

Assignment 3

Modulation Classification

Guehad Mohamed 3861

Menna Ghanem 3798

Esraa Wael El-Hawash 3959

Introduction

We were asked to classify signals into their proper modulation type using a synthetic dataset, generated with GNU Radio, consisting of 11 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.

Step 1: Downloading Dataset:

We downloaded the dataset from the website and then we uploaded it to google drive and we saved the file in a .pkl form in order to use the `_unpickler` function to extract data from a .dat file.

```
download = drive.CreateFile({'id': '1iG8UA_Loj5SJvuU_cFGZ_WfbBuCQZyXM'})
```

```
download.GetContentFile('DOWNLOAD.pkl')
```

And we open the DOWNLOAD.pkl file as 'rb' like we would any other ordinary file.

Step 2: Create feature Spaces:

First thing we had to do was understand exactly how the dataset was presented in order to deal with it, so we downloaded the data using pickle library and we put all the data into a dictionary and we printed said dictionary.


Inside it was each modulation type and it's respective SNR value as such ('8PSK' , -20) with SNRs ranging from [-20] to [18] with a step of 2 for each modulation type, which are 10 total modulation types (8PSK , AM-DSB, BPSK, CPFSK, GFSK, PAM4, QAM-16, QAM-64, QPSK, WBFM)

Each modulation type and equivalent snr has 3000 samples in it , each one of these samples is represented by 2 vectors of 128 element each.

Which gave us a total of 1,200,000 sample for all given dataset and a shape of (1200000, 128,2)

Next thing was separating the labels of each signal and putting them in arrays, we did so by extracting the first dimension of the 2D array and saved them as mods (modulation type) and the second dimension was saved as SNRs.

We created a third array of elements [] and within a nested loop we access every modulation alongside it's 20 equivalent SNR to extract the samples and vectors representing that certain



signal, and we put the actual label (both modulation type and SNR) of a certain signal in an array of labels.

Differentiation:

For the differentiation part we used `np.diff(traindata)` function which calculates the slope of the signals and therefore the derivative, but we had to concatenate a zero to the end of the array to fix the shape back to 128 in vectors instead of 127 produced from the subtraction.

Integration :

For the integration we used `integrate.cumtrapz(X_npArray, initial=0)` which is a built in function that does the integration for us, no concatenation was needed for that one.

Step 3: Supervised Learning:

```
np.random.seed(2016)
```

For splitting the data, we generate an array of train indices by using an `np.random.choice()` function which ranges from 0 to all examples in the dataset and only choose a size of 50% of the examples without replacement and we subtract the remaining examples and put them in test indices.

Then we create an array of the elements (vectors) of those indices giving us two halves of the data generated randomly.

For the labels we first have to represent the modulation types which are string into their numerical value by enumerating them or mapping each index to a number indicating a specific modulation type using the training indices we extracted.

But that's not enough, we then have to convert the numerical figures into binary figures using the 'one hot' concept which gives a vector of zeroes and ones representing the labels in our dataset.

Fully Connected Neural Network:

We used Keras for the implementation of our neural networks.

We decided on the sequential implementation of the model so we used

`model = keras.models.Sequential()`, and started adding layer by layer to our network. First things first we used the `.add()` method and the `reshape()` method to change the dimensions from (128,2) to (128,2,1) and we set the input shape to [2,128] which is the size of each sample.

Then we added a dropout layer and a flatten layer that has a job of connected the input layer to our hidden layer. We kept adding hidden layers with different neurons all with the Dense function and All with the Relu activation function.

Last layer [the output layer] has 10 neurons which is the number of classes we have [modulation types]. In the end we compile using the optimizer 'adam' and the following parameters.

```
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

We tried different combinations to find the best parameters for our model and we followed some rules to reduce the number of trials :

- 1-The number of hidden neurons should be between the size of the input layer and the size of the output layer
- 2-The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer
- 3-The number of hidden neurons should be less than twice the size of the input layer

Some of our trials are shown below:

We fixed the number of epochs to be always 100 and the dropout ratio to be 0.1

Trial	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Batch Size	accuracy
1	256	256	128	128			200	0.51781499 84
2	128	128	64	64			200	0.4504366 666
3	256	256	128	128	64	64	200	0.46639166 61
4	256	256	128	128	64	64	200	0.46738166 71
5	256	256	128	128			600	0.51858166 38
6	256	256	180	180			600	0.52345166 57
7	170	170	85	85			600	0.4576900 005
8	256	256	180	180			270	0.4902033 364

We choose to work with the best accuracy maintained by tuning the model marked in the above table. And then we got our **summary** of the model :

Layer (type)	Output Shape	Param #
reshape_25 (Reshape)	(None, 2, 128, 1)	0
dropout_72 (Dropout)	(None, 2, 128, 1)	0
flatten_14 (Flatten)	(None, 256)	0
dense5 (Dense)	(None, 256)	65792
dropout_73 (Dropout)	(None, 256)	0
dense6 (Dense)	(None, 256)	65792
dropout_74 (Dropout)	(None, 256)	0
dense7 (Dense)	(None, 180)	46260
dropout_75 (Dropout)	(None, 180)	0
dense8 (Dense)	(None, 180)	32580
dropout_76 (Dropout)	(None, 180)	0
dense11 (Dense)	(None, 10)	1810
activation_12 (Activation)	(None, 10)	0
reshape_26 (Reshape)	(None, 10)	0
=====		
Total params: 212,234		
Trainable params: 212,234		
Non-trainable params: 0		

The training process took 55 epochs to be done :

```
Train on 570000 samples, validate on 30000 samples
Epoch 1/100
- 7s - loss: 1.7369 - acc: 0.2974 - val_loss: 1.5134 - val_acc: 0.3713
Epoch 2/100
- 5s - loss: 1.4649 - acc: 0.3873 - val_loss: 1.3720 - val_acc: 0.4225
Epoch 3/100
- 5s - loss: 1.3900 - acc: 0.4110 - val_loss: 1.3416 - val_acc: 0.4387
Epoch 4/100
- 5s - loss: 1.3673 - acc: 0.4203 - val_loss: 1.3438 - val_acc: 0.4381
Epoch 5/100
- 5s - loss: 1.3541 - acc: 0.4256 - val_loss: 1.3128 - val_acc: 0.4414
Epoch 6/100
- 5s - loss: 1.3428 - acc: 0.4298 - val_loss: 1.3137 - val_acc: 0.4494
Epoch 7/100
- 5s - loss: 1.3331 - acc: 0.4334 - val_loss: 1.3227 - val_acc: 0.4436
Epoch 8/100
- 5s - loss: 1.3249 - acc: 0.4354 - val_loss: 1.3199 - val_acc: 0.4446
Epoch 9/100
- 5s - loss: 1.3179 - acc: 0.4388 - val_loss: 1.2905 - val_acc: 0.4550
Epoch 10/100
- 5s - loss: 1.3120 - acc: 0.4407 - val_loss: 1.3006 - val_acc: 0.4411
Epoch 11/100
- 5s - loss: 1.3065 - acc: 0.4422 - val_loss: 1.2902 - val_acc: 0.4543
Epoch 12/100
- 5s - loss: 1.3015 - acc: 0.4443 - val_loss: 1.3099 - val_acc: 0.4414
Epoch 13/100
- 5s - loss: 1.2969 - acc: 0.4459 - val_loss: 1.2874 - val_acc: 0.4556
Epoch 14/100
- 5s - loss: 1.2924 - acc: 0.4483 - val_loss: 1.3013 - val_acc: 0.4477
Epoch 15/100
- 5s - loss: 1.2880 - acc: 0.4507 - val_loss: 1.2870 - val_acc: 0.4559
Epoch 16/100
- 5s - loss: 1.2850 - acc: 0.4518 - val_loss: 1.2867 - val_acc: 0.4528
Epoch 17/100
- 5s - loss: 1.2809 - acc: 0.4528 - val_loss: 1.2754 - val_acc: 0.4570
Epoch 18/100
- 5s - loss: 1.2770 - acc: 0.4553 - val_loss: 1.2941 - val_acc: 0.4500
Epoch 19/100
- 5s - loss: 1.2724 - acc: 0.4572 - val_loss: 1.2833 - val_acc: 0.4554
```

```
Epoch 20/100
- 5s - loss: 1.2692 - acc: 0.4589 - val_loss: 1.2745 - val_acc: 0.4614
Epoch 21/100
- 5s - loss: 1.2648 - acc: 0.4616 - val_loss: 1.2818 - val_acc: 0.4568
Epoch 22/100
- 5s - loss: 1.2607 - acc: 0.4631 - val_loss: 1.2787 - val_acc: 0.4554
Epoch 23/100
- 5s - loss: 1.2560 - acc: 0.4657 - val_loss: 1.2805 - val_acc: 0.4564
Epoch 24/100
- 5s - loss: 1.2517 - acc: 0.4679 - val_loss: 1.2618 - val_acc: 0.4638
Epoch 25/100
- 5s - loss: 1.2477 - acc: 0.4694 - val_loss: 1.2743 - val_acc: 0.4567
Epoch 26/100
- 5s - loss: 1.2424 - acc: 0.4717 - val_loss: 1.2750 - val_acc: 0.4544
Epoch 27/100
- 5s - loss: 1.2373 - acc: 0.4742 - val_loss: 1.2525 - val_acc: 0.4680
Epoch 28/100
- 5s - loss: 1.2340 - acc: 0.4765 - val_loss: 1.2632 - val_acc: 0.4591
Epoch 29/100
- 5s - loss: 1.2295 - acc: 0.4782 - val_loss: 1.2484 - val_acc: 0.4690
Epoch 30/100
- 5s - loss: 1.2253 - acc: 0.4798 - val_loss: 1.2554 - val_acc: 0.4662
Epoch 31/100
- 5s - loss: 1.2187 - acc: 0.4831 - val_loss: 1.2328 - val_acc: 0.4729
Epoch 32/100
- 5s - loss: 1.2161 - acc: 0.4844 - val_loss: 1.2414 - val_acc: 0.4716
Epoch 33/100
- 5s - loss: 1.2126 - acc: 0.4853 - val_loss: 1.2527 - val_acc: 0.4642
Epoch 34/100
- 5s - loss: 1.2083 - acc: 0.4873 - val_loss: 1.2323 - val_acc: 0.4749
Epoch 35/100
- 5s - loss: 1.2035 - acc: 0.4889 - val_loss: 1.2296 - val_acc: 0.4711
Epoch 36/100
- 5s - loss: 1.2008 - acc: 0.4904 - val_loss: 1.2244 - val_acc: 0.4768
Epoch 37/100
- 5s - loss: 1.1971 - acc: 0.4921 - val_loss: 1.2283 - val_acc: 0.4748
Epoch 38/100
- 5s - loss: 1.1930 - acc: 0.4939 - val_loss: 1.2339 - val_acc: 0.4746
Epoch 39/100
- 5s - loss: 1.1902 - acc: 0.4951 - val_loss: 1.2164 - val_acc: 0.4827
```



