Project 1

Neural Network for Modulation Classification

Juan Tarrat

1 Introduction

This project asked us to re-do the simulations done by the authors in the paper Convolutional Radio Modulation Recognition Networks, where the authors explored different architecture to generalize module recognition from the Radio ML dataset.

For this project, I tried to re-do the simulations done by the authors implementing several dense networks and several convolutional networks changing different parameters like the activation functions, architecture or learning rate; however, all the networks use the Adam optimizer. I developed the code using the Keras library.

2 The Dataset

The dataset used is the Radio ML dataset, which is a specialized collection of datasets for radio signal modulation classification. There are 11 modulation classes, 8 of which are digital and 3 are analog with a variable SNR (ranging from -20 to 18). The code is downloaded as a python dictionary with the keys being in the format (modulation name, SNR) and the values being a 2x128 time series.

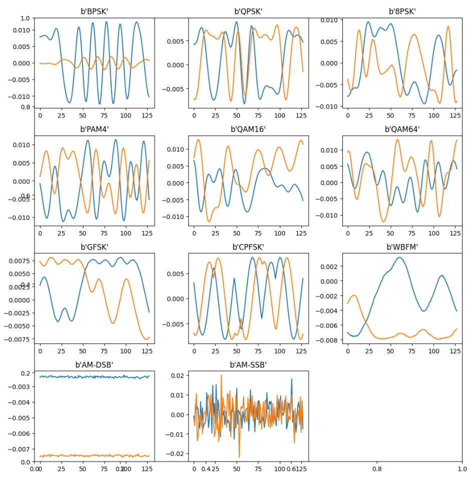
As part of the code, I included a function to plot the modulations under different SNRs: Below are two pictures showing -20 SNR and 18 SNR. As it can be observed, the higher the SNR, the smoother the signal. Results from the paper also show that as the SNR increases, the accuracy of the predictions also increase.

The function to plot the signals is provided below:

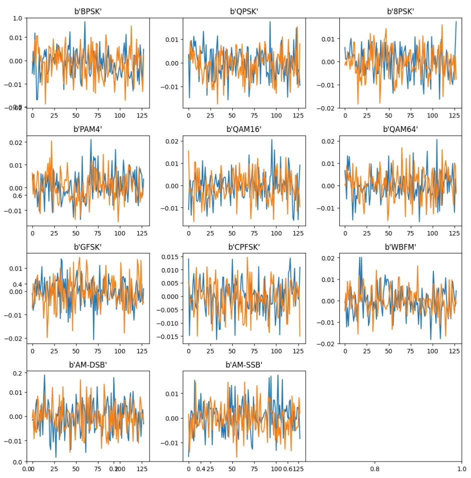
```
def plotTimeDomain(snr, offset):
    fig, ax = plt.subplots(figsize=(11,11))

    for i, mod in enumerate(modTypes):
```

```
    I = dataSet[(mod, snr)][offset][0]
    Q = dataSet[(mod, snr)][offset][1]
    ax = fig.add_subplot(4,3,i+1)
    plt.plot(I)
    plt.plot(Q)
    plt.title(mod)
    plt.tight_layout()
plt.show()
```



(a) 18 SNR



(b) -20 SNR

3 Simulation

There were 5 DNN models and 3 CNN models developed. The architecture of the DNN models is as follows:

- Input layer of 128 neurons
- Hidden layer of 1024 neurons
- Hidden layer of 512 neurons
- Output layer of 11 neurons

The loss function is cross-entropy loss. Models 2 through 5 vary in the activation function (sigmoid, tanh, selu, elu) and model 6 uses relu with a different learning rate (0.002) in the Adam optimizer.

There were 3 convolutional layers developed with the following architecture:

- Convolutional Layer – 256 filters – kernel size (1,3) - relu

- Max Pooling Layer – pool size (2,1)
- Convolutional Layer – 64 filters – kernel size (3,3)) - relu
- Max Pooling Layer – pool size (2,2)
- Input Dense Layer – 128 neurons - relu
- Dense Hidden Layer – 256 neurons - relu
- Output Layer - 11 neurons - softmax

Models 9-11 vary in their learning rate (0.001,0.002,0.003). All the dropout layers are 0.5.

3.1 Model 2 - Sigmoid

This is a dense model that used sigmoid in the output layer instead of softmax, here are the results.

