PATTERN RECOGNITION CCE

MODULATION CLASSIFICATION

FULLY CONNECTED NEURAL NETWORK AND CONVOLUTIONAL NEURAL NETWORK

Noha Nomier ID:4638

Hana Nazmy ID:4952

Fatma Sherif Tawfik ID:4701

PROBLEM STATEMENT

A synthetic dataset, generated with GNU Radio, consisting of 10 modulations. This is a variable-SNR dataset with moderate LO drift, light fading, and numerous different labeled SNR increments for use in measuring performance across different signal and noise power scenarios.

Dataset

- The dataset contains 20 modulation types and 10 SNR values. The combination between both will give us 200 type.
- Each type exists 6000 times.
- This makes the dataset size = 20*10*6000 = 1,200,000

Reading the dataset

We added the zipped file to the drive then we unzipped the file.

Loading the dataset

The output of unzipping was a .dat file extension. Therefore we used unpickle in order to return the data to the normal form.

Splitting the dataset

We used the train_test_split() function in order to split the dataset into 70% training and 30% testing.
Then we split the training set into 95% training and 5% validation.

Creating Feature spaces

Every sample is presented using two vectors each of them has 128 elements. In order to increase the features of the dataset we will need to create features from the derivative and the integral of the sample.

- ***Raw data***
  Raw time series that is originally given (two channels).
- ***Derivative***

By using the built-in function in the numpy library np.diff() we calculated the first derivative of the raw data and saved it. Now we have two more channels.

- ***Integral***

  By using the built-in function in scipy library scipy.integrate.cumtrapz we calculated the integral to the raw data and saved it. Now we have two more channels.

- ***Combinations***
  - ***Raw***

    Without adding any features to the dataset:
    The training set size is (798000,2,128)
    The validation set size is (42000,2,128)
    The test set size is (360000,2,128)
  - ***Raw + Derivative***

    We used concatenation to make a larger dataset including more features. Now the size of the sample is 4*128 instead of 2*128.
    The training set size is (798000,4,128)
    The validation set size is (42000,4,128)
    The test set size is (360000,4,128)
  - ***Raw + Integral***

    We used concatenation to make a larger dataset including more features. Now the size of the sample is 4*128 instead of 2*128.
    The training set size is (798000,4,128)
    The validation set size is (42000,4,128)
    The test set size is (360000,4,128)
  - ***Raw + Derivative + Integral***

    We used concatenation to make a larger dataset including more features. Now the size of the sample is 6*128 instead of 2*128.
    The training set size is (798000,6,128)
    The validation set size is (42000,6,128)
    The test set size is (360000,6,128)

Using Label Encoding

We used Label encoding of sklearn to convert the labels of the dataset to a specific value instead of a string. This is needed for the logistic regression classifier, Decision tree classifier and Random forest classifier.

Using One Hot Encoding

We used one hot encoding in order to convert the labels of the dataset into a 10 bits vector where there is only one single high (1) bit and all the others low (0). This is needed for the neural networks model

CLASSIFICAION:

Classification was done on two types of input, the first is using raw input data and the second is the combination of raw+derivative+integral

1. Logistic Regression Classifier:

Logistic Regression is a 'Statistical Learning' technique categorized in 'Supervised' Machine Learning (ML) methods dedicated to 'Classification' tasks. It has gained a tremendous reputation for last two decades especially in financial sector due to its prominent ability of detecting defaulters.

Using `sklearn.linear_model.LogisticRegression:`

- **Using raw data (2x128):**

Training data shape : `(798000, 256)`
Validation data shape : `(42000, 256)`
Test data shape : `(360000, 256)`

Output:

Validation Accuracy: **16.4%**
Test Accuracy: **16.3%**

- **Using combined data (6x128):**

Training data shape : `(798000, 768)`
Validation data shape : `(42000, 768)`
Test data shape : `(360000, 768)`

Output:

Validation Accuracy: **16.87 %**

Test Accuracy: **16.2%**

2. Decision Tree Classifier:

A Decision Tree is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter.

Using `sklearn.tree.DecisionTreeClassifier:`

- **Using raw data (2x128):**

Training data shape : `(798000, 256)`
Validation data shape : `(42000, 256)`
Test data shape : `(360000, 256)`

Output:

Validation Accuracy: **31.3%**
Test Accuracy: **30.9%**

- **Using combined data (6x128):**

Training data shape : `(798000, 768)`
Validation data shape : `(42000, 768)`
Test data shape : `(360000, 768)`

Output:

Validation Accuracy: **32.2 %**

Test Accuracy: **32.17%**

3. Random Forrest Classifier:

Random forest, consists of a large number of individual decision trees that operate as an **ensemble**. Each individual tree in the random forest outputs a class prediction and the class with the most votes becomes our model's prediction.

