Architecture of 1D inception module [3]

2.2.3 CNN based on ResNet [5]

Residual networks have done very well in computer vision tasks because they allow networks to go deep without facing the problem of vanishing gradients by having skip connections between layers, which also allows features to operate at multiple scales and depths throughout the network. This idea was adopted in [3] and [5] with the residual unit and residual stack module shown in Figure 4, where each convolutional layer contains 1x3 filters. The former paper did not

manage to achieve exceptional results with ResNet modules and thought that it was because CNNs were limited in how much they could learn from radio signals so it did not matter how deep the network can go. A year later, the same researchers [5] produced and used the 2018.01A radioML dataset and showed that its modified ResNet architecture outperformed previous neural architectures. These modifications include the activation functions, dropout layers and initialisation functions, but I am not sure of the exact details.

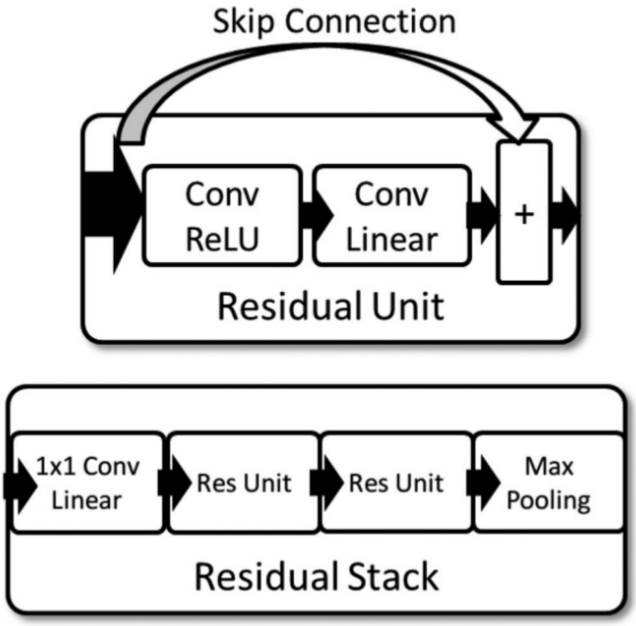

Figure 4: Architecture of 1D Residual Stack module [5]

2.2.4 CLDNN: CNN followed by LSTM [3]

Convolutional Long short-term Deep Neural Networks (CLDNN) have been used extensively for voice-processing research. Unlike the previous CNN architectures, CLDNN adds a recurrent unit (LSTM) after the convolutional layers to extract temporal features. [3] also adds a skip connection after the first convolutional layer to the concatenated layer so that the LSTM can process the waveform in a rawer state (see Figure 5).

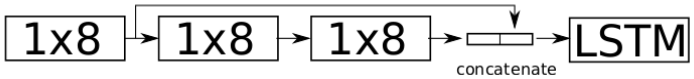

Figure 5: Architecture of CLDNN [3]

2.3 Representing I/Q time-series as amplitude-phase time-series

Some papers [4] [6] used amplitude-phase time-series as input to their deep learning models rather than using I/Q time-series directly, and reported better results. [4] also tried using the frequency spectra as inputs but this performed badly so I will not discuss it here.

2.3.1 Basic CNN with amplitude-phase time-series as input [4]

[4] used a similar CNN architecture as in [2] (Figure 2) and tried training it with both I/Q and amplitude-phase time-series from the 2016.10A radioML dataset. It was found that training with amplitude-phase time series improves classification accuracy only at high SNR, while training with I/Q data resulted in higher accuracy at low SNR, but it was not clear why this was so.

2.3.2 LSTM with amplitude-phase time-series as input [6]

Recurrent Neural Networks (RNN) are commonly used for learning persistent features in time-series data, and the LSTM (Long Short Term Memory network) is a type of RNN that is able to retain a longer history and thus capable of learning longer term patterns. [6] showed that a simple network consisting of two LSTM layers and two fully connected layers (see architecture in Figure 6), trained with amplitude-phase time-series data, was able to achieve very good classification accuracy on the 2016.10A radioML dataset. The number of layers and number of cells were determined experimentally.

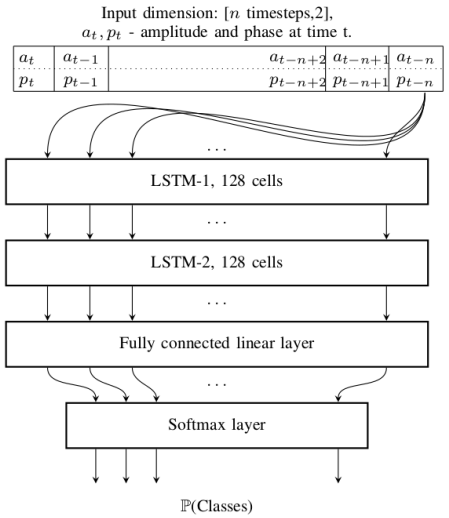

Figure 6: Architecture of LSTM network comprising 2 LSTM layers and 2 dense layers [6]

2.4 Representing I/Q time-series as constellation images

Several papers [7] [8] [9] proposed to transform the I/Q raw data into constellation images to be fed into a CNN image classifier. Constellation diagrams are widely used as a 2-D representation of a modulated signal by mapping signal samples into scatter points on the complex plane. [7] simply transforms the I/Q data into a grayscale image, while [9] and [8] propose a data conversion method by mapping point density to a colours from a colour scale, with an example shown in Figure 7. All three papers used CNN models based on variants of AlexNet for constellation image classification. [8] compared the constellation model with time-series IQ models on two specific QAM modulations, and found that the constellation model was able to achieve 100% accuracy across a range of SNRs and outperforms the IQ models significantly. [9] compared [7]'s single-channel constellation images with its RGB constellation images and found that training with the coloured version resulted in better accuracy. These papers did not make the RF datasets

they used available so it is difficult to benchmark them against other classification models, and it was also not clear how many data points were used per image.