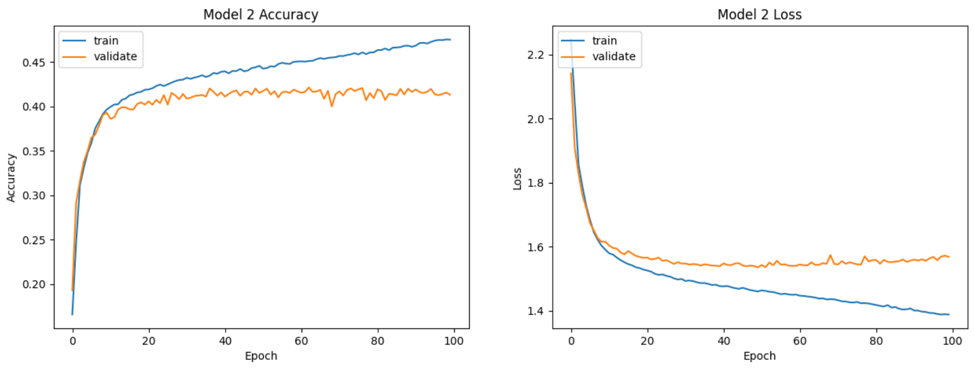


Figure 2: Model 2

3.2 Model 3 - Tanh

This is a dense model that used tanh in the output layer instead of softmax. The results were very dissapointing and not relevant at all so there is no point on plotting them

3.3 Model 4 - Selu

This is a dense model that used softmax in the output layer and Selu activation functions in the hidden layers

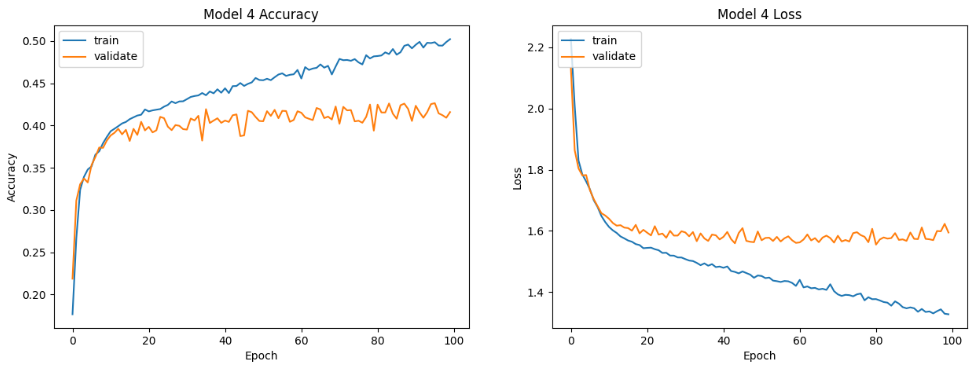


Figure 3: Model 2

3.4 Model 5 - Elu

This is a dense model that used softmax in the output layer and Elu activation functions in the hidden layers. This is the one that seems to have the brightest future since it does not seem to be converging yet and the accuracy is around 40%. There is not much overfitting either, which is a good sign.

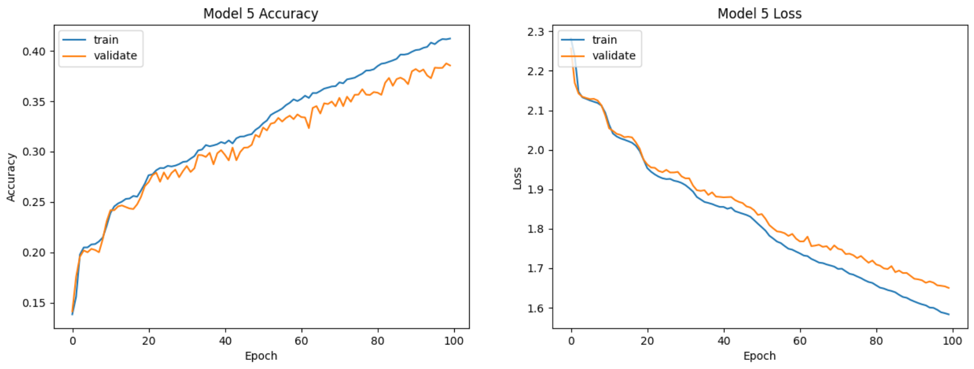


Figure 4: Model 2

3.5 Model 6 - Learning Rate = 0.002

This is a dense model that used softmax in the output layer and relu activation functions in the hidden layers but changed the learning rate from 0.001 to 0.002 in the Adam optimizer.

3.6 SNR Performance

[h] Below is the performance of each model with respect to the SNRs. As expected and as shown in the paper, the lower the SNR, the less accuracy the models show.

This is the code snippet that gets the accuracy for each SNR in the model given.

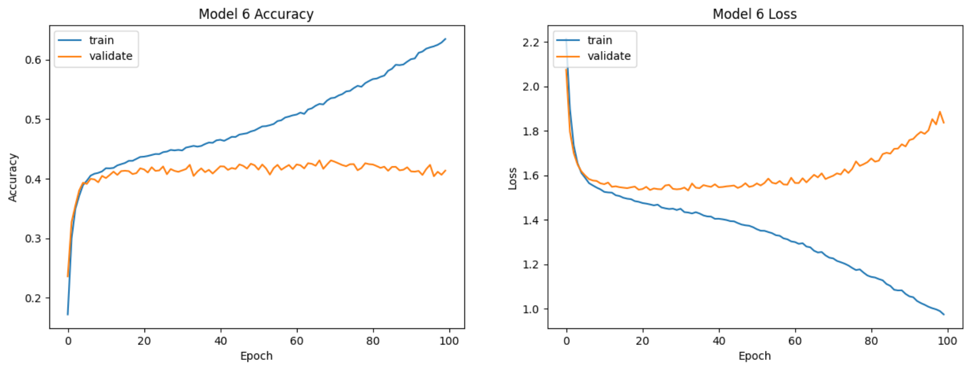


Figure 5: Model 2

```
def getSNRAccuracy(yPred, yTrue):
    x = list()
    y = list()

    for i in snrList:
        index = np.nonzero(snrTest == i)
        x.append(i)
        temp = accuracy_score(np.argmax(yTrue[index], axis=1), np.argmax(yPred[index], axis=1))
        y.append(temp)

    return x, y
```

3.7 CNN Evaluation

Below is the evaluation of every CNN developed. Sadly, I could not get the SNR performance since my computer turned off and the work was not saved so I would have had to re train the networks and it takes too long. In addition, the performance is not very good, on top of the lower accuracy, there is a lot of overfitting, so more regularization would be needed and a whole re evaluation of the architecture for enhanced performance.

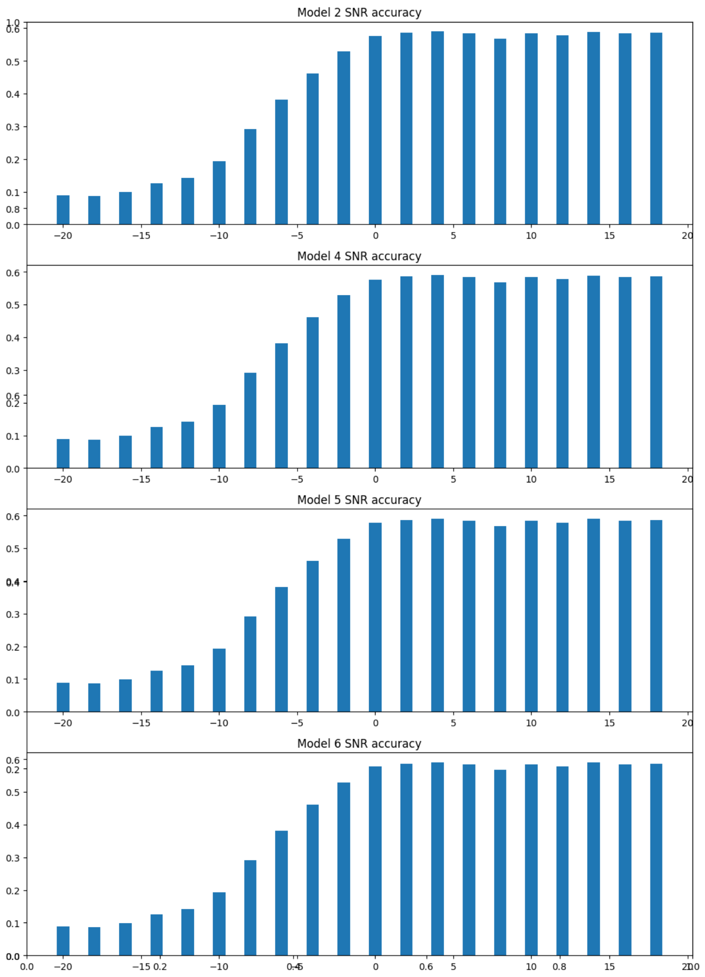
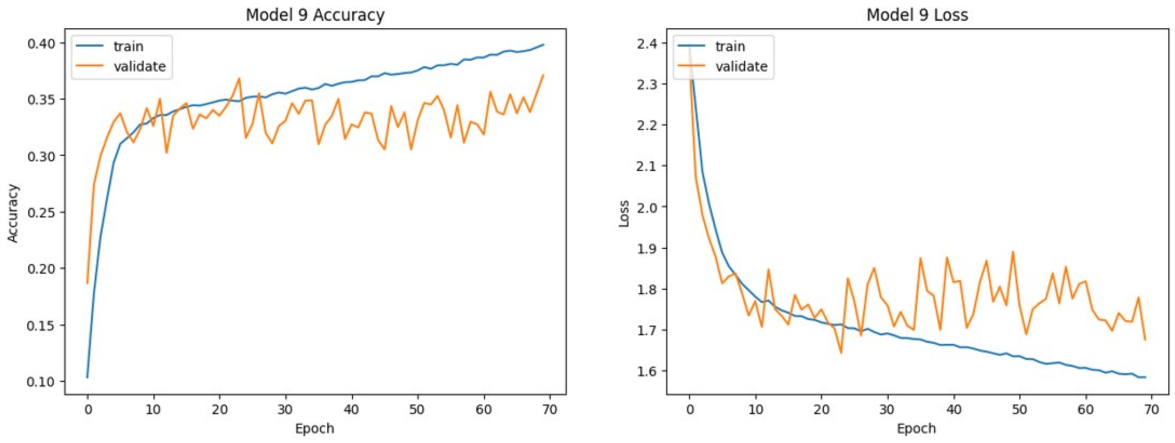
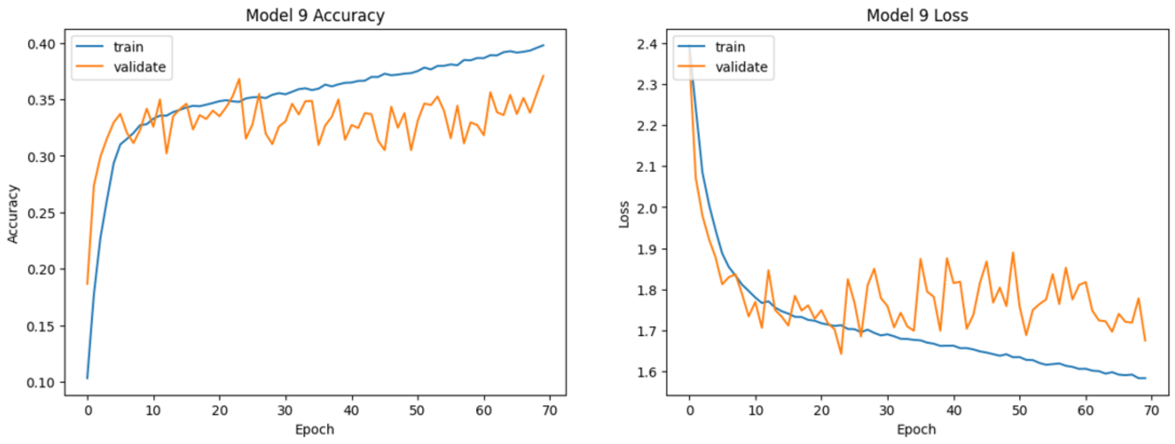


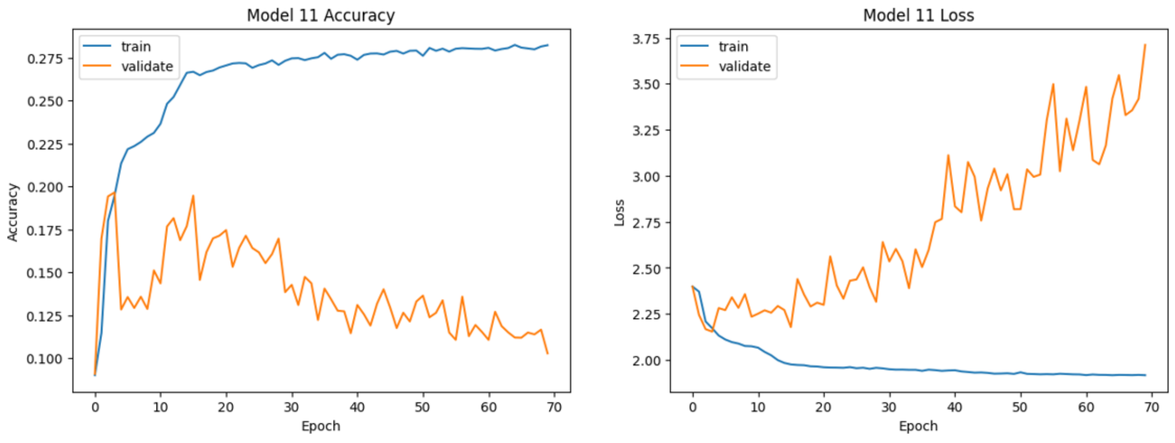
Figure 6: SNR vs Accuracy



(a) lr = 0.001



(b) lr = 0.002



(c) lr = 0.003