Using `sklearn.ensemble.RandomForestClassifier`:

- **Using raw data (2x128):**

Training data shape : `(798000, 256)`
Validation data shape : `(42000, 256)`
Test data shape : `(360000, 256)`

Output:

Validation Accuracy: **44.004%**
Test Accuracy: **44.02 %**

- **Using combined data (6x128):**

Training data shape : `(798000, 768)`
Validation data shape : `(42000, 768)`
Test data shape : `(360000, 768)`

Output:

Validation Accuracy: **42.4 %**

Test Accuracy: **42.56 %**

Convolutional Neural Network:

Output Plot of the accuracy against the SNR:

Fully Connected Layer Classifier:

We used the keras library to help us build our neural network. After experimenting with different numbers for the neurons and the layers, we found this architecture to provide us with the desired results. We use 4 dense layers, with 256, 200, 128 and 10 neurons respectively. The data is flattened before fed to our model, and the last layer has 10 neurons because we have 10 modulation classes. We used “categorical crossentropy” as our loss function, the Relu function for our activation function and Adam as our optimizer.

Learning rate = 0.001

```
def fully_connected(input_sz):
    model = Sequential()
    model.add(Reshape(input_sz+[1], input_shape=input_sz))
    model.add(Flatten())
    model.add(Dense(256 ,activation='relu', kernel_initializer='he_normal', name="dense1"))
    model.add(Dense(200, activation='relu', kernel_initializer='he_normal', name="dense2"))
    model.add(Dense(128, activation='relu', kernel_initializer='he_normal', name="dense3"))
    model.add(Dense(10, activation='softmax'))
    adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
    model.compile(loss='categorical_crossentropy', optimizer=adam , metrics=['accuracy'])
    model.summary()
    return model
```

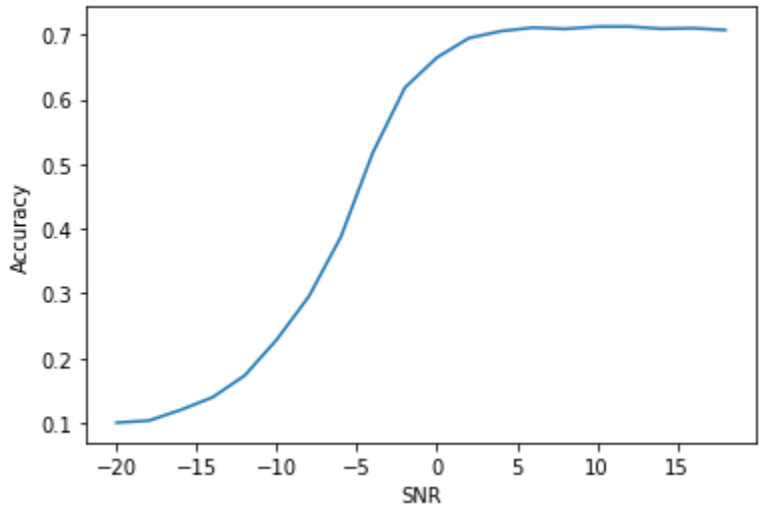
After inputting the raw data in the neural network, we received an accuracy of 53% on the training data and 48% on the validation data.

```
Epoch 38/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1180 - accuracy: 0.5241 - val_loss: 1.2449 - val_accuracy: 0.4855
Epoch 39/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1151 - accuracy: 0.5256 - val_loss: 1.2421 - val_accuracy: 0.4877
Epoch 40/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1129 - accuracy: 0.5261 - val_loss: 1.2323 - val_accuracy: 0.4912
Epoch 41/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1112 - accuracy: 0.5273 - val_loss: 1.2558 - val_accuracy: 0.4828
Epoch 42/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1083 - accuracy: 0.5284 - val_loss: 1.2432 - val_accuracy: 0.4886
Epoch 43/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1057 - accuracy: 0.5298 - val_loss: 1.2449 - val_accuracy: 0.4858
Epoch 44/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1046 - accuracy: 0.5299 - val_loss: 1.2475 - val_accuracy: 0.4884
Epoch 45/50
798000/798000 [==============================] - 6s 7us/step - loss: 1.1017 - accuracy: 0.5317 - val_loss: 1.2429 - val_accuracy: 0.4886
Epoch 00045: early stopping
```