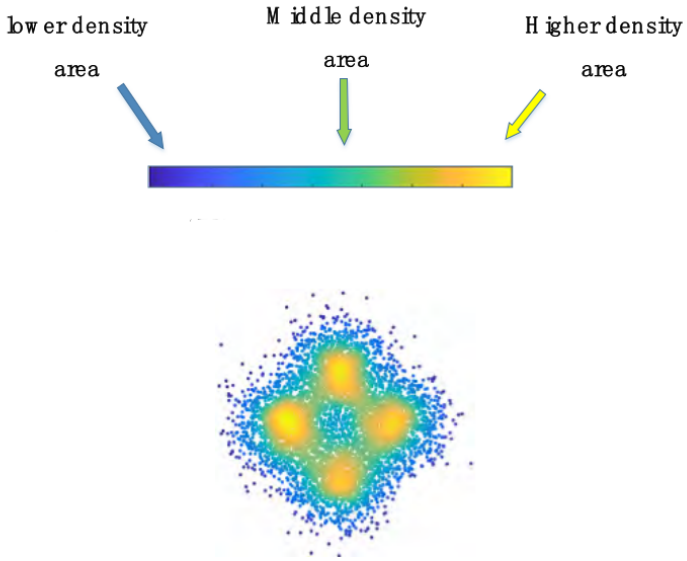

Figure 7: Data conversion from I/Q time-series to constellation image coloured by density, with an example QPSK signal [9]

3 General implementation methodology

In this section, I describe in general the implementation process such as computing resources, training parameters and performance metrics used.

3.1 Google Colaboratory

All code was written and executed on Google Colaboratory using the Keras library. Google Colab provides for free a 12GB NVIDIA Tesla K80 GPU, Intel(R) Xeon(R) CPU @ 2.30GHz and 12GB RAM. The GPU was sufficient - each epoch for time-series model takes less than 30s and for the constellation model around a minute. However, the 12GB RAM was quite limiting because loading the RF datasets (from my Google Drive) into Colab consumes a lot of memory, especially when I convert the time-series data into constellation images, so care has to be taken to delete variables when not needed. Also, Google Drive provides only 15GB free storage, while each RF dataset can take up a few GB, so I had to delete and re-upload files onto Google Drive frequently.

3.2 Training parameters

3.2.1 Dataset used

Only the 8 digital modulation classes from the 2016.10A radioML dataset [1] are used for training and evaluation, while the analogue modulations are removed because it is impossible to distinguish between them anyway due to pauses in the audio source they originated from. Although this makes it difficult to compare my implementation's results with those from previous papers

that used all 11 classes in the radioML dataset, I think the tradeoff for a better reflection of model accuracy is worth it.

3.2.2 Train-validation-test splits

67% of the radioML dataset was randomly selected for training, 13% for validation and 20% for testing. I initially considered doing 5-Fold validation for a fairer evaluation but it was taking too long and consuming too much memory and so I gave up. Since I had 1000 samples for each (modulation, SNR), after random selection and doing train-val-test splits, I have roughly 670 for training, 130 for validation and 200 for testing for each (modulation, SNR).

3.2.3 Batch size

Batch sizes of 1024 were used for time-series data inputs and batch sizes of 64 for constellation image data inputs due to memory constraints.

3.2.4 Training epochs

I set a maximum training epochs of 100, and activated Early Stopping with a patience of 10 epochs for time-series and 4 epochs for constellation images, meaning that training stops when the validation loss has not improved for n consecutive epochs. Usually for time-series training, the model overfits before 70 epochs while for constellation image training, the model overfits before 40 epochs.

3.2.5 Optimisers and other hyperparameters

In this project, I did not spend a lot of time on optimising hyperparameters, and most of the hyperparameters were default values in Keras except when otherwise specified in the papers. I used the Adam optimiser for all models as it is known to be relatively easy to configure because the default configuration parameters do well on most problems. The time-series model uses a learning rate of 10^{-3} while the constellation model uses a learning rate of 10^{-4} because 10^{-3} did not work for the constellation model.

3.3 Performance metrics

I use the overall classification accuracy ($\frac{number\_classified\_correctly}{total}$) across all modulation classes and SNRs as the main metric. However, this is not very helpful because performance varies vastly across the SNR range, so I also calculate three more metrics for each SNR range: low SNR (-20dB to -12dB), medium SNR (-10dB to 4dB) and high SNR (6dB to 18dB) classification accuracy.
I also plot confusion matrices at different SNRs to gain more insight into which modulation classes usually get confused and why.

4 Part 1: Reproduction of previous architectures on radioML dataset

Part 1 focuses on implementing promising model architectures and feature types in Table 1, comparing them and drawing insights into the best architecture and feature type for modulation classification. Then, I try to incorporate these insights into a combined model.

4.1 Methodology

I created three Colab notebooks for Part 1: (1) the `time-series_classification.ipynb` notebook for training different architectures (see Figure 1) on time-series features and analysing the effects of training with amplitude-phase time-series instead of I/Q time-series ; (2) the `constellation_classification.ipynb` notebook for experimenting with different representations of constellation images and their effects on classification accuracy; (3) the `model_evaluation_radioml.ipynb` notebook for evaluating and comparing the best performing models, highlighting insights and presenting a combined model that integrates these insights. All of the implementation details can be found in the notebooks, so below I only explain key things to note.

4.1.1 Conversion of I/Q data to amplitude-phase data

Note that although the I and Q components were already normalised, after obtaining the amplitude and phase data from the I, Q components via the standard formula, it is still necessary to normalise them, otherwise the model will perform poorly. The normalisation process (see code below) follows [6].

```
import numpy as np
def iq2ampphase(inphase, quad):
    amplitude = np.sqrt(np.square(inphase) + np.square(quad))
    amp_norm = np.linalg.norm(amplitude) #L2 norm
    amplitude = amplitude/amp_norm #normalise
    phase = np.arctan(np.divide(quad, inphase))
    phase = 2.*(phase - np.min(phase))/np.ptp(phase)-1 #rescale phase to range
    [-1, 1]
    return amplitude, phase
```

4.1.2 Conversion of I/Q data to constellation images

Conversion to constellation image involves dividing the I and Q axes (in the desired region of the complex plane) into *nbins* and counting the number of points that fall into each bin, and then normalising the counts to a range of -1 to 1. A colormap from matplotlib is then applied, with `cmap='gray'` for single channel and `cmap='hot'` for three channels. I explored constellations with $nbins = 32, 48, 64, 96$ and single, triple channels.

```
xyrange = 0.02 #0.05 for matlab data, this sets the region in complex plane to be
    captured
counts, xedges, yedges = np.histogram2d(inphase, quadrature, bins=b,range = [[-
    xyrange, xyrange], [-xyrange, xyrange]])
def arr2img(arr, chnum):
    norm = plt.Normalize(vmin=arr.min(), vmax=arr.max())
    if chnum == 1:
        cmap = plt.cm.gray
        image4d = cmap(norm(arr)) #RGBA
        img = image4d[:,:,0] #All RGBA channels identical
    elif chnum == 3:
        cmap = plt.cm.hot #or can choose any other colormap
        image4d = cmap(norm(arr)) #RGBA
        img = image4d[:,:,:3] #ignore A channel

    return img
```

Figure 8 shows the constellation images at different resolutions and colourations. Initially, I assumed that higher resolution and colour images would work the best, in accordance to [8] and [9],

especially when we have high order modulations like 64QAM. By human eye, it is clear that the coloured ones look more distinctive, so I assumed the CNN would work similarly. Ideally, I want to pick the lowest resolution that works accurately to save on memory consumption.