```

Epoch 40/100
- 5s - loss: 1.1868 - acc: 0.4966 - val_loss: 1.2144 - val_acc: 0.4824
Epoch 41/100
- 5s - loss: 1.1832 - acc: 0.4982 - val_loss: 1.2203 - val_acc: 0.4830
Epoch 42/100
- 5s - loss: 1.1809 - acc: 0.4993 - val_loss: 1.2254 - val_acc: 0.4788
Epoch 43/100
- 5s - loss: 1.1777 - acc: 0.4998 - val_loss: 1.2060 - val_acc: 0.4882
Epoch 44/100
- 5s - loss: 1.1760 - acc: 0.5015 - val_loss: 1.2015 - val_acc: 0.4902
Epoch 45/100
- 5s - loss: 1.1728 - acc: 0.5019 - val_loss: 1.2218 - val_acc: 0.4794
Epoch 46/100
- 5s - loss: 1.1698 - acc: 0.5041 - val_loss: 1.2033 - val_acc: 0.4918
Epoch 47/100
- 5s - loss: 1.1678 - acc: 0.5049 - val_loss: 1.1974 - val_acc: 0.4897
Epoch 48/100
- 5s - loss: 1.1659 - acc: 0.5050 - val_loss: 1.2015 - val_acc: 0.4904
Epoch 49/100
- 5s - loss: 1.1633 - acc: 0.5062 - val_loss: 1.2072 - val_acc: 0.4867
Epoch 50/100
- 5s - loss: 1.1610 - acc: 0.5076 - val_loss: 1.1828 - val_acc: 0.4962
Epoch 51/100
- 5s - loss: 1.1593 - acc: 0.5077 - val_loss: 1.1959 - val_acc: 0.4906
Epoch 52/100
- 6s - loss: 1.1574 - acc: 0.5077 - val_loss: 1.1989 - val_acc: 0.4892
Epoch 53/100
- 6s - loss: 1.1562 - acc: 0.5089 - val_loss: 1.2057 - val_acc: 0.4877
Epoch 54/100
- 5s - loss: 1.1543 - acc: 0.5097 - val_loss: 1.1940 - val_acc: 0.4947
Epoch 55/100
- 5s - loss: 1.1529 - acc: 0.5100 - val_loss: 1.1894 - val_acc: 0.4947
600000/600000 [=====] - 2s 4us/step

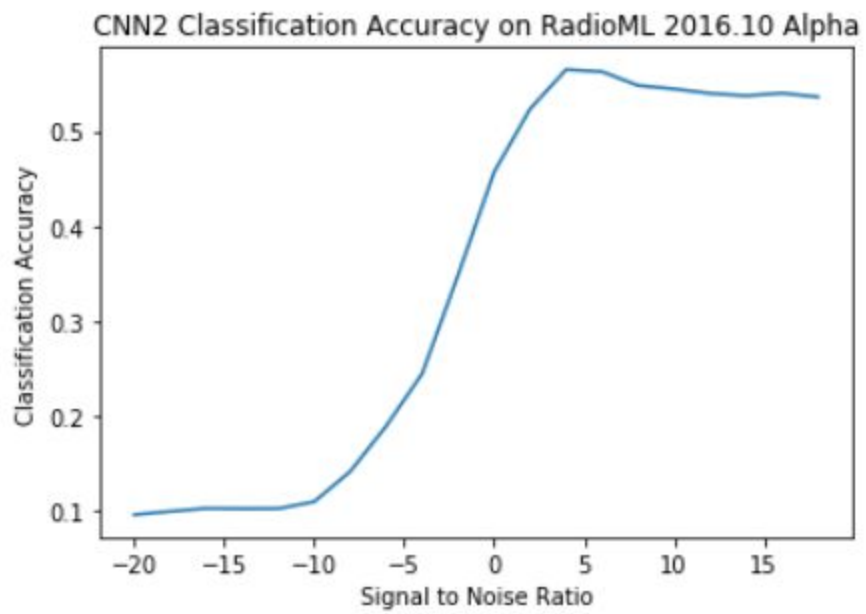
```

Finally we evaluate our model and got the accuracy **0.498** for our test data:

```
600000/600000 [=====] - 2s 4us/step
['loss', 'acc']
score :
[1.1189251502752304, 0.5239433321654796]
600000/600000 [=====] - 2s 4us/step
['loss', 'acc']
[1.1765535290569242, 0.4984966675885953]
Overall Accuracy: 0.10577115757130452
Overall Accuracy: 0.10971535480422291
Overall Accuracy: 0.11837237977805179
Overall Accuracy: 0.14017345879516457
Overall Accuracy: 0.1826980842400374
Overall Accuracy: 0.26486558512941805
Overall Accuracy: 0.3387979955530481
Overall Accuracy: 0.44136018973177005
Overall Accuracy: 0.5679349636836143
Overall Accuracy: 0.6557349591317164
Overall Accuracy: 0.6887904119246652
Overall Accuracy: 0.6995006657789614
Overall Accuracy: 0.7036014765056035
Overall Accuracy: 0.7099585749792875
Overall Accuracy: 0.7083679091844525
Overall Accuracy: 0.7060857998004656
Overall Accuracy: 0.7055122828040743
Overall Accuracy: 0.7034997512025212
Overall Accuracy: 0.7071421423091037
Overall Accuracy: 0.7074736666109346
Overall Accuracy: 0.7074736666109346
```

When we applied the **differentiation vector** on the fully connected network , We got an accuracy of 0.34643000036478044 of training set and 0.34461500112852084 for test set.

```
['loss', 'acc']
score :
[1.5576589628458024, 0.34643000036478044]
600000/600000 [=====] - 4s 7us/step
['loss', 'acc']
[1.5708337266631425, 0.34461500112852084]
Overall Accuracy: 0.09588537839823659
Overall Accuracy: 0.09939195509822264
Overall Accuracy: 0.10254274002732695
Overall Accuracy: 0.10223353313464822
Overall Accuracy: 0.10232961751551965
Overall Accuracy: 0.10953063060061961
Overall Accuracy: 0.14137324527926193
Overall Accuracy: 0.18919731436015633
Overall Accuracy: 0.24441927100686348
Overall Accuracy: 0.3483518692214927
Overall Accuracy: 0.45681001779299696
Overall Accuracy: 0.523335552596538
Overall Accuracy: 0.5647301386718101
Overall Accuracy: 0.5625186412593206
Overall Accuracy: 0.5480134098980981
Overall Accuracy: 0.5443631526438311
Overall Accuracy: 0.5396444977032155
Overall Accuracy: 0.5372035163377011
Overall Accuracy: 0.5398805751075825
Overall Accuracy: 0.5359304464136432
```



Epochs:

Train on 570000 samples, validate on 30000 samples

Epoch 1/100

- 8s - loss: 1.9656 - acc: 0.2171 - val_loss: 1.8337 - val_acc: 0.2636

Epoch 2/100

- 7s - loss: 1.8046 - acc: 0.2702 - val_loss: 1.8528 - val_acc: 0.2568

Epoch 3/100

- 7s - loss: 1.7746 - acc: 0.2819 - val_loss: 1.7656 - val_acc: 0.2919

Epoch 4/100

- 8s - loss: 1.7279 - acc: 0.2971 - val_loss: 1.7820 - val_acc: 0.2891

Epoch 5/100

- 7s - loss: 1.7001 - acc: 0.3044 - val_loss: 1.7180 - val_acc: 0.3011

Epoch 6/100

- 8s - loss: 1.6889 - acc: 0.3073 - val_loss: 1.7349 - val_acc: 0.3020

Epoch 7/100

- 7s - loss: 1.6798 - acc: 0.3107 - val_loss: 1.7052 - val_acc: 0.3129

Epoch 8/100

- 7s - loss: 1.6750 - acc: 0.3119 - val_loss: 1.7117 - val_acc: 0.3080

Epoch 9/100

- 7s - loss: 1.6691 - acc: 0.3134 - val_loss: 1.7605 - val_acc: 0.3043

Epoch 10/100

- 7s - loss: 1.6662 - acc: 0.3140 - val_loss: 1.6986 - val_acc: 0.3100

Epoch 11/100

- 7s - loss: 1.6600 - acc: 0.3154 - val_loss: 1.7158 - val_acc: 0.3125

Epoch 12/100

- 7s - loss: 1.6555 - acc: 0.3173 - val_loss: 1.6977 - val_acc: 0.3113

Epoch 13/100

- 7s - loss: 1.6491 - acc: 0.3184 - val_loss: 1.6779 - val_acc: 0.3157

Epoch 14/100

- 8s - loss: 1.6381 - acc: 0.3221 - val_loss: 1.7042 - val_acc: 0.3123

Epoch 15/100

- 7s - loss: 1.6171 - acc: 0.3262 - val_loss: 1.6676 - val_acc: 0.3239

Epoch 16/100

- 7s - loss: 1.5954 - acc: 0.3314 - val_loss: 1.6156 - val_acc: 0.3319

Epoch 17/100

- 7s - loss: 1.5794 - acc: 0.3348 - val_loss: 1.5996 - val_acc: 0.3327

Epoch 18/100

- 7s - loss: 1.5691 - acc: 0.3377 - val_loss: 1.6051 - val_acc: 0.3342

Epoch 19/100

- 7s - loss: 1.5598 - acc: 0.3404 - val_loss: 1.5800 - val_acc: 0.3418