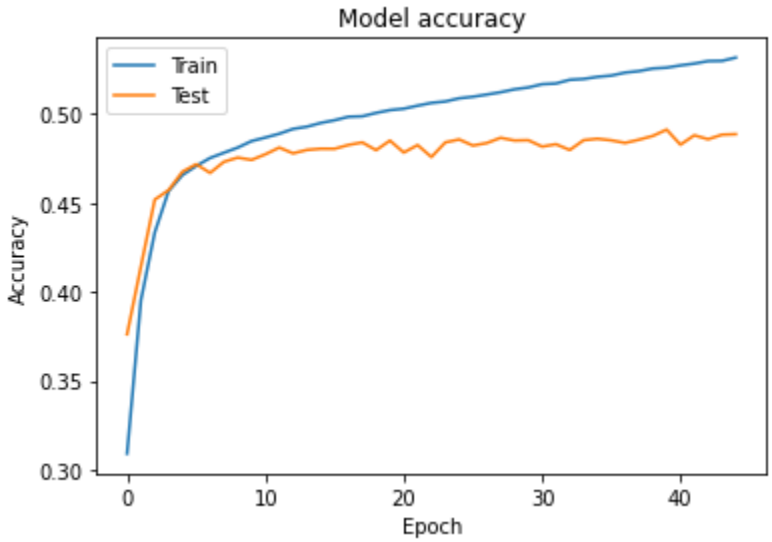
For the raw data with different SNR values, we received the following accuracies.

```
Accuracy is: 10.027777777777777
Accuracy is: 10.327777777777778
Accuracy is: 11.983333333333333
Accuracy is: 13.950000000000001
Accuracy is: 17.316666666666666
Accuracy is: 22.866666666666667
Accuracy is: 29.522222222222222
Accuracy is: 38.73888888888889
Accuracy is: 51.794444444444444
Accuracy is: 61.82222222222222
Accuracy is: 66.47777777777777
Accuracy is: 69.4888888888889
Accuracy is: 70.5611111111111
Accuracy is: 71.1
Accuracy is: 70.89999999999999
Accuracy is: 71.25555555555556
Accuracy is: 71.26666666666667
Accuracy is: 70.92222222222222
Accuracy is: 71.01111111111112
Accuracy is: 70.72777777777777
```

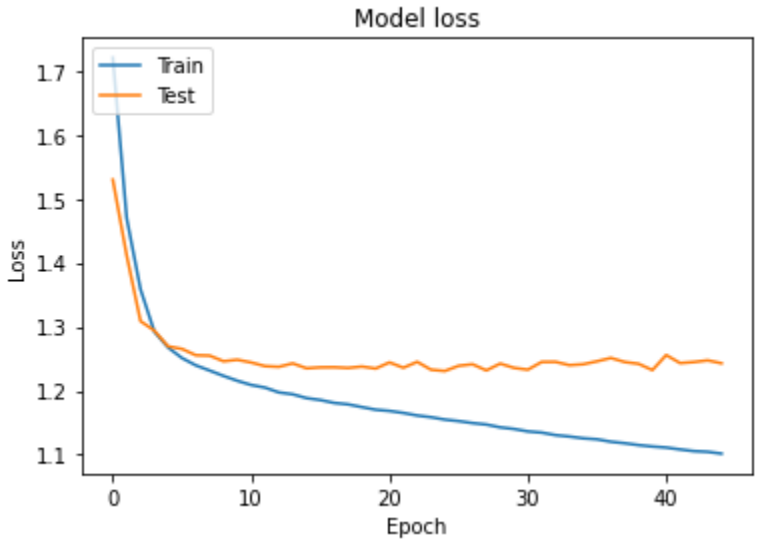
Plot for accuracies against the SNR values:



Plot of model's accuracy against each epoch:



Plot of Loss fn against the epochs:



After using the combined data [of size 1200000,6,128] to the neural network:

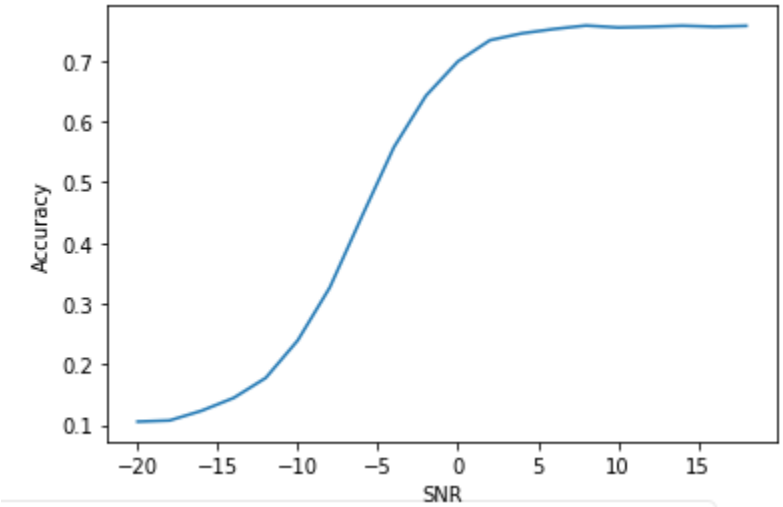
```
Epoch 38/50
798000/798000 [==============================] - 7s 9us/step - loss: 1.0527 - accuracy: 0.5588 - val_loss: 1.1710 - val_accuracy: 0.5175
Epoch 39/50
798000/798000 [==============================] - 7s 9us/step - loss: 1.0512 - accuracy: 0.5584 - val_loss: 1.1751 - val_accuracy: 0.5173
Epoch 40/50
798000/798000 [==============================] - 7s 9us/step - loss: 1.0469 - accuracy: 0.5609 - val_loss: 1.1738 - val_accuracy: 0.5201
Epoch 41/50
798000/798000 [==============================] - 8s 9us/step - loss: 1.0450 - accuracy: 0.5618 - val_loss: 1.1686 - val_accuracy: 0.5234
Epoch 42/50
798000/798000 [==============================] - 9s 11us/step - loss: 1.0432 - accuracy: 0.5628 - val_loss: 1.1762 - val_accuracy: 0.5212
Epoch 43/50
798000/798000 [==============================] - 8s 9us/step - loss: 1.0405 - accuracy: 0.5641 - val_loss: 1.1775 - val_accuracy: 0.5193
Epoch 00043: early stopping
```

We get an accuracy of 56% on the training data and 51% on the validation data.

Now feeding the combined test data separated by SNR values, we got these different accuracies:

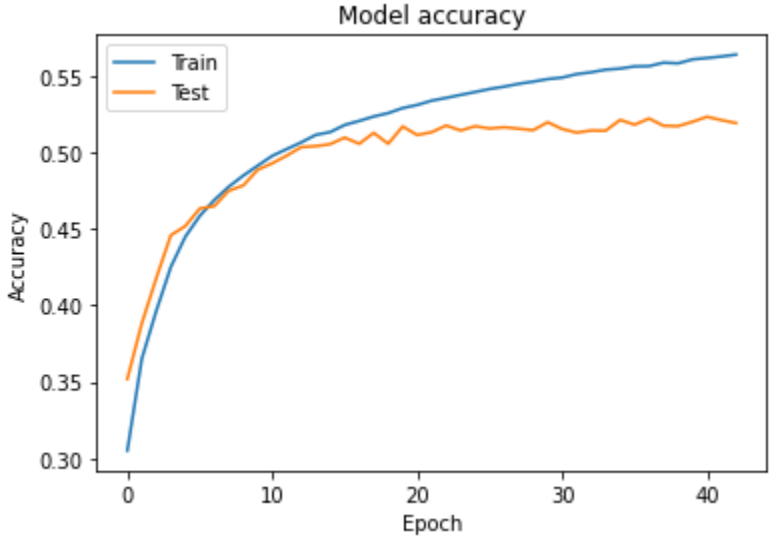
```
Accuracy is: 10.527777777777777
Accuracy is: 10.722222222222221
Accuracy is: 12.327777777777778
Accuracy is: 14.444444444444443
Accuracy is: 17.744444444444444
Accuracy is: 23.933333333333334
Accuracy is: 32.666666666666664
Accuracy is: 44.361111111111114
Accuracy is: 55.78888888888889
Accuracy is: 64.28333333333333
Accuracy is: 69.95555555555556
Accuracy is: 73.45
Accuracy is: 74.57777777777778
Accuracy is: 75.29444444444444
Accuracy is: 75.86666666666667
Accuracy is: 75.53888888888889
Accuracy is: 75.66111111111111
Accuracy is: 75.83888888888889
Accuracy is: 75.67222222222222
Accuracy is: 75.8111111111111
```