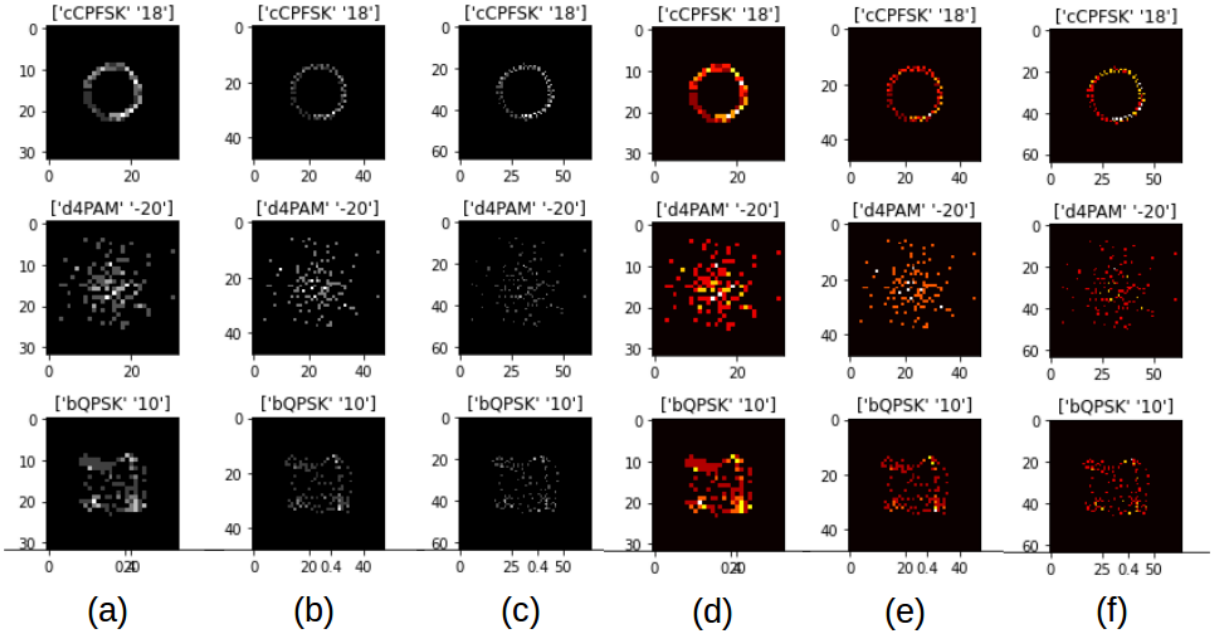

Figure 8: Examples of constellation images at different resolutions and colourations: (a) 32 bins, grayscale; (b) 48 bins, grayscale; (c) 64 bins, grayscale; (d) 32 bins, colour; (e) 48 bins, colour; (f) 64 bins, colour

4.2 Results

In this section, I present the evaluation results of the different models and different features. They were trained, validated and evaluated on the same train-val-test split, so all the test data for evaluation were unseen by all the models. For some models, I only trained with higher SNR data due to memory and speed issues. I also divided the SNRs into three groups: low (¡-10dB), medium (-10 to 5dB), high (¿5dB) for easier comparison and evaluation.

4.2.1 Performance of models trained on time-series inputs

I trained five types of architectures on time-series inputs: Basic CNN, Inception, ResNet, CLDNN and LSTM (for full model architectures, see my notebook), with each model trained separately on IQ and amplitude-phase (AP) data, except for Inception and LSTM. Inception performed so poorly on the raw IQ data that I did not train it with AP data; LSTM performed extremely poorly on IQ data so I did not continue is training. The classification accuracies over the entire SNR range is shown in Figure 9.

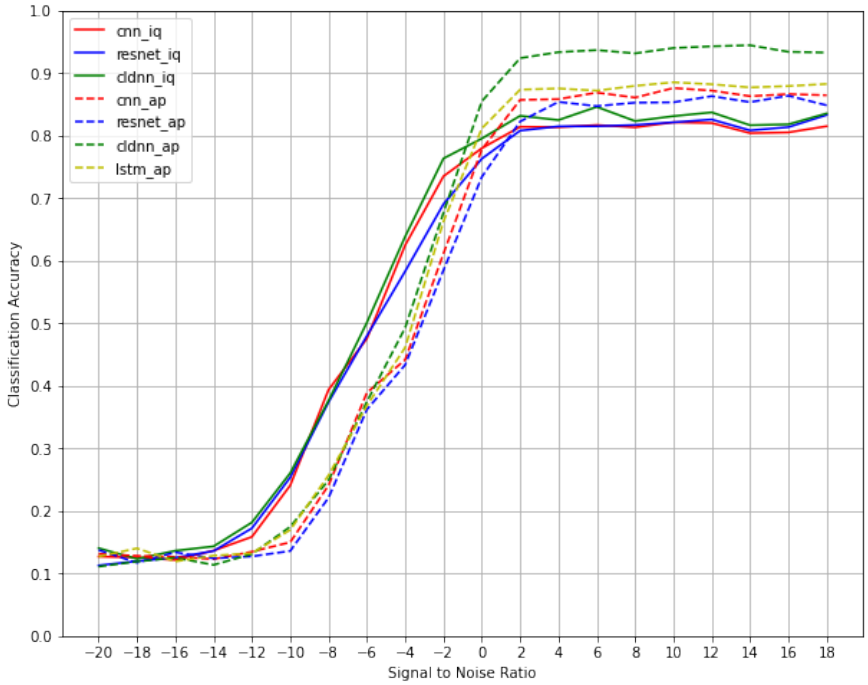

Figure 9: Classification accuracy over SNR range for time-series models

A few significant observations:

1. CLDNN model trained on amplitude-phase time-series outperformed the rest of the models significantly at high SNR. This could mean that a combination of CNN layers followed by LSTM layers is promising for the task of modulation classification.

2. Across all models, training with AP data yielded significantly higher (5%) accuracy at high SNRs.

3. Across all models, training with IQ data made them more resistant to low SNR conditions compared to training with AP data.

To get some insight into why the AP models are performing better than the IQ models at high SNR, I looked at the confusion matrices of cldnn-iq and cldnn-ap. It seems like AP data helps the model to differentiate QAMs and higher order PSKs better than the raw IQ data. 8PSK and QPSK were perfectly differentiated by cldnn-ap while there remained some confusion for differentiating QAMs. This seems to indicate that amplitude-phase time-series are more distinctive features for modulation classification, but they are more easily affected by noise conditions. These findings about the effect of amplitude-phase features are largely consistent with [4].