```

Epoch 19/100
- 7s - loss: 1.5598 - acc: 0.3404 - val_loss: 1.5800 - val_acc: 0.3418
Epoch 20/100
- 7s - loss: 1.5556 - acc: 0.3415 - val_loss: 1.6449 - val_acc: 0.3314
Epoch 21/100
- 7s - loss: 1.5496 - acc: 0.3427 - val_loss: 1.6520 - val_acc: 0.3365
Epoch 22/100
- 7s - loss: 1.5448 - acc: 0.3454 - val_loss: 1.6348 - val_acc: 0.3402
Epoch 23/100
- 8s - loss: 1.5418 - acc: 0.3449 - val_loss: 1.7090 - val_acc: 0.3349
Epoch 24/100
- 8s - loss: 1.5398 - acc: 0.3453 - val_loss: 1.6173 - val_acc: 0.3345
600000/600000 [=====] - 4s 7us/step

```

Summary:

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 2, 128, 1)	0
dropout_1 (Dropout)	(None, 2, 128, 1)	0
flatten_1 (Flatten)	(None, 256)	0
dense5 (Dense)	(None, 256)	65792
dropout_2 (Dropout)	(None, 256)	0
dense6 (Dense)	(None, 256)	65792
dropout_3 (Dropout)	(None, 256)	0
dense7 (Dense)	(None, 180)	46260
dropout_4 (Dropout)	(None, 180)	0
dense8 (Dense)	(None, 180)	32580
dropout_5 (Dropout)	(None, 180)	0
dense11 (Dense)	(None, 10)	1810
activation_1 (Activation)	(None, 10)	0
reshape_2 (Reshape)	(None, 10)	0
Total params: 212,234		
Trainable params: 212,234		
Non-trainable params: 0		

—

Convolutional Neural Network:

For the convolutional Neural Network we also used the sequential model with the same reshaping and input shape as the fully connected

However this time we were given a particular structure to follow for our layers and neurons which suggested the following:

Two layers as conv2D layers with 64 neurons and 16 neurons respectively with flatten() and dropout() layers in between them

And a hidden dense layer of 128 neurons and ending with a 10 neurons output layer and compiled with optimization 'adam' just like the fully connected NN.

In both NN cases we fit on a validation data of 0.05 out of the training data.

We use early stopping criterion on the validation data with a patience of 5, meaning if after 5 epochs the accuracy hasn't changed then the model stops iterating.

We then evaluate the model on the test data and print the scores/accuracies.

For the CNN we tried the following tunings on training data on the batch size and we came up with the following results:

batch size	epcoh	drop out rate	loss	accuracy
200	100	0.1		0.5751500003
400				0.5857566661
600				0.6002550018

270	0.5927233325
650	0.5827299986
700	0.5834616661

Here we have the summary of the model, the epochs it took & the final accuracy on the Test data..

Layer (type)	Output Shape	Param #
=====		
reshape_17 (Reshape)	(None, 2, 128, 1)	0
zero_padding2d_17 (ZeroPaddi	(None, 2, 132, 1)	0
conv1 (Conv2D)	(None, 2, 130, 64)	256
dropout_25 (Dropout)	(None, 2, 130, 64)	0
zero_padding2d_18 (ZeroPaddi	(None, 2, 134, 64)	0
conv2 (Conv2D)	(None, 1, 132, 16)	6160
dropout_26 (Dropout)	(None, 1, 132, 16)	0
flatten_9 (Flatten)	(None, 2112)	0

dense1 (Dense)	(None, 128)	270464
----------------	-------------	--------

dropout_27 (Dropout)	(None, 128)	0
----------------------	-------------	---

dense2 (Dense)	(None, 10)	1290
----------------	------------	------

activation_9 (Activation)	(None, 10)	0
---------------------------	------------	---

reshape_18 (Reshape)	(None, 10)	0
----------------------	------------	---

=====

Total params: 278,170

Trainable params: 278,170

Non-trainable params: 0

Train on 570000 samples, validate on 30000 samples

Epoch 1/100

- 16s - loss: 1.9540 - acc: 0.2225 - val_loss: 1.7173 - val_acc: 0.3034

Epoch 2/100

- 14s - loss: 1.5118 - acc: 0.3865 - val_loss: 1.3109 - val_acc: 0.4606

Epoch 3/100


- 14s - loss: 1.2890 - acc: 0.4643 - val_loss: 1.2048 - val_acc: 0.4931

Epoch 4/100

- 15s - loss: 1.2220 - acc: 0.4867 - val_loss: 1.1738 - val_acc: 0.5078

Epoch 5/100

- 14s - loss: 1.1987 - acc: 0.4960 - val_loss: 1.1639 - val_acc: 0.5144



Epoch 6/100

- 14s - loss: 1.1850 - acc: 0.5014 - val_loss: 1.1500 - val_acc: 0.5162

Epoch 7/100

- 14s - loss: 1.1734 - acc: 0.5067 - val_loss: 1.1495 - val_acc: 0.5069

Epoch 8/100

- 14s - loss: 1.1630 - acc: 0.5117 - val_loss: 1.1464 - val_acc: 0.5222

Epoch 9/100

- 15s - loss: 1.1515 - acc: 0.5176 - val_loss: 1.1249 - val_acc: 0.5299

Epoch 10/100

- 15s - loss: 1.1417 - acc: 0.5214 - val_loss: 1.1134 - val_acc: 0.5335

Epoch 11/100

- 14s - loss: 1.1339 - acc: 0.5246 - val_loss: 1.1075 - val_acc: 0.5383

Epoch 12/100

- 14s - loss: 1.1279 - acc: 0.5268 - val_loss: 1.1033 - val_acc: 0.5343

Epoch 13/100

- 14s - loss: 1.1218 - acc: 0.5295 - val_loss: 1.0950 - val_acc: 0.5381

Epoch 14/100

- 14s - loss: 1.1172 - acc: 0.5314 - val_loss: 1.0950 - val_acc: 0.5391

Epoch 15/100

- 15s - loss: 1.1115 - acc: 0.5346 - val_loss: 1.0909 - val_acc: 0.5420

Epoch 16/100

- 14s - loss: 1.1046 - acc: 0.5380 - val_loss: 1.0849 - val_acc: 0.5476

Epoch 17/100

- 14s - loss: 1.0964 - acc: 0.5415 - val_loss: 1.0797 - val_acc: 0.5461

Epoch 18/100

Epoch 18/100

- 14s - loss: 1.0889 - acc: 0.5435 - val_loss: 1.0685 - val_acc: 0.5514

Epoch 19/100

- 14s - loss: 1.0833 - acc: 0.5456 - val_loss: 1.0649 - val_acc: 0.5526

Epoch 20/100

- 14s - loss: 1.0785 - acc: 0.5483 - val_loss: 1.0661 - val_acc: 0.5507

Epoch 21/100

- 15s - loss: 1.0758 - acc: 0.5486 - val_loss: 1.0641 - val_acc: 0.5530

Epoch 22/100

- 15s - loss: 1.0714 - acc: 0.5503 - val_loss: 1.0582 - val_acc: 0.5566

Epoch 23/100

- 14s - loss: 1.0667 - acc: 0.5527 - val_loss: 1.0648 - val_acc: 0.5498

Epoch 24/100

- 14s - loss: 1.0634 - acc: 0.5542 - val_loss: 1.0581 - val_acc: 0.5525

Epoch 25/100

- 14s - loss: 1.0619 - acc: 0.5554 - val_loss: 1.0542 - val_acc: 0.5597

Epoch 26/100

- 14s - loss: 1.0592 - acc: 0.5568 - val_loss: 1.0509 - val_acc: 0.5600

Epoch 27/100

- 14s - loss: 1.0564 - acc: 0.5582 - val_loss: 1.0529 - val_acc: 0.5611

Epoch 28/100


- 14s - loss: 1.0549 - acc: 0.5591 - val_loss: 1.0582 - val_acc: 0.5563

Epoch 29/100

- 14s - loss: 1.0525 - acc: 0.5599 - val_loss: 1.0531 - val_acc: 0.5565

Epoch 30/100

- 14s - loss: 1.0493 - acc: 0.5617 - val_loss: 1.0521 - val_acc: 0.5630



Epoch 31/100

- 14s - loss: 1.0469 - acc: 0.5625 - val_loss: 1.0496 - val_acc: 0.5632

Epoch 32/100

- 15s - loss: 1.0450 - acc: 0.5635 - val_loss: 1.0576 - val_acc: 0.5603

Epoch 33/100

- 14s - loss: 1.0429 - acc: 0.5654 - val_loss: 1.0544 - val_acc: 0.5635

Epoch 34/100

- 14s - loss: 1.0407 - acc: 0.5653 - val_loss: 1.0482 - val_acc: 0.5621

Epoch 35/100

- 14s - loss: 1.0385 - acc: 0.5672 - val_loss: 1.0488 - val_acc: 0.5593

Epoch 36/100

- 14s - loss: 1.0352 - acc: 0.5691 - val_loss: 1.0441 - val_acc: 0.5675

Epoch 37/100

- 14s - loss: 1.0319 - acc: 0.5704 - val_loss: 1.0445 - val_acc: 0.5671

Epoch 38/100

- 15s - loss: 1.0300 - acc: 0.5712 - val_loss: 1.0426 - val_acc: 0.5719

Epoch 39/100

- 14s - loss: 1.0279 - acc: 0.5730 - val_loss: 1.0486 - val_acc: 0.5691

Epoch 40/100

- 14s - loss: 1.0267 - acc: 0.5735 - val_loss: 1.0412 - val_acc: 0.5703

Epoch 41/100

- 14s - loss: 1.0245 - acc: 0.5744 - val_loss: 1.0395 - val_acc: 0.5695

Epoch 42/100


- 14s - loss: 1.0221 - acc: 0.5755 - val_loss: 1.0430 - val_acc: 0.5683

Epoch 43/100

```

- 15s - loss: 1.0198 - acc: 0.5768 - val_loss: 1.0424 - val_acc: 0.5685
Epoch 44/100
- 14s - loss: 1.0172 - acc: 0.5774 - val_loss: 1.0413 - val_acc: 0.5706
Epoch 45/100
- 14s - loss: 1.0150 - acc: 0.5790 - val_loss: 1.0367 - val_acc: 0.5738
Epoch 46/100
- 14s - loss: 1.0125 - acc: 0.5799 - val_loss: 1.0416 - val_acc: 0.5757
Epoch 47/100
- 14s - loss: 1.0117 - acc: 0.5799 - val_loss: 1.0352 - val_acc: 0.5696
Epoch 48/100
- 14s - loss: 1.0091 - acc: 0.5815 - val_loss: 1.0373 - val_acc: 0.5694
Epoch 49/100
- 15s - loss: 1.0086 - acc: 0.5814 - val_loss: 1.0364 - val_acc: 0.5739
Epoch 50/100
- 14s - loss: 1.0066 - acc: 0.5827 - val_loss: 1.0385 - val_acc: 0.5762
Epoch 51/100
- 14s - loss: 1.0060 - acc: 0.5832 - val_loss: 1.0369 - val_acc: 0.5745
Epoch 52/100
- 14s - loss: 1.0050 - acc: 0.5835 - val_loss: 1.0399 - val_acc: 0.5690
600000/600000 [=====] - 4s 7us/step
['loss', 'acc']
[1.0322691246357862, 0.5720166669893079]
Overall Accuracy: 0.1017634092578986
Overall Accuracy: 0.10824535614058532
Overall Accuracy: 0.11520645182790683

```



```

Overall Accuracy:  0.1365234571208519
Overall Accuracy:  0.16964822108003472
Overall Accuracy:  0.2615010493354209
Overall Accuracy:  0.3813759001758869
Overall Accuracy:  0.5072318535591409
Overall Accuracy:  0.6118478043579663
Overall Accuracy:  0.7255460270668632
Overall Accuracy:  0.8016920132943902
Overall Accuracy:  0.8276964047936085
Overall Accuracy:  0.8327292075421503
Overall Accuracy:  0.8354266777133389
Overall Accuracy:  0.8365253759086534
Overall Accuracy:  0.8396740937811773
Overall Accuracy:  0.8393249450768924
Overall Accuracy:  0.8321778072648864
Overall Accuracy:  0.8392100610468025
Overall Accuracy:  0.8310650392910884

```

Results Using the differentiated data:

Layer (type)	Output Shape	Param #
=====		
reshape_15 (Reshape)	(None, 2, 128, 1)	0
<hr/>		
zero_padding2d_15 (ZeroPaddi	(None, 2, 132, 1)	0
<hr/>		

conv1 (Conv2D)	(None, 2, 130, 64)	256
dropout_22 (Dropout)	(None, 2, 130, 64)	0
zero_padding2d_16 (ZeroPaddi	(None, 2, 134, 64)	0
conv2 (Conv2D)	(None, 1, 132, 16)	6160
dropout_23 (Dropout)	(None, 1, 132, 16)	0
flatten_8 (Flatten)	(None, 2112)	0
dense1 (Dense)	(None, 128)	270464
dropout_24 (Dropout)	(None, 128)	0
dense2 (Dense)	(None, 10)	1290
activation_8 (Activation)	(None, 10)	0
reshape_16 (Reshape)	(None, 10)	0
=====		
Total params: 278,170		
Trainable params: 278,170		
Non-trainable params: 0		



Train on 570000 samples, validate on 30000 samples

Epoch 1/100

- 16s - loss: 2.2929 - acc: 0.1187 - val_loss: 2.2913 - val_acc: 0.1171

Epoch 2/100

- 15s - loss: 2.2915 - acc: 0.1207 - val_loss: 2.2928 - val_acc: 0.1186

Epoch 3/100

- 15s - loss: 2.2913 - acc: 0.1212 - val_loss: 2.2921 - val_acc: 0.1183

Epoch 4/100

- 15s - loss: 2.2909 - acc: 0.1231 - val_loss: 2.2906 - val_acc: 0.1181

Epoch 5/100

- 15s - loss: 2.2900 - acc: 0.1238 - val_loss: 2.2908 - val_acc: 0.1193

Epoch 6/100

- 15s - loss: 2.2157 - acc: 0.1466 - val_loss: 1.9632 - val_acc: 0.2169

Epoch 7/100

- 15s - loss: 1.9200 - acc: 0.2281 - val_loss: 1.8267 - val_acc: 0.2728

Epoch 8/100

- 15s - loss: 1.7428 - acc: 0.2912 - val_loss: 1.6673 - val_acc: 0.3092

Epoch 9/100

- 15s - loss: 1.6527 - acc: 0.3158 - val_loss: 1.6081 - val_acc: 0.3300


Epoch 10/100

- 15s - loss: 1.6261 - acc: 0.3238 - val_loss: 1.5991 - val_acc: 0.3287


Epoch 11/100

- 15s - loss: 1.6089 - acc: 0.3305 - val_loss: 1.5802 - val_acc: 0.3384

Epoch 12/100