Plot of accuracies against SNR values:

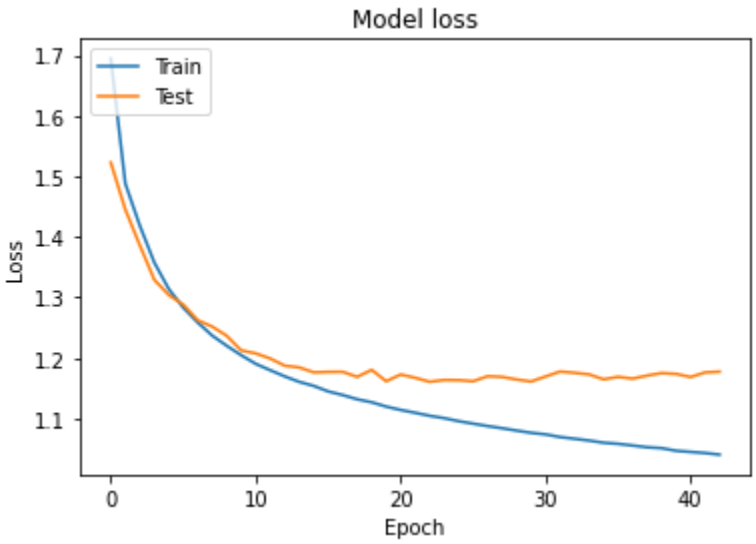


We can see that as SNR value increases, the accuracy increases.

Accuracy values with each epoch when training with the combined data:



Loss against each epoch:



Convolutional Neural Network:

We applied the CNN architecture as given in the problem statement.

```python
from keras.optimizers import Adam
def cnn(in_shp, dr):
    model = Sequential()
    model.add(Reshape(in_shp+[1], input_shape=in_shp))
    model.add(Conv2D(64, (1, 3), padding='same', activation='relu'))
    model.add(Dropout(dr))
    model.add(Conv2D(16, (2, 3), padding='same', activation='relu'))
    model.add(Dropout(dr))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(dr))
    model.add(Dense(10, activation='softmax'))
    adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
    model.compile(loss='categorical_crossentropy',
                  optimizer=adam,
                  metrics=['accuracy'])
    return model
```

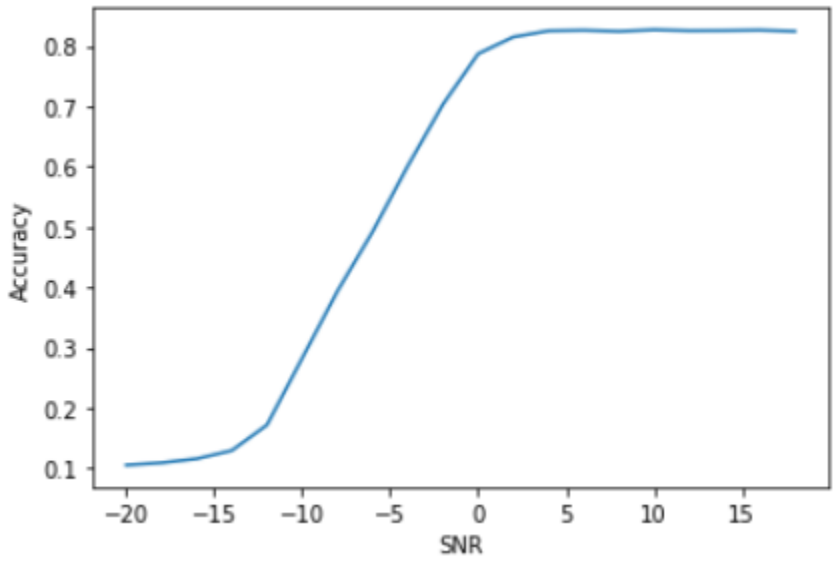
After Training the model with the raw data:

```
798000/798000 [==============================] - 15s 19us/step - loss: 1.0303 - accuracy: 0.5683 - val_loss: 1.0421 - val_accuracy: 0.5644
Epoch 49/50
798000/798000 [==============================] - 15s 19us/step - loss: 1.0290 - accuracy: 0.5686 - val_loss: 1.0420 - val_accuracy: 0.5617
Epoch 50/50
798000/798000 [==============================] - 15s 19us/step - loss: 1.0274 - accuracy: 0.5692 - val_loss: 1.0431 - val_accuracy: 0.5648
360000/360000 [==============================] - 22s 62us/step
0.5654194355010986
```