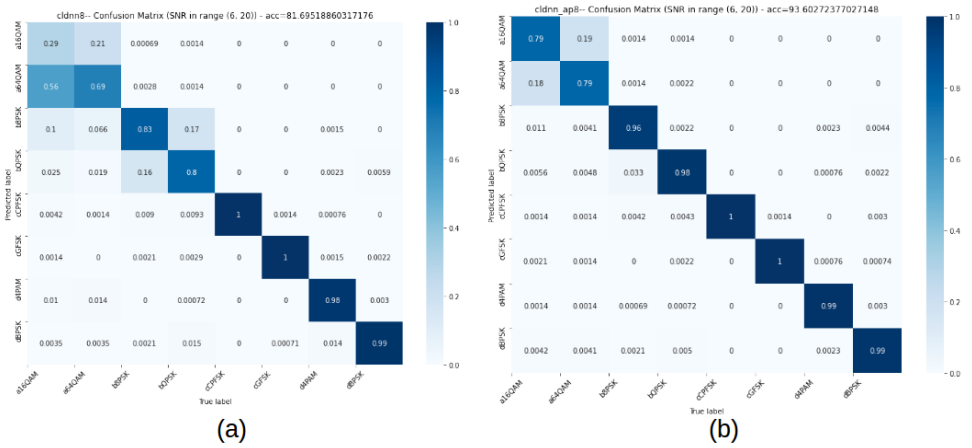

Figure 10: Confusion matrices of (a) cldnn-iq and (b) cldnn-ap at high SNR

4.2.2 Performance of constellation models on images of different resolutions and colourations

As expected, higher resolution results in slightly better performance, but it was surprising that adding colour to the constellations did not improve performance significantly and even worsens performance in some cases. Furthermore, the constellations were still not able to perfectly differentiate the QAMs unlike what was achieved in [8] [9], and only performed slightly better than the time-series model for QAMs differentiation. Perhaps, it was because there were only 128 data points in each constellation which corresponds to approximately 16 symbols given a sample rate of 8 samples per symbol in the radioML dataset, so not enough details were captured.

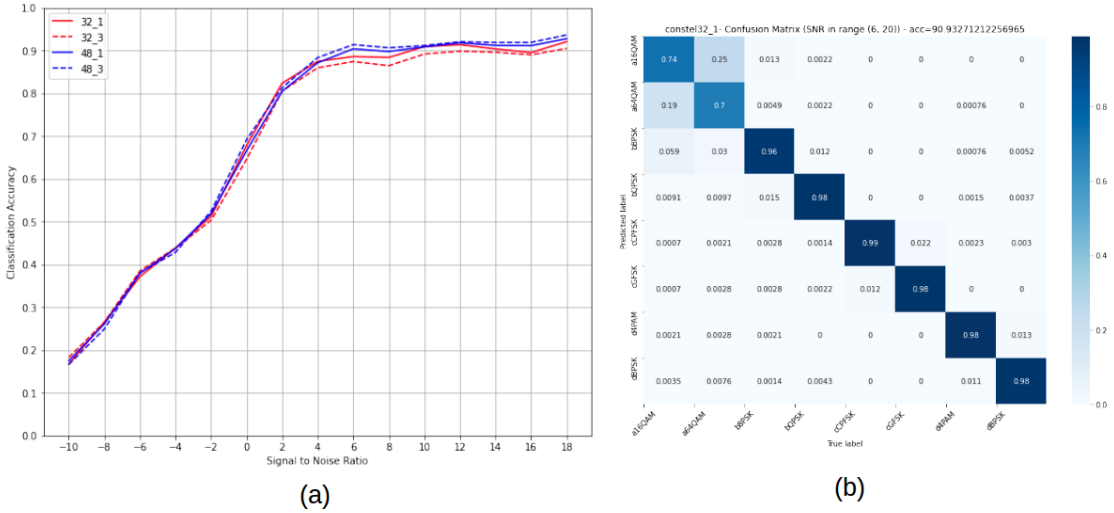

Figure 11: (a) Classification accuracy over SNR range for constellation models trained with different resolutions and colourations; (b) Confusion matrix for constellation model trained on resolution 32, single colour images

Even after I tried training on resolution 64 and resolution 96 constellations to differentiate between QAMs, I never managed to achieve 100% accuracy and only close to 90% for the highest SNRs.

4.2.3 Overall insights for Part 1

For the radioML dataset, it seems that the constellation feature does not help that much, and the CLDNN-AP model is good at high SNR while the CLDNN-IQ model is better at low SNRs. Therefore, combining the CLDNN-AP and CLDNN-IQ model seems like a good idea to get both advantages. In fact, [8] also suggests a similar approach by first having an IQ model to separate easier modulations and subsequently using a constellation model to differentiate more difficult modulations (see Figure 12). In my case, I used CLDNN-IQ for the first model, and CLDNN-AP for the second model. The resultant classification accuracy of the combined model is compared with the other individual models in Figure 13.

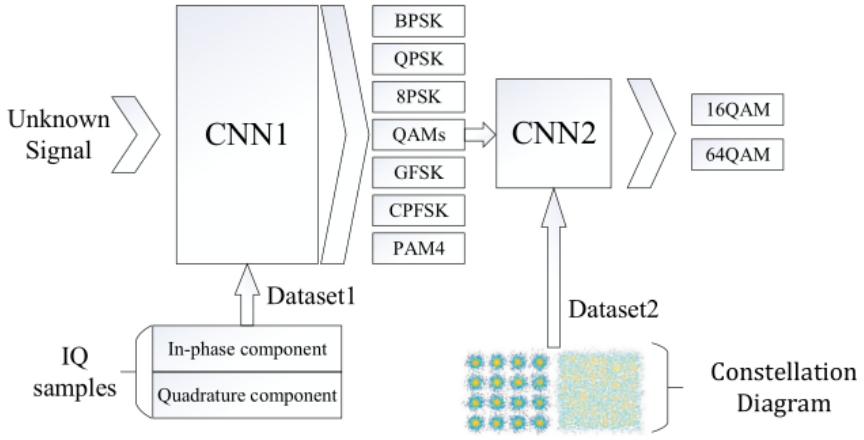

Figure 12: Approach of two cascaded models, IQ followed by constellation to separate classes of different difficulties [8]

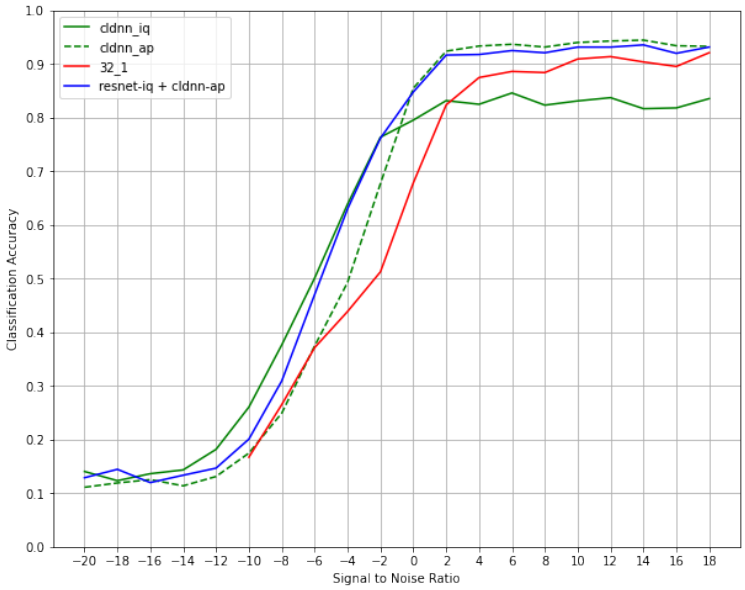

Figure 13: Classification accuracy over SNR range for cldnn-iq, cldnn-ap, and constellation models

As seen in Figure 13, the combined model manages to incorporate advantages of each model and performs well at both low and high SNRs.

5 Part 2: Evaluation of models on a more realistic dataset

Part 2 focuses on creating a dataset simulating a more realistic and difficult channel environment than the one in radioML, and testing how the best performing models and feature types from Part 1 perform on this dataset. Ideally, I would have created datasets simulating different types of conditions, such as foliage, urban and rain and seeing how the models respond to each condition, but due to time constraints, I simply followed this Matlab tutorial (https://www.mathworks.com/help/deeplearning/ug/modulation-classification-with-deep-learning.html) for realistic channel parameters. These are not specific to any particular environment.

5.1 Methodology: Dataset creation using Matlab

I used Matlab's communications toolbox to generate a more difficult RF dataset with the same 8 digital modulations in radioML, but this time each data sample has 2x1024 points but still roughly the same symbol rate (8 samples per symbol), and with 1000 samples for each (mod, snr) and snr ranging from -10dB to 30dB. Below, I explain the process of dataset creation, important channel parameters used and their justification.

5.1.1 Dataset generation process for M-ary modulation scheme

The general workflow for generating I/Q samples from an M-ary modulation scheme is shown below:

1. Source msg: Generate random symbols from 0 to M-1 with a uniform distribution.

2. Convert to I/Q complex form using modulation functions inbuilt into Matlab's communications toolbox.

3. Upsample to get 8 samples per symbol.

4. Filter with root-raised cosine filter with roll-off factor 0.35.

5. Apply channel effects.

6. Normalise and extract 1024 samples

5.1.2 Important parameters and channel effects

Important parameters and channels effects are shown in Table 3, and are input into Matlab's `comm.RicianChannel` System object, with the exception of max clock shift which was separately accounted for by readjusting frequency offset and sampling rate. To briefly explain the channel impairments: Rician fading assumes a direct line-of-sight propagation path superposed together with other reflected paths that follow Rayleigh fading processes; clock shift is from inaccuracies of Local Oscillators (LO) resulting from heat or another conditions leading to clock inaccuracies; Doppler shifts try to capture effects from moving emitters and receivers that cause frequency distortion.

Parameters	Value	Justification
Sampling freq	200kHz	
Carrier freq	902MHz	
Rician multipath fading	3 paths with: Delay [0, 9E-6, 1.7E-5] (s) Gains [0, -2, -10] (dB) Kfactor = 4	- Path delay of E-5 s typical of outdoors RF propagation, path length difference of ∼km - Gains arbitrary - Kfactor 4 typical for Rician fading, 0 for Rayleigh fading
Max clock shift (ppm)	Light - 0.001, Heavy - 5	- Affects frequency offset: f_offset = -fc*(clkshift/1M) - Affects sampling rate: fs_new = fs*(1+clkshift/1M)
Max Doppler shift (Hz)	4	-4Hz correspond to walking speed given f_c=902MHz
AWGN (dB)	-10 to 30	

Table 3: Table showing important parameters and channel effects

5.1.3 Creation of easy, medium and hard datasets

I decided to create three datasets of varying levels of difficulty, with the hard dataset being the one simulating the realistic and difficult channel conditions, while the easy and medium datasets had more perfect channel conditions (see complete description in Table 4, and dataset visualisations can be found in my notebook). The rationale for this is that I wanted to observe how well a model trained on one dataset would perform on a dataset that it had not train on, as this would be the typical situation in real-life.

	AWGN	Rician Fading	Max Clock Offset	Max Doppler Shift
Hard	-10 to 30dB	Present	5ppm	4Hz
Medium		Present	0.001ppm	None
Easy		None	None	None

Table 4: Table showing characteristics of hard, medium and easy datasets

In Figures 14, 15 and 16, I show the constellation diagrams of the 8 different digital modulations at SNR=30dB for the hard, medium and easy datasets respectively. I observe that the hard dataset is extremely noisy, with BPSK being completely unrecognisable , while the medium and easy datasets look much more discernable. However, even the easy dataset at 30dB, which does not undergo any channel impairments other than AWGN, looks noisier than radioML's dataset at 18dB (see Figure 1) which was supposed to have been affected by channel effects. This suggests the importance of a benchmark dataset as different ways of dataset creation, even for supposedly the same SNR level, result in datasets that are very different.

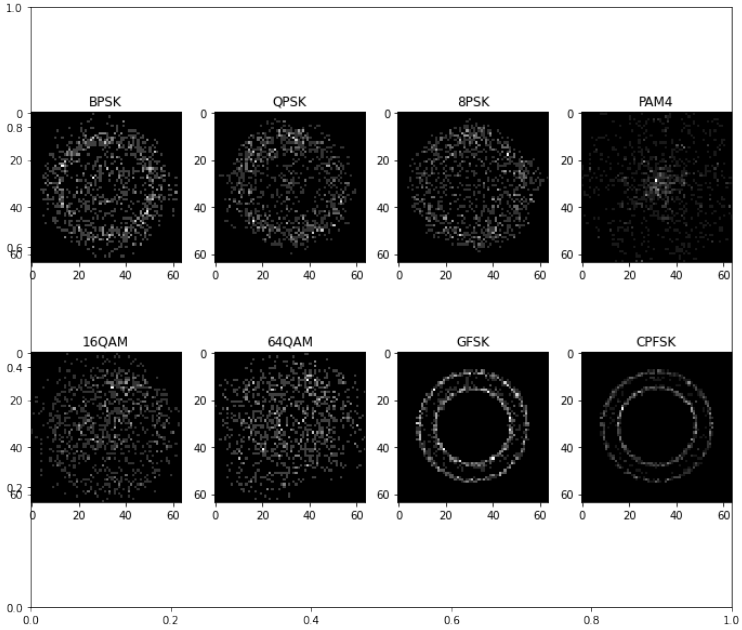

Figure 14: Constellation diagrams of different modulations at SNR=30dB for Matlab hard dataset

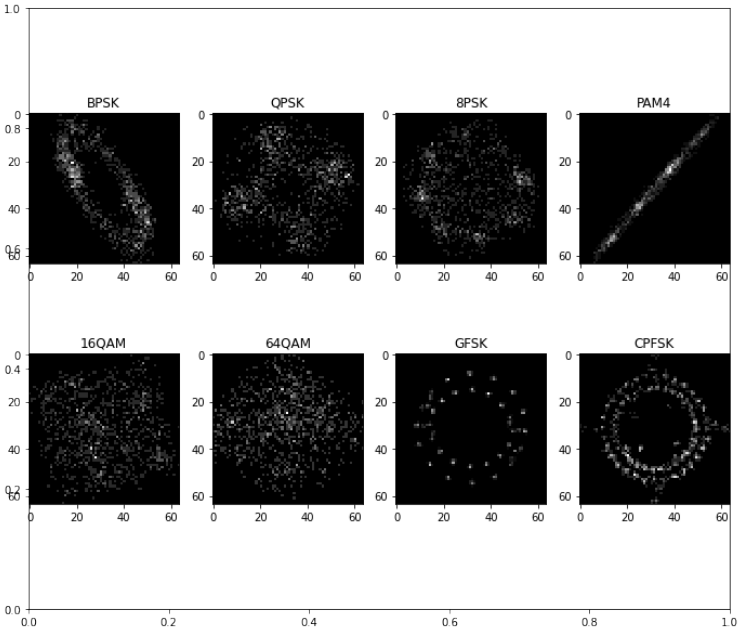

Figure 15: Constellation diagrams of different modulations at SNR=30dB for Matlab medium dataset

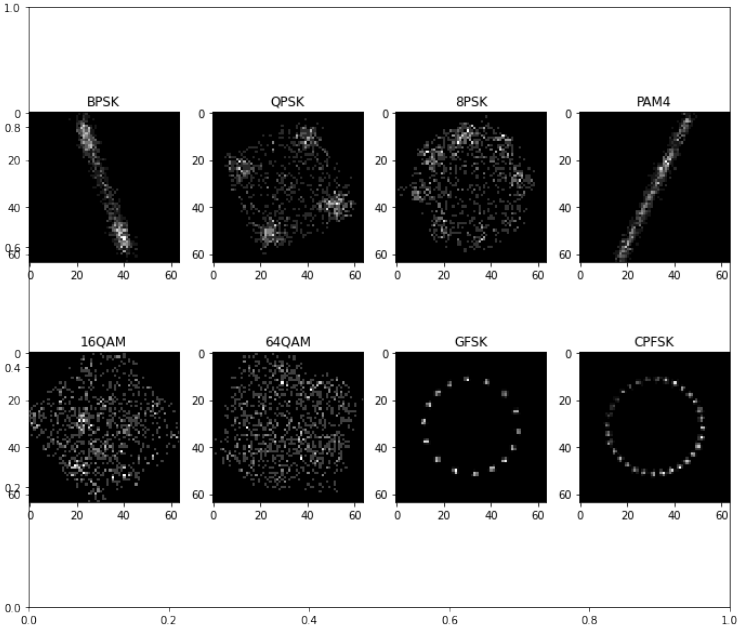

Figure 16: Constellation diagrams of different modulations at SNR=30dB for Matlab easy dataset

5.2 Results

5.2.1 Performance of models on hard (realistic) dataset

I tried the best performing models on the radioML dataset on the hard dataset, with full details in my `model_evaluation_matlab.ipynb` notebook. Figure 17 compares the classification accuracies of the ResNet-IQ, ResNet-AP and constellation model on the hard dataset. I did not put in CLDNN for comparison because surprisingly, CLDNN did not perform as well as ResNet on the hard dataset unike in the radioML dataset. Perhaps it was because the CLDNN architecture was already tuned for data samples of 2x128 dimension and thus not able to exploit the greater amount of information in 2x1024 data samples from the hard dataset, while ResNet, which had many skip connections in between layers, was able to glean more patterns from the longer input data. Perhaps combining ResNet with LSTM layers in a way similar to CLDNN would yield even better results.

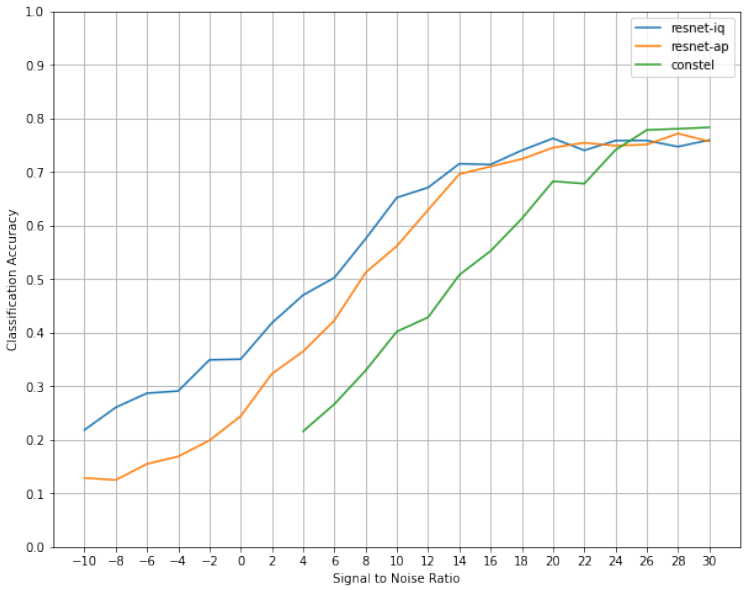

Figure 17: Classification accuracies over SNR for models on hard dataset

From Figure 17, the AQ data does not produce a big improvement in classification accuracy unlike in the radioML dataset. I suspect it is because the hard dataset is already quite noisy and amplitude-phase data is not resistant to noise. I expected the constellation model to achieve better results than in the radioML dataset because the hard dataset had longer data samples, but it only performs slightly better than ResNet at high SNR. A closer look at the confusion matrices (see Figure 18) show that while the constellation model was much better than ResNet-IQ at differentiating QAMs and PSKs, it was still nowhere close to 100% accuracy. Also, similar to the radioML dataset, the constellation model was not as good as the time-series models in telling apart the easier modulations (BPSK, 4PAM, GFSK, CPFSK).

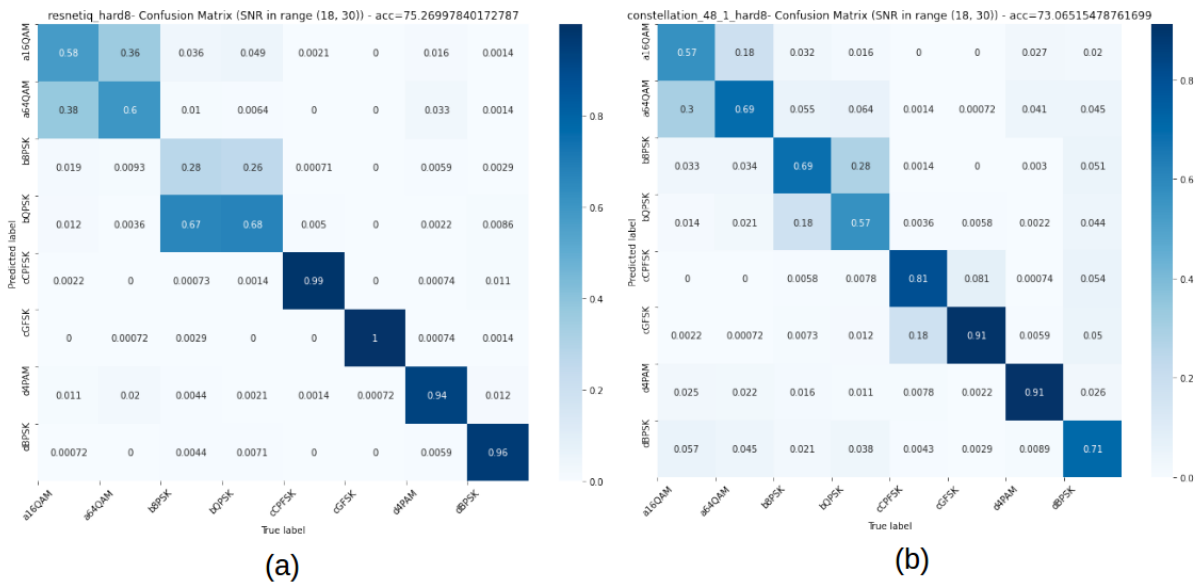

Figure 18: Confusion matrices of (a) ResNet-IQ and (b) constellation model at high SNR

Nevertheless, the constellation images and IQ time-series once again prove to be complementary features that are able to distinguish different types of modulations, and this suggests that future models should work on combining both features for better results.

5.2.2 Cross-evaluating models with easy, medium and hard datasets

This section was done with only a quarter of the full dataset because of memory limitations, and the model tested was a multi-input model that combined both the resnet-iq and constellation model. It essentially learns the best weight combination for both features extracted by the resnet-iq and constellation model (Figure 19). On the hard dataset, its performance was slightly worse than ResNet-IQ for low SNRs but it managed to outperform ResNet-IQ at high SNR, showing that it managed to synergise the two input features to some extent.

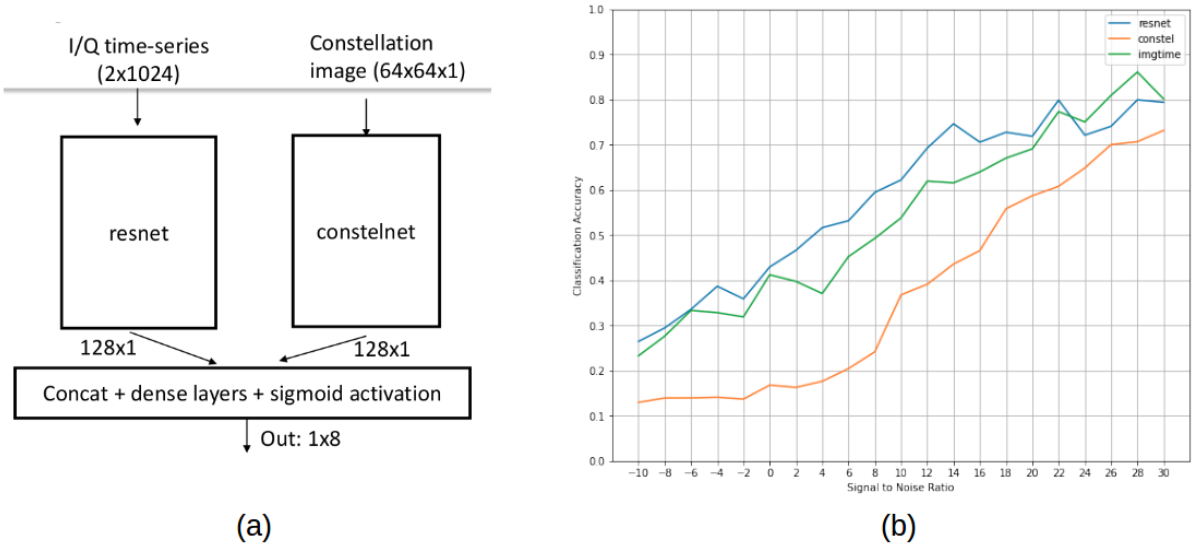

Figure 19: (a) Architecture of multi-input combined model (b) Classification accuracy of combined model vs individual models

A model was trained for each of the easy, medium and hard dataset, and cross-evaluated on each other to assess how well models can adapt to another dataset. Figure 20 shows the classification accuracy over SNR of the hard, medium and easy dataset, and unsurprisingly, the hard dataset had the worst performance.

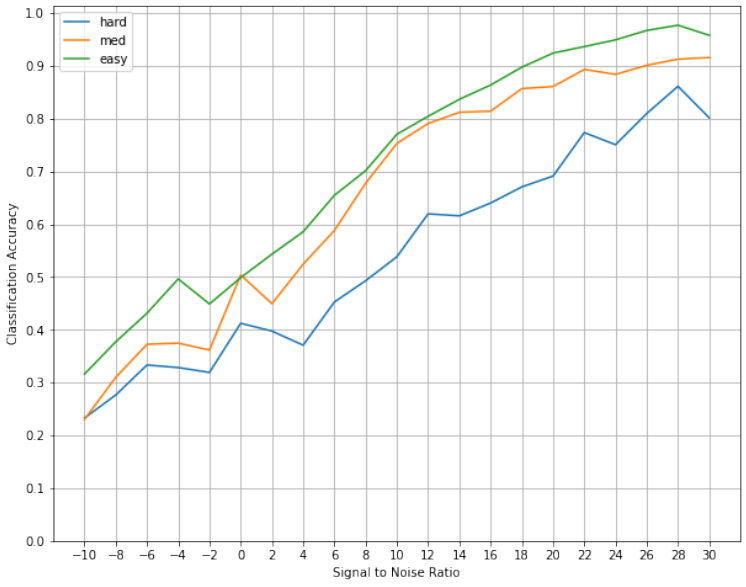

Figure 20: Classification accuracy over SNR on hard, medium and easy datasets

Table 5 shows the classification accuracy at 30dB when each model is cross-evaluated on another

dataset. It is unsurprising that models trained on easier datasets were unable to perform well on hard dataset, but it is surprising that models trained on harder datasets did not perform better when tested on easier datasets. This does not bode well for application of this model in real-life since the model is likely to be exposed to different noise conditions.

Classification accuracy at 30dB/%		**Tested on**		
		Easy	***Medium***	***Hard***
Trained on	***Easy***	95	87	48
	Medium	94	91	48
	Hard	68	71	81

Table 5: Table showing classification accuracy at 30dB when models are cross-evaluated on different datasets

5.2.3 Overall insights for Part 2

As expected, models do not perform so well on the Matlab hard dataset now that there is more noise and distortions. The behaviour of different features and models in differentiating different modulation type is largely similar to that of radioML dataset, with a few differences: amplitude-phase does not seem to have that great of an effect in accuracy and ResNet performed better than CLDNN. Constellation models were still unable to perfectly classify QAMs even at the highest SNRs which is very surprising. When tested for its ability to adapt to different datasets, the models performed poorly on another set it was not trained on. Overall, this is a very brief study and more conclusions can be drawn if we have data from a variety of channel conditions.

6 Conclusion and Future work

In this section, I summarise my main findings and suggest future directions to work on.

6.1 Model architecture: CLDNN, ResNet

For time-series input features, I found that the CLDNN architecture which combined both convolutional and LSTM layers was the best performing on the cleaner radioML dataset (2x128 time-series). This seems to suggest that the CNN+LSTM combination is useful for time-series analysis. On the other hand, ResNet was best performing on the Matlab hard dataset which had longer input time-series (2x1024), which seems to suggest that that skip connections between layers help to analyse longer time-series. Therefore, I conjecture that a network combining skip connections with a CNN+LSTM combination would work well on inputs with relatively large number of time-steps.

6.2 Input features

6.2.1 Time-series features

For time-series features, amplitude-phase (AP) format has shown improve model performance at high SNR but worsen performance at low SNR compared to raw IQ data, which led to me concluding that amplitude-phase time-series are more distinctive for each modulation type but also more easily corrupted by noise. Therefore, a combination of IQ and AP models might lead to better performance at both low and high SNR conditions.

6.2.2 Constellation image feature

For constellation images, I did not find that colouration improved model performance, but I did find that higher image resolution led to better performance, but this comes with a tradeoff in memory consumption. Also, constellation models are better at distinguishing higher order modulations such as QAMs and PSKs, but only at high SNR, and are very sensitive to noise conditions. Also, constellation models are not as good as time-series models in telling apart lower order modulations. This complementary behaviour of constellation and time-series models suggests that perhaps we can use constellation models only for higher order modulations, or have a model that combines both time-series and image features in model classification.

6.3 Model performance in different datasets

I found that general trends, such as the effects of amplitude-phase and constellation image features, apply across different datasets. However, the models that I tried adapted poorly when tested on a different dataset than it was trained on, and this is a big problem. A more rigorous analysis with data in different noise conditions can be undertaken to study model performance under different conditions.

6.4 Future work

In this project, I did not focus much on designing my own model architecture, but after analysing common models used in literature, I find that there is potential in designing ensemble-types of architectures that combine different models and input features for modulation classification. I was also quite disappointed that my coloured constellations did not work as well as expected, perhaps it could also be due to the architecture I used (which I tried to replicate from [8]), and more work can be done here to explore different architectures for constellation image classification. Finally, creating more datasets simulating different channel conditions and studying how models work under these conditions would be very helpful in realising effective modulation classification in real-life applications.

7 References

References

[1] T. O'Shea and N. West, "Radio machine learning dataset generation with gnu radio," *Proceedings of the GNU Radio Conference*, vol. 1, no. 1, 2016. [Online]. Available: https://pubs.gnuradio.org/index.php/grcon/article/view/11

[2] T. J. O'Shea and J. Corgan, "Convolutional radio modulation recognition networks," *CoRR*, vol. abs/1602.04105, 2016. [Online]. Available: http://arxiv.org/abs/1602.04105

[3] N. E. West and T. J. O'Shea, "Deep architectures for modulation recognition," *CoRR*, vol. abs/1703.09197, 2017. [Online]. Available: http://arxiv.org/abs/1703.09197

[4] M. Kulin, T. Kazaz, I. Moerman, and E. D. Poorter, "End-to-end learning from spectrum data: A deep learning approach for wireless signal identification in spectrum monitoring applications," *CoRR*, vol. abs/1712.03987, 2017. [Online]. Available: http://arxiv.org/abs/1712.03987

[5] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over the air deep learning based radio signal classification," *CoRR*, vol. abs/1712.04578, 2017. [Online]. Available: http://arxiv.org/abs/1712.04578

[6] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Distributed deep learning models for wireless signal classification with low-cost spectrum sensors," *CoRR*, vol. abs/1707.08908, 2017. [Online]. Available: http://arxiv.org/abs/1707.08908

[7] S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y. Yao, "Modulation classification using convolutional neural network based deep learning model," in *2017 26th Wireless and Optical Communication Conference (WOCC)*, 2017, pp. 1–5.

[8] Y. Wang, M. Liu, J. Yang, and G. Gui, "Data-driven deep learning for automatic modulation recognition in cognitive radios," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 4074–4077, 2019.

[9] B. Tang, Y. Tu, Z. Zhang, and Y. Lin, "Digital signal modulation classification with data augmentation using generative adversarial nets in cognitive radio networks," *IEEE Access*, vol. 6, pp. 15 713–15 722, 2018.

8 Appendix: How to access code and data files

1. All my notebooks and scripts for data generation are on github: https://github.com/interngithub2020/RF_modulation_classification.
2. The 2016.10A radioML dataset can be downloaded here: https://www.deepsig.ai/datasets
3. The Matlab RF dataset generation tutorial can be accessed here: https://www.mathworks.com/help/deeplearning/ug/modulation-classification-with-deep-learning.html.