```
- 15s - loss: 1.5927 - acc: 0.3371 - val_loss: 1.5654 - val_acc: 0.3501
Epoch 13/100
- 15s - loss: 1.5744 - acc: 0.3436 - val_loss: 1.5472 - val_acc: 0.3512
Epoch 14/100
- 15s - loss: 1.5572 - acc: 0.3516 - val_loss: 1.5282 - val_acc: 0.3603
Epoch 15/100
- 15s - loss: 1.5407 - acc: 0.3586 - val_loss: 1.5231 - val_acc: 0.3614
Epoch 16/100
- 15s - loss: 1.5246 - acc: 0.3652 - val_loss: 1.5144 - val_acc: 0.3643
Epoch 17/100
- 15s - loss: 1.5099 - acc: 0.3710 - val_loss: 1.4976 - val_acc: 0.3678
Epoch 18/100
- 15s - loss: 1.4970 - acc: 0.3766 - val_loss: 1.4699 - val_acc: 0.3865
Epoch 19/100
- 15s - loss: 1.4871 - acc: 0.3800 - val_loss: 1.4775 - val_acc: 0.3776
Epoch 20/100
- 15s - loss: 1.4794 - acc: 0.3840 - val_loss: 1.4690 - val_acc: 0.3842
Epoch 21/100
- 15s - loss: 1.4721 - acc: 0.3869 - val_loss: 1.4575 - val_acc: 0.3891
Epoch 22/100
- 15s - loss: 1.4645 - acc: 0.3902 - val_loss: 1.4499 - val_acc: 0.3901
Epoch 23/100
- 15s - loss: 1.4595 - acc: 0.3919 - val_loss: 1.4523 - val_acc: 0.3874
Epoch 24/100
- 15s - loss: 1.4548 - acc: 0.3943 - val_loss: 1.4535 - val_acc: 0.3881
```



Epoch 25/100

- 15s - loss: 1.4498 - acc: 0.3971 - val_loss: 1.4388 - val_acc: 0.3977

Epoch 26/100

- 15s - loss: 1.4453 - acc: 0.3984 - val_loss: 1.4421 - val_acc: 0.3915

Epoch 27/100

- 15s - loss: 1.4415 - acc: 0.4007 - val_loss: 1.4333 - val_acc: 0.3973

Epoch 28/100

- 15s - loss: 1.4369 - acc: 0.4026 - val_loss: 1.4587 - val_acc: 0.3833

Epoch 29/100

- 15s - loss: 1.4339 - acc: 0.4041 - val_loss: 1.4356 - val_acc: 0.3956

Epoch 30/100

- 15s - loss: 1.4313 - acc: 0.4052 - val_loss: 1.4457 - val_acc: 0.3956

Epoch 31/100

- 15s - loss: 1.4259 - acc: 0.4086 - val_loss: 1.4299 - val_acc: 0.3998

Epoch 32/100

- 15s - loss: 1.4221 - acc: 0.4105 - val_loss: 1.4293 - val_acc: 0.3989

Epoch 33/100

- 15s - loss: 1.4201 - acc: 0.4109 - val_loss: 1.4291 - val_acc: 0.4004

Epoch 34/100

- 15s - loss: 1.4169 - acc: 0.4127 - val_loss: 1.4256 - val_acc: 0.4061


Epoch 35/100

- 15s - loss: 1.4144 - acc: 0.4148 - val_loss: 1.4303 - val_acc: 0.4017

Epoch 36/100

- 15s - loss: 1.4128 - acc: 0.4151 - val_loss: 1.4253 - val_acc: 0.4039

Epoch 37/100



```
- 15s - loss: 1.4093 - acc: 0.4177 - val_loss: 1.4275 - val_acc: 0.4001
Epoch 38/100
- 15s - loss: 1.4068 - acc: 0.4191 - val_loss: 1.4248 - val_acc: 0.4007
Epoch 39/100
- 15s - loss: 1.4039 - acc: 0.4199 - val_loss: 1.4200 - val_acc: 0.4034
Epoch 40/100
- 15s - loss: 1.4001 - acc: 0.4227 - val_loss: 1.4196 - val_acc: 0.4069
Epoch 41/100
- 15s - loss: 1.3990 - acc: 0.4220 - val_loss: 1.4302 - val_acc: 0.4031
Epoch 42/100
- 15s - loss: 1.3975 - acc: 0.4234 - val_loss: 1.4188 - val_acc: 0.4064
Epoch 43/100
- 15s - loss: 1.3961 - acc: 0.4233 - val_loss: 1.4171 - val_acc: 0.4080
Epoch 44/100
- 15s - loss: 1.3918 - acc: 0.4250 - val_loss: 1.4143 - val_acc: 0.4091
Epoch 45/100
- 15s - loss: 1.3903 - acc: 0.4268 - val_loss: 1.4197 - val_acc: 0.4087
Epoch 46/100
- 15s - loss: 1.3889 - acc: 0.4275 - val_loss: 1.4123 - val_acc: 0.4114
Epoch 47/100
- 15s - loss: 1.3854 - acc: 0.4290 - val_loss: 1.4094 - val_acc: 0.4141
Epoch 48/100
- 15s - loss: 1.3833 - acc: 0.4302 - val_loss: 1.4107 - val_acc: 0.4112
Epoch 49/100
- 15s - loss: 1.3826 - acc: 0.4307 - val_loss: 1.4105 - val_acc: 0.4093
```

Epoch 50/100

- 15s - loss: 1.3789 - acc: 0.4324 - val_loss: 1.4106 - val_acc: 0.4125

Epoch 51/100

- 15s - loss: 1.3780 - acc: 0.4332 - val_loss: 1.4171 - val_acc: 0.4098

Epoch 52/100

- 15s - loss: 1.3768 - acc: 0.4331 - val_loss: 1.4182 - val_acc: 0.4102

600000/600000 [=====] - 4s 7us/step

['loss', 'acc']

[1.3225805779695512, 0.4632566666007042]

Overall Accuracy: 0.10070530308071063

Overall Accuracy: 0.0992084608221365

Overall Accuracy: 0.0994565398593005

Overall Accuracy: 0.09632677439692072

Overall Accuracy: 0.09930754377788135

Overall Accuracy: 0.10850205129915613

Overall Accuracy: 0.11671744734991797

Overall Accuracy: 0.1232411934936633

Overall Accuracy: 0.1185553258187154

Overall Accuracy: 0.11403741541727477

Overall Accuracy: 0.10876112931519545

Overall Accuracy: 0.10370493991989319

Overall Accuracy: 0.10598416251795917

Overall Accuracy: 0.1057837384744342

Overall Accuracy: 0.10176413483747866

Overall Accuracy: 0.10273972602739725



Overall Accuracy: 0.10220976033113024

Overall Accuracy: 0.10380170825657344

Overall Accuracy: 0.1039869433434367

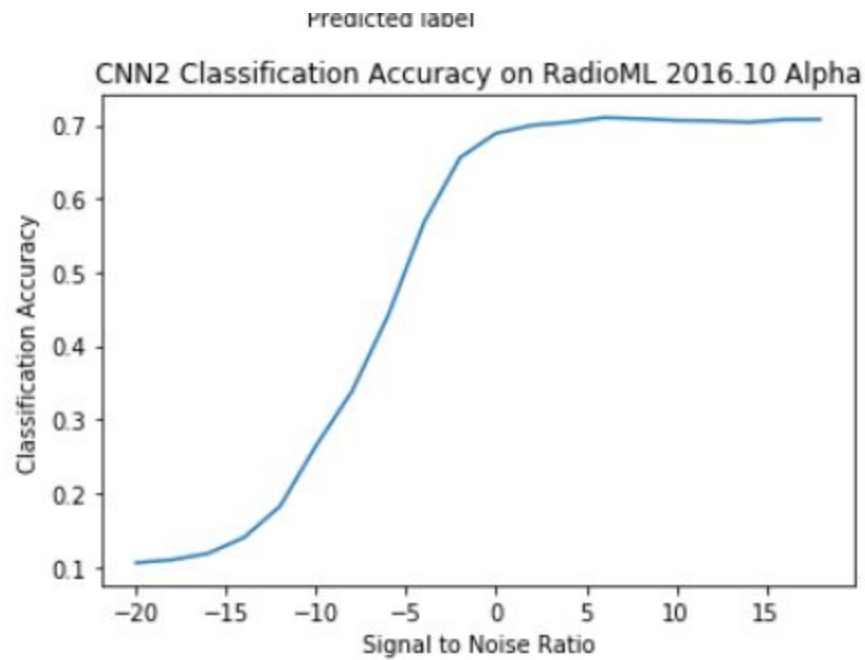
Overall Accuracy: 0.10227612560225952

Step 4: Big Picture:

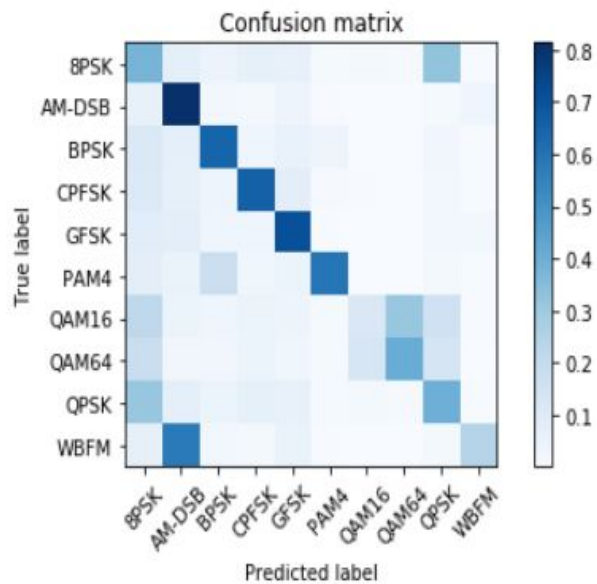
You need to compare the performance of the learned models against important factors.

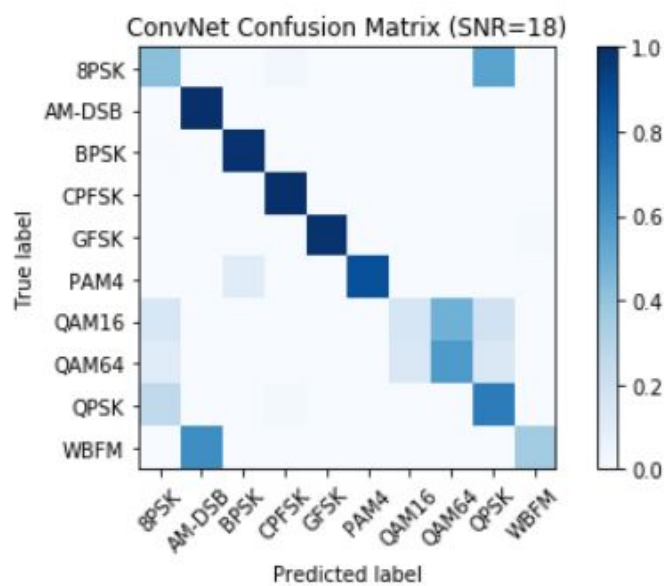
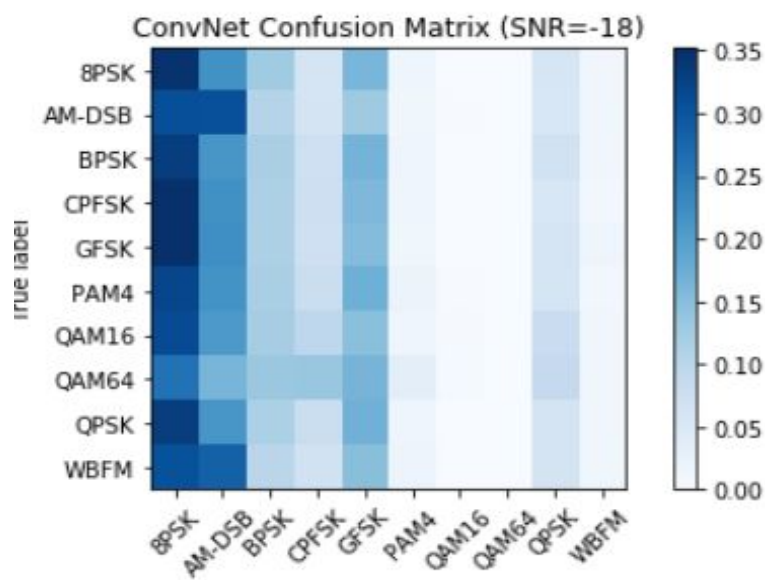
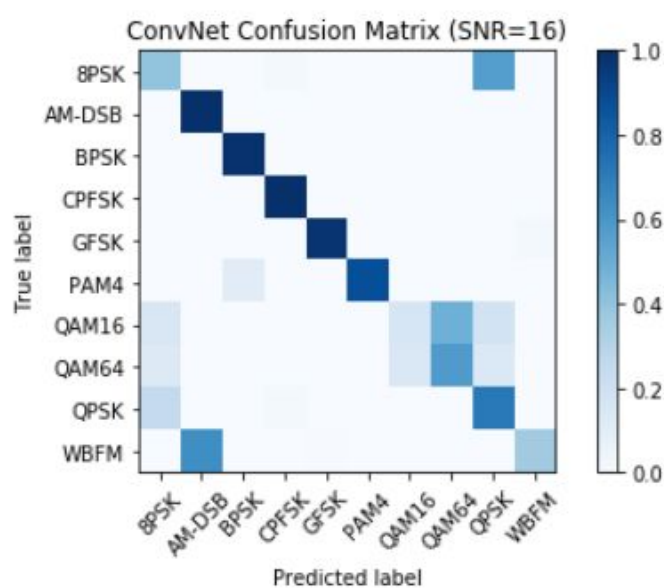
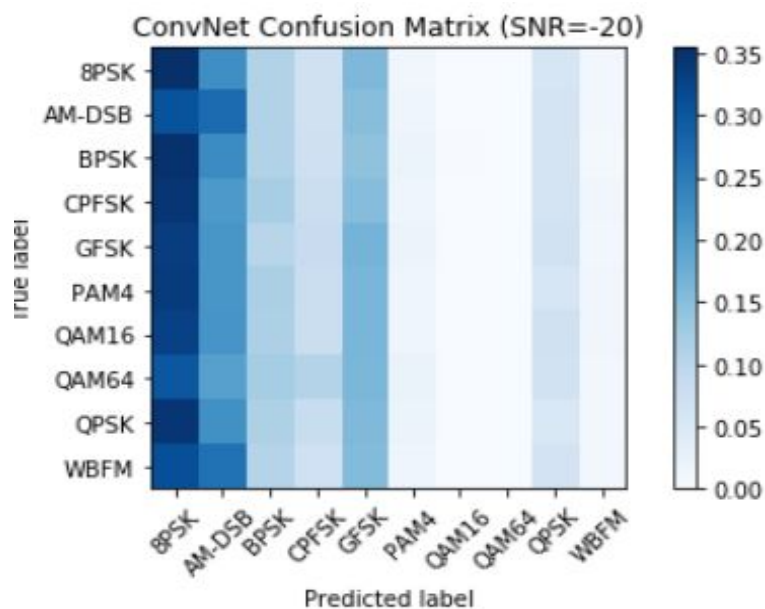
For the **Fully Connected Neural Network**:-

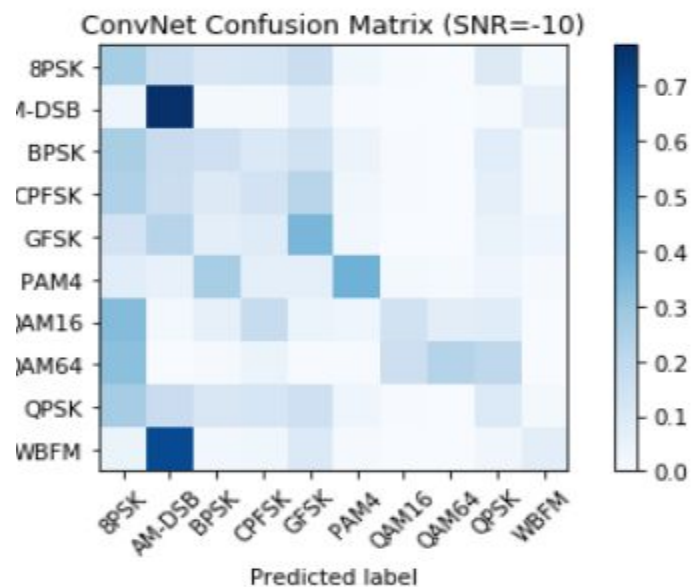
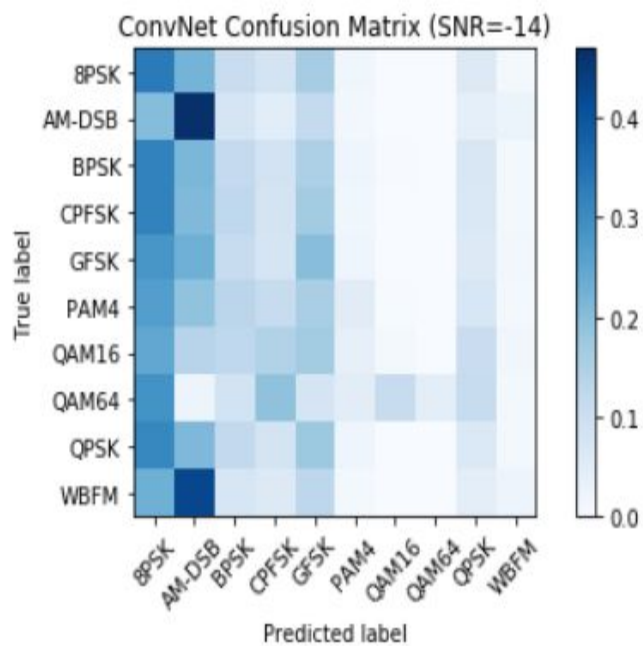
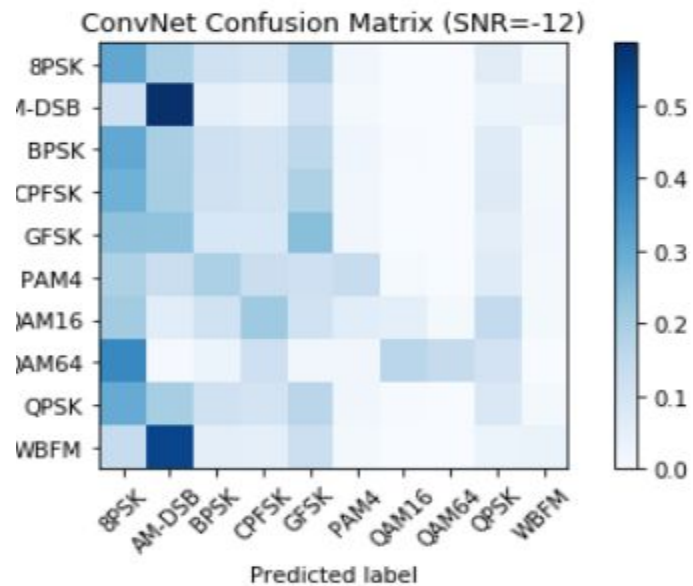
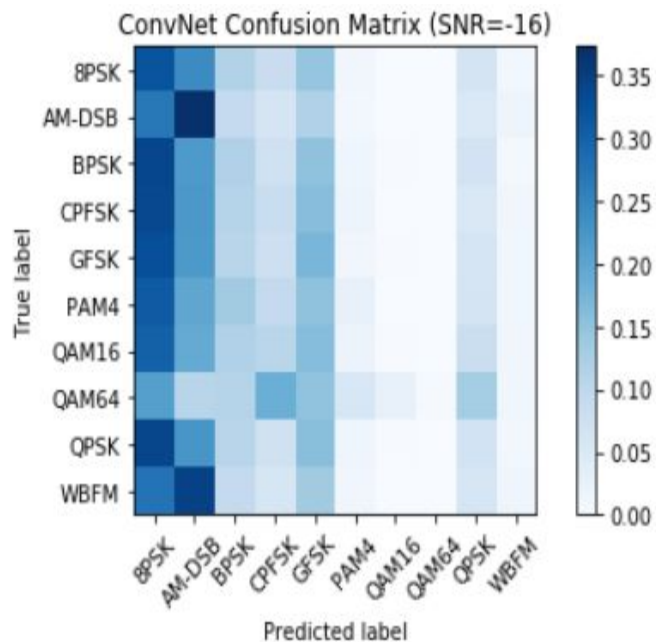
Plots of the accuracy against the SNR as below:

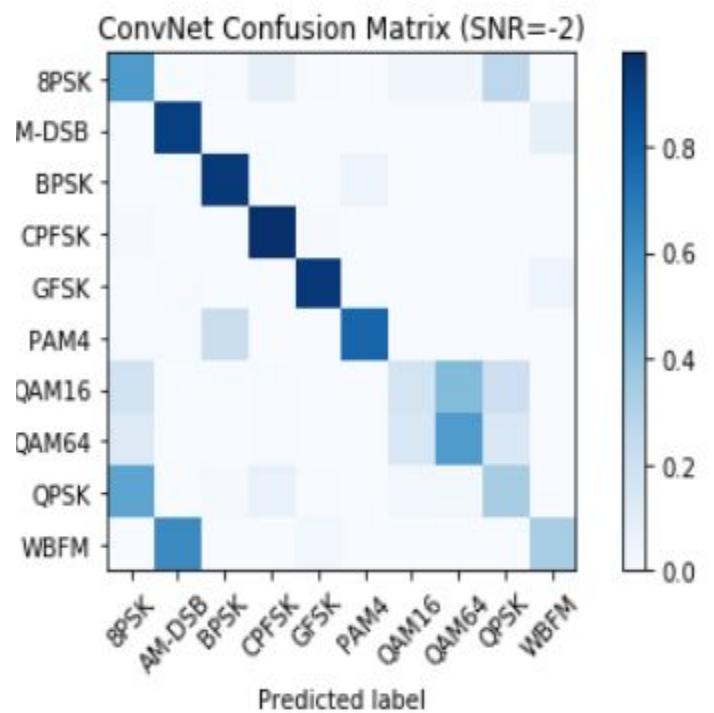
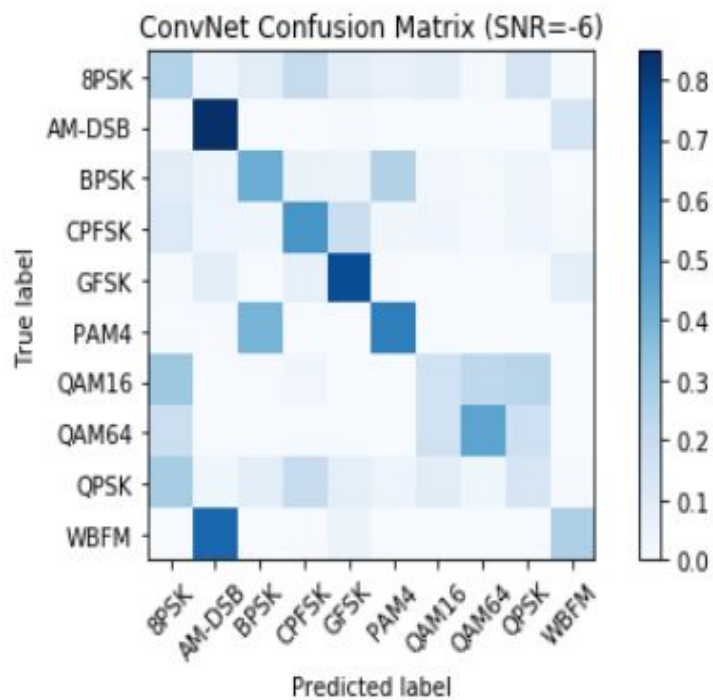
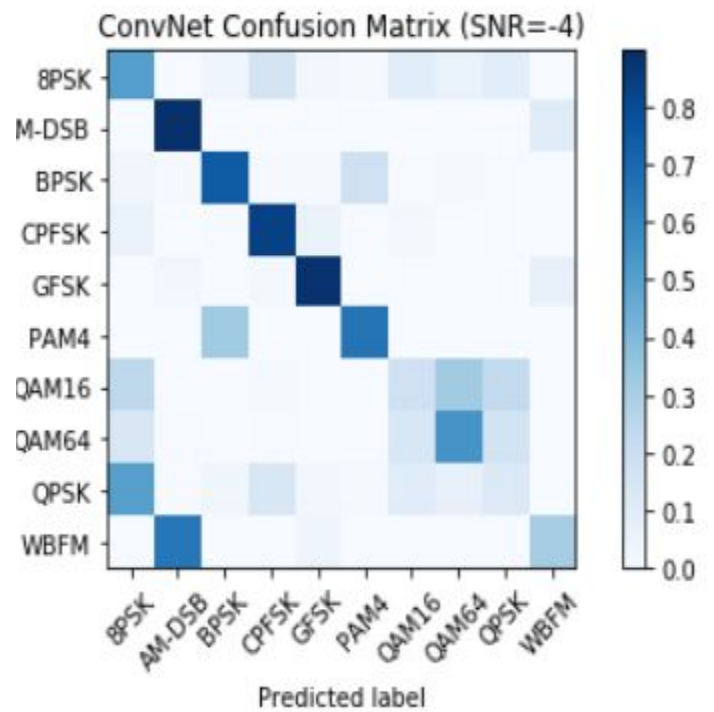
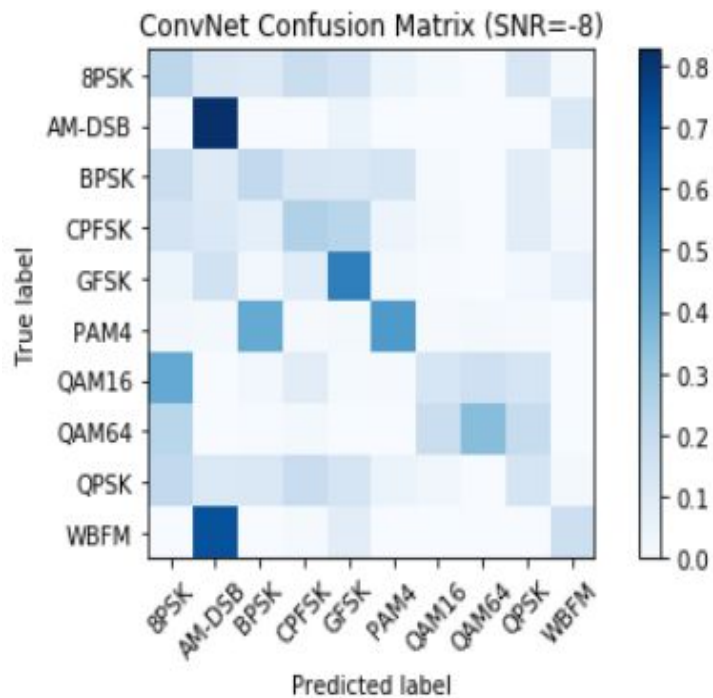


Plot of all confusion matrices:

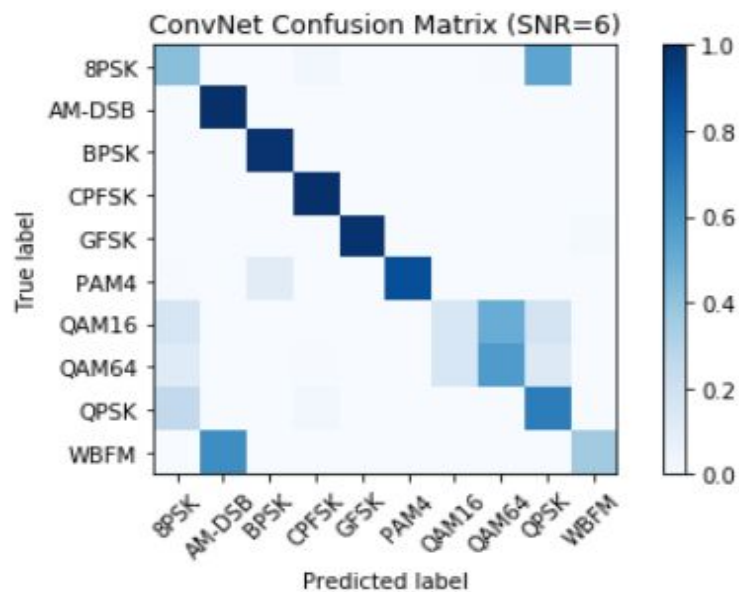
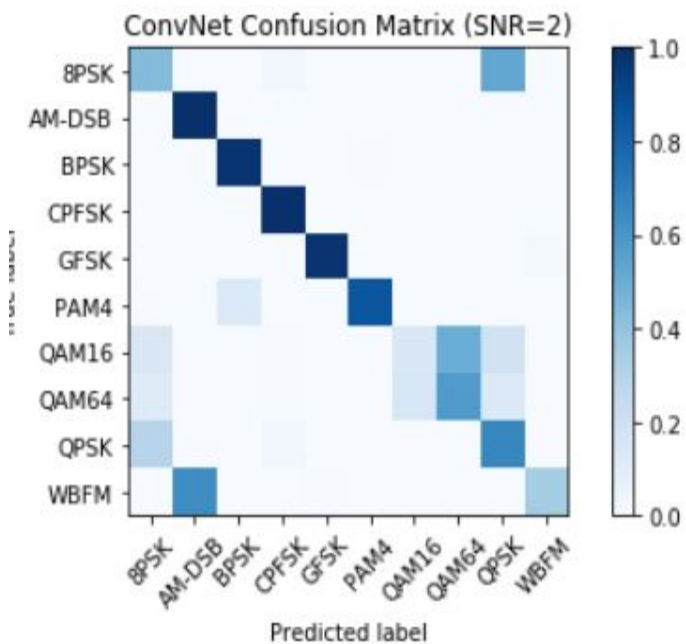
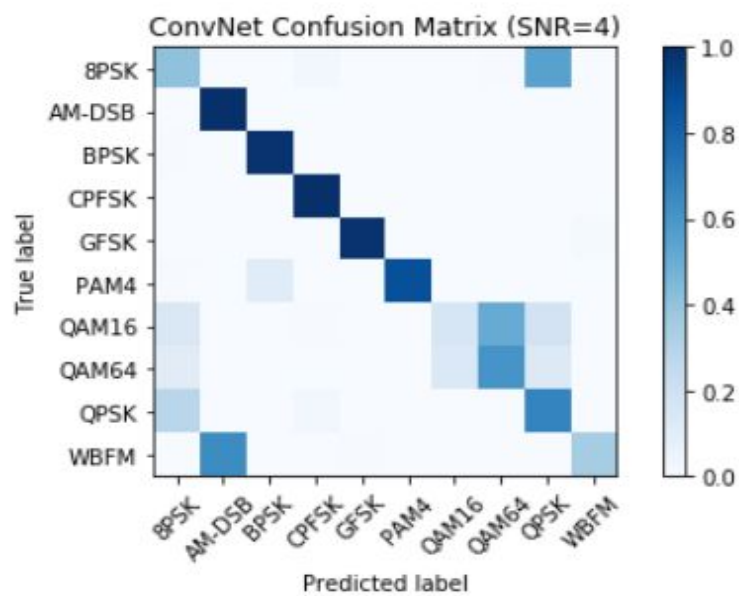
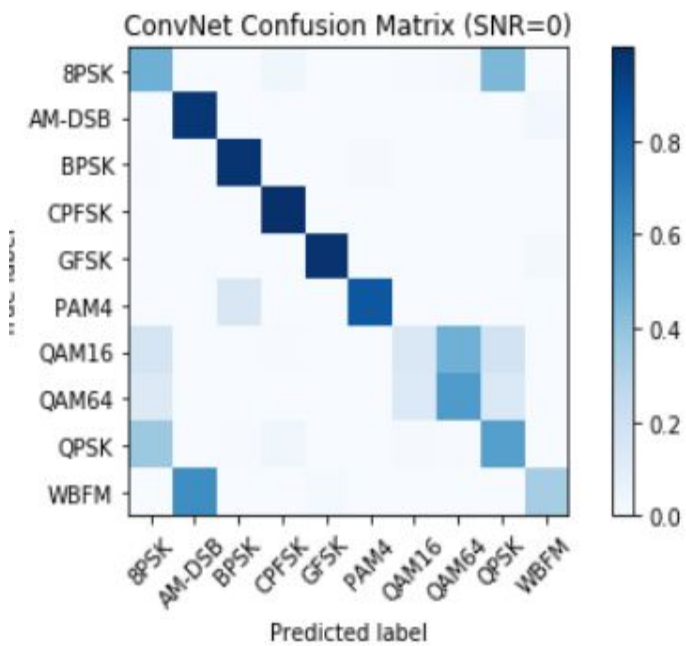


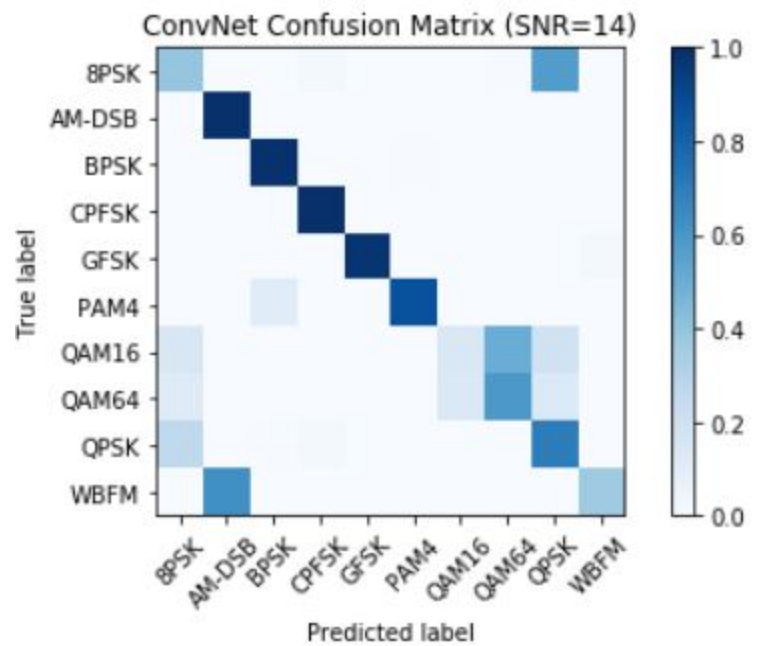
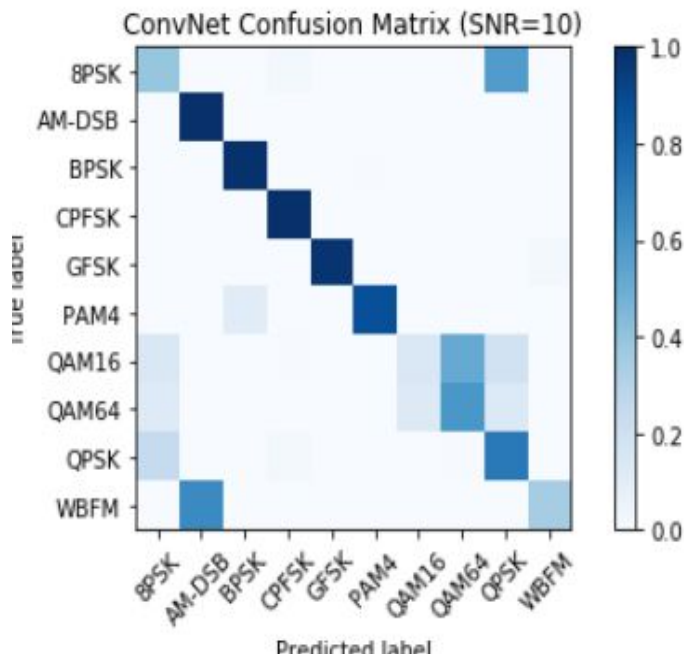
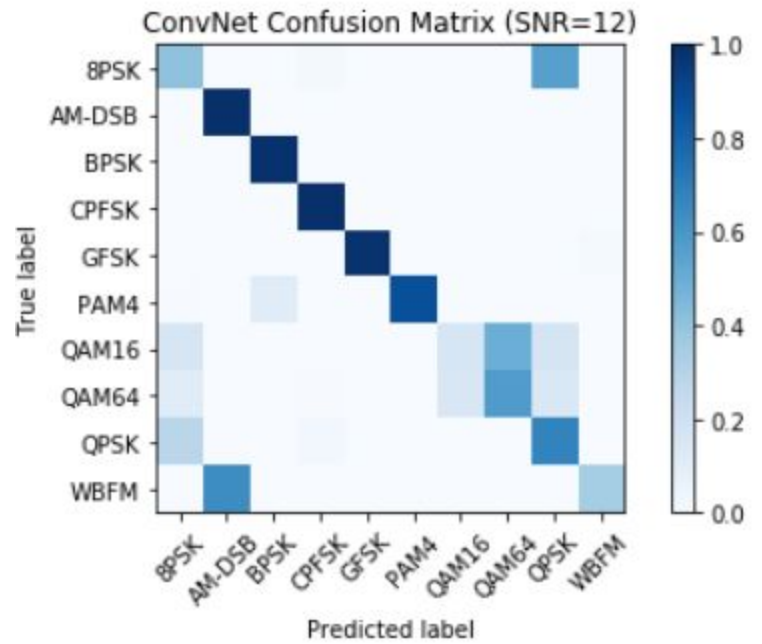
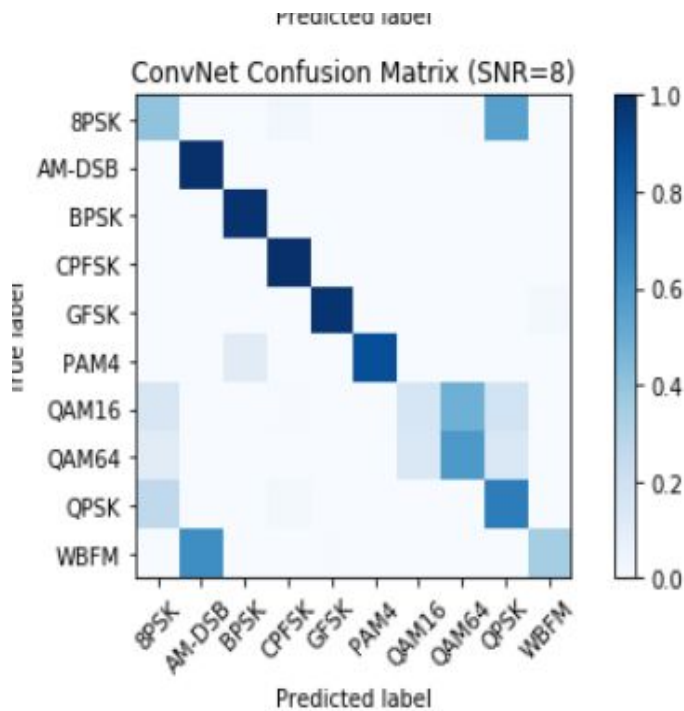






100%

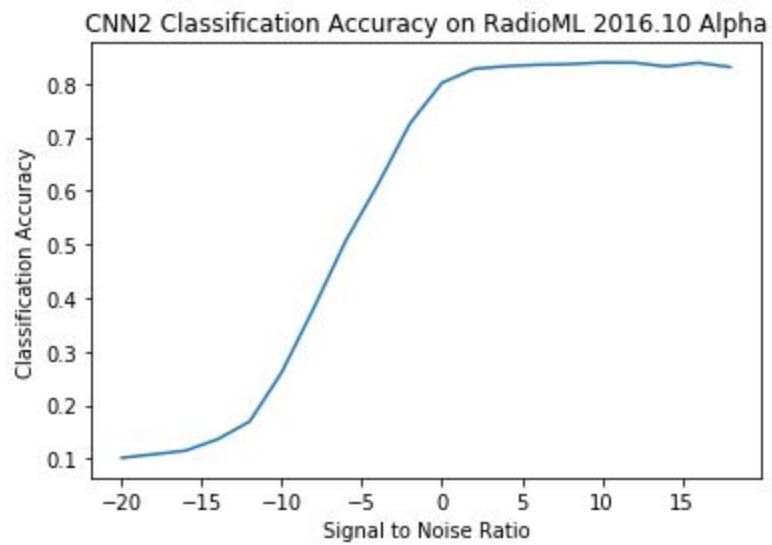




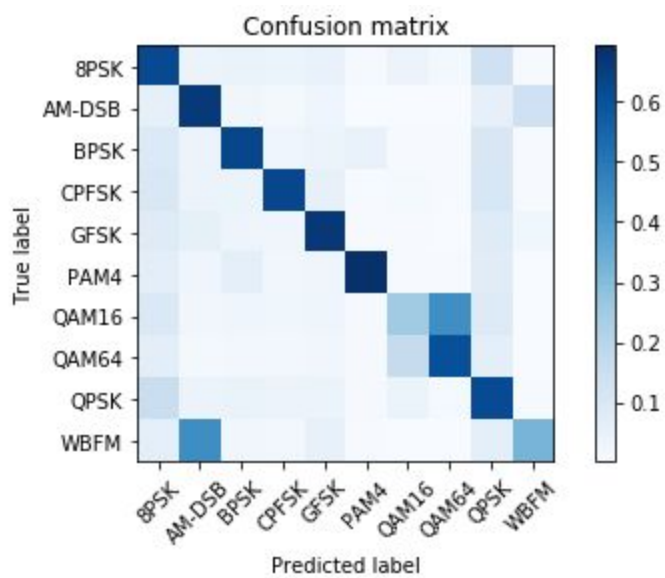
CNN plots

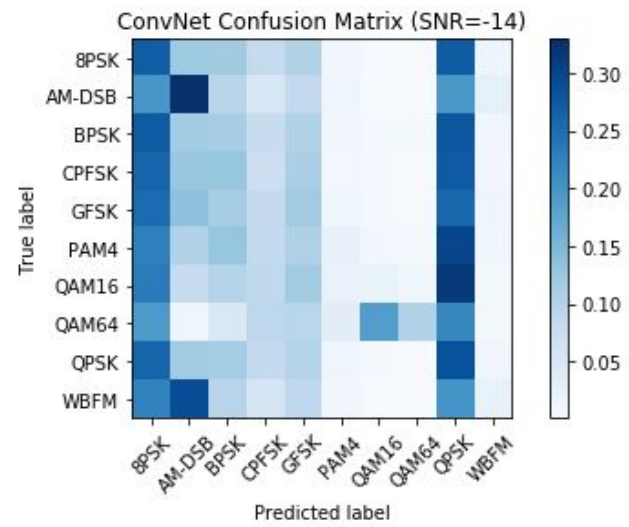
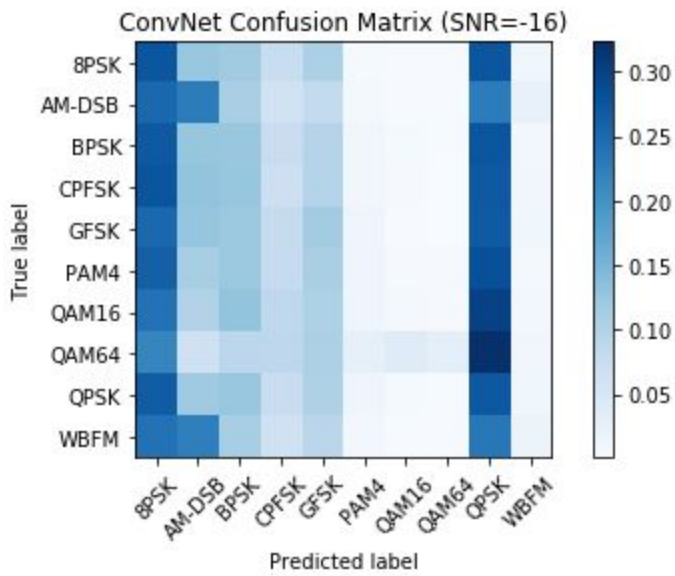
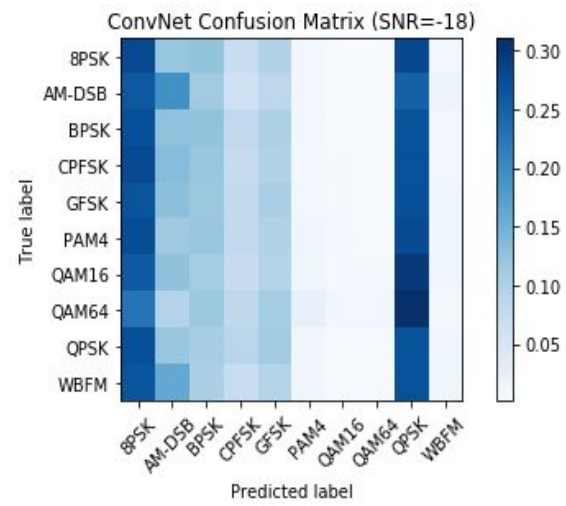
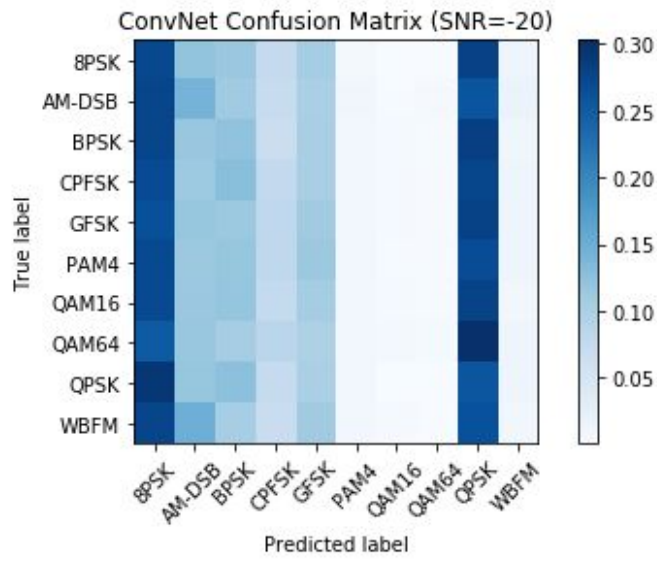
Raw data:

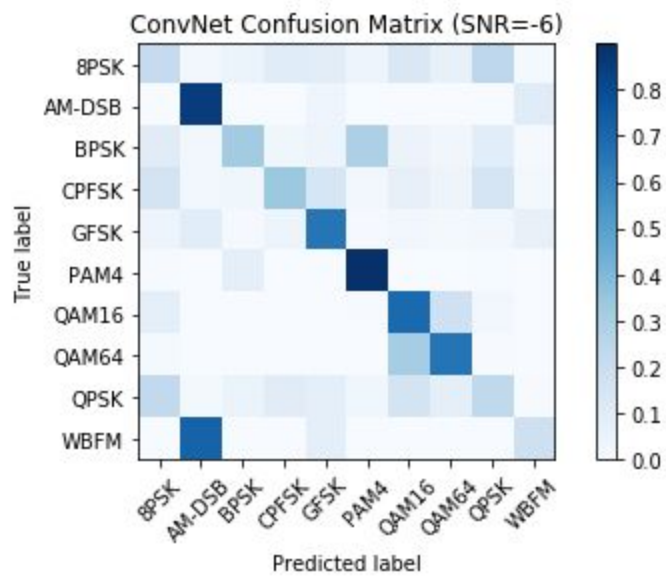
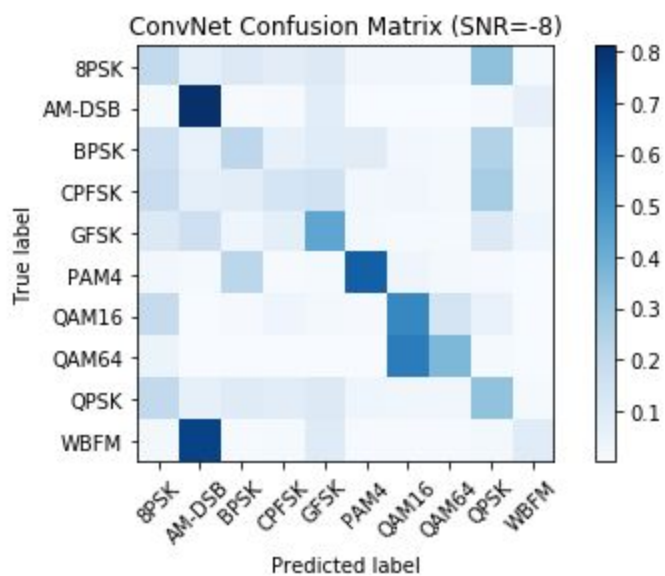
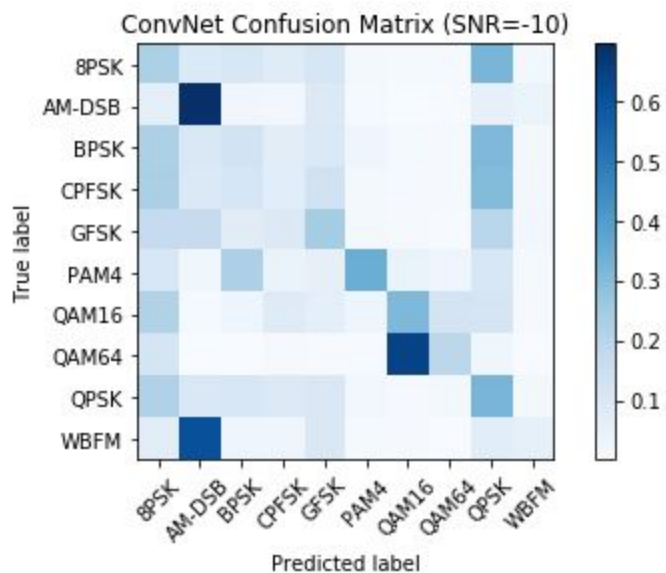
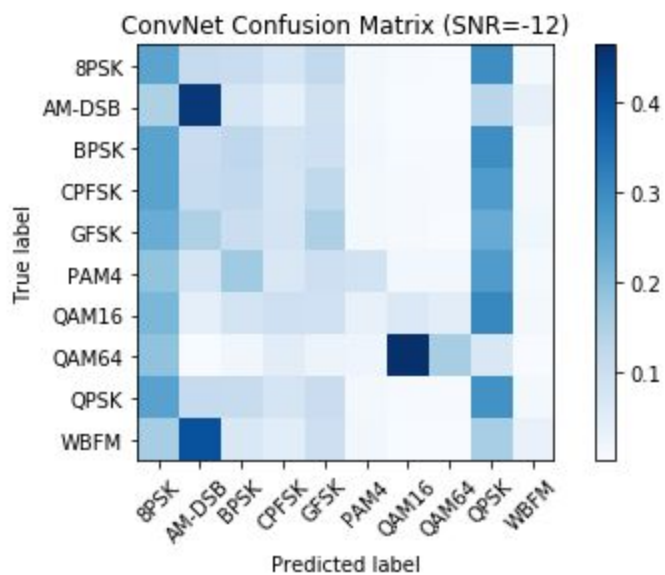
Plots of the accuracy against the SNR

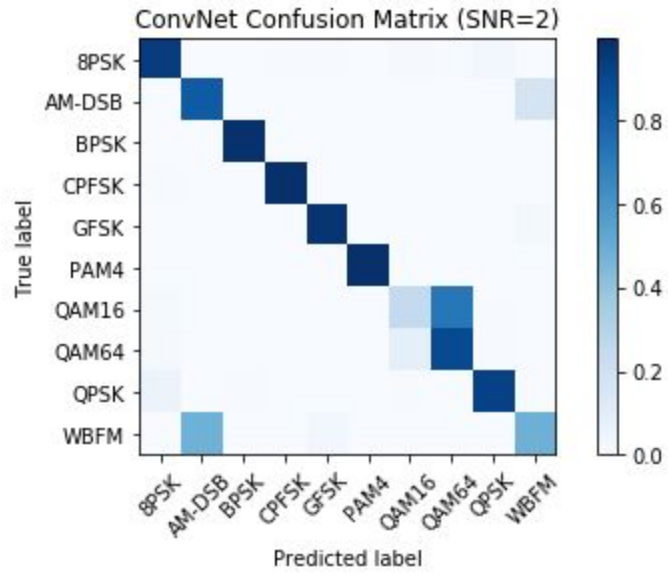
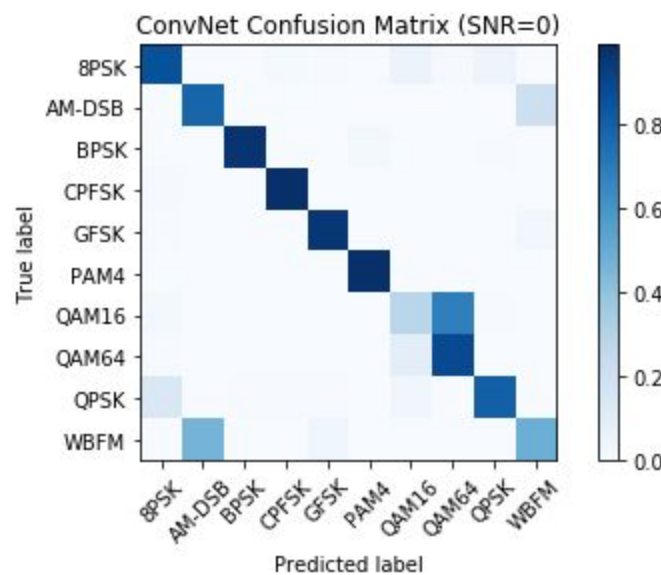
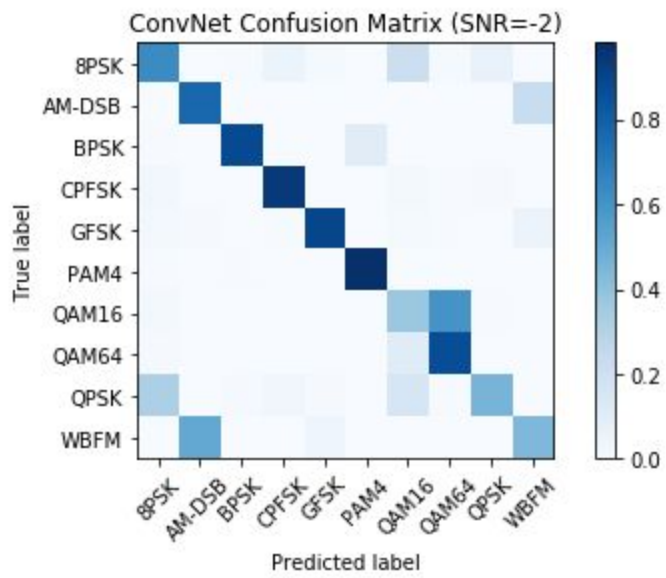
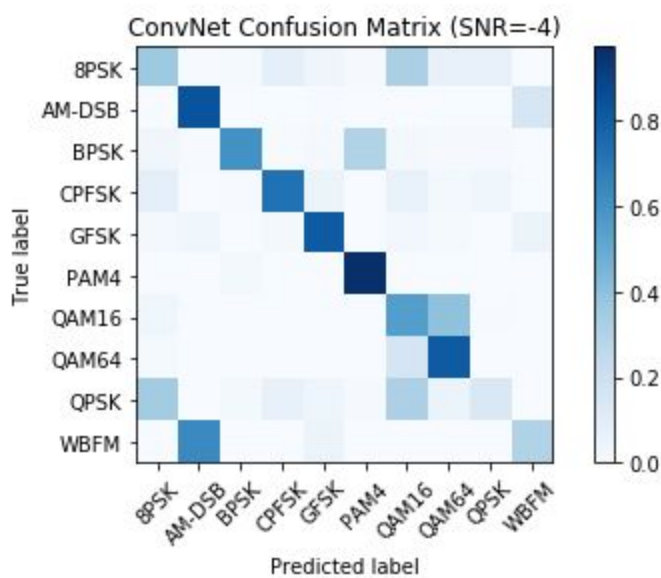


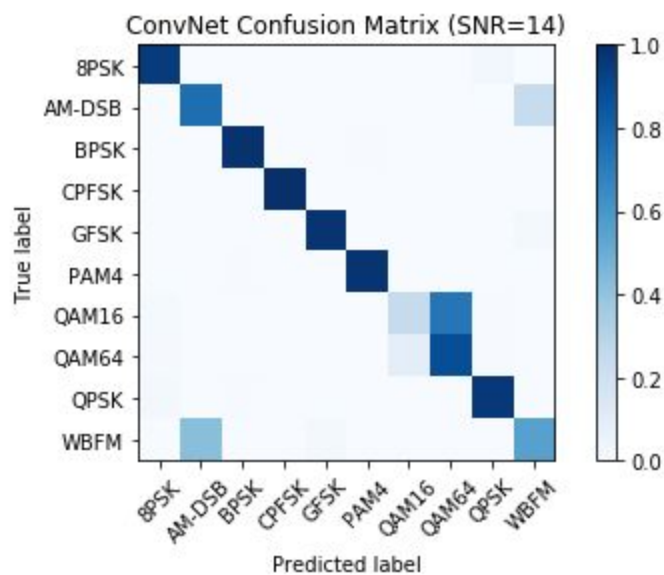
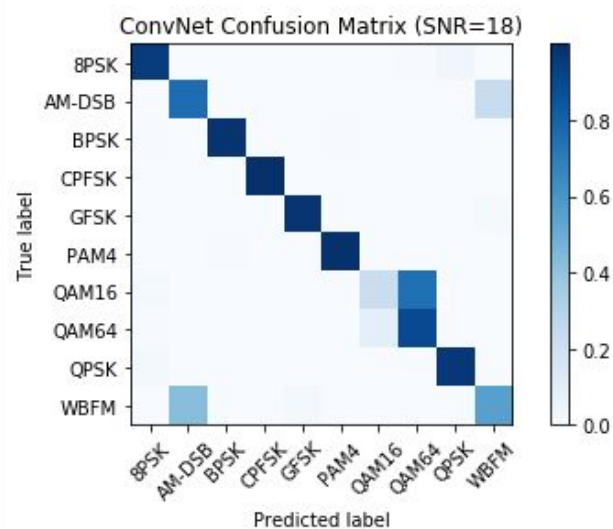