We received an accuracy of 56.5% on the training set and 56.4% on the validation set, and for the data divided by SNR values:

```
[[0.10494445]
 [0.10872222]
 [0.11544444]
 [0.12911111]
 [0.17155555]
 [0.28233334]
 [0.39327776]
 [0.49200001]
 [0.60111111]
 [0.70327777]
 [0.7871111 ]
 [0.81494445]
 [0.82522219]
 [0.82611114]
 [0.82438886]
 [0.82688886]
 [0.82544446]
 [0.82566667]
 [0.82633334]
 [0.82450002]]
```

Plot of Accuracies against SNR values:



For combined Data:

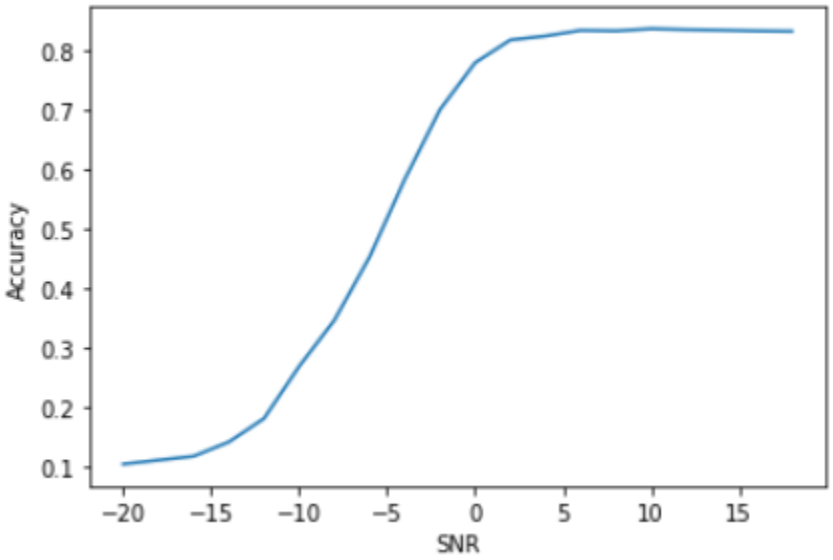
```
Epoch 41/50
798000/798000 [==============================] - 38s 47us/step - loss: 1.0009 - accuracy: 0.5809 - val_loss: 1.0893 - val_accuracy: 0.5606
Epoch 42/50
798000/798000 [==============================] - 37s 47us/step - loss: 0.9981 - accuracy: 0.5829 - val_loss: 1.0791 - val_accuracy: 0.5625
Epoch 43/50
798000/798000 [==============================] - 37s 47us/step - loss: 0.9959 - accuracy: 0.5839 - val_loss: 1.0887 - val_accuracy: 0.5602
Epoch 44/50
798000/798000 [==============================] - 37s 47us/step - loss: 0.9935 - accuracy: 0.5838 - val_loss: 1.0884 - val_accuracy: 0.5615
Epoch 45/50
798000/798000 [==============================] - 37s 47us/step - loss: 0.9915 - accuracy: 0.5858 - val_loss: 1.0866 - val_accuracy: 0.5577
Epoch 46/50
798000/798000 [==============================] - 37s 47us/step - loss: 0.9890 - accuracy: 0.5865 - val_loss: 1.0888 - val_accuracy: 0.5624
Epoch 00046: early stopping
360000/360000 [==============================] - 30s 84us/step
0.5638555288314819
```

We received an accuracy of 56.3% on training data, and an accuracy of 56.24% on the validation data.

For Data separated by SNR Values we received these accuray values:

```
[[0.10388889]
 [0.1105    ]
 [0.11716667]
 [0.14111111]
 [0.18077777]
 [0.26872224]
 [0.34638888]
 [0.45305556]
 [0.58399999]
 [0.70133334]
 [0.78049999]
 [0.8187778 ]
 [0.82544446]
 [0.83494443]
 [0.83405554]
 [0.83761114]
 [0.83616668]
 [0.8352778 ]
 [0.83399999]
 [0.83338886]]
```

With highest accuracy of value 83.3%



Confusion Matrix for Classification by Convolutional Neural Network with raw test data as input to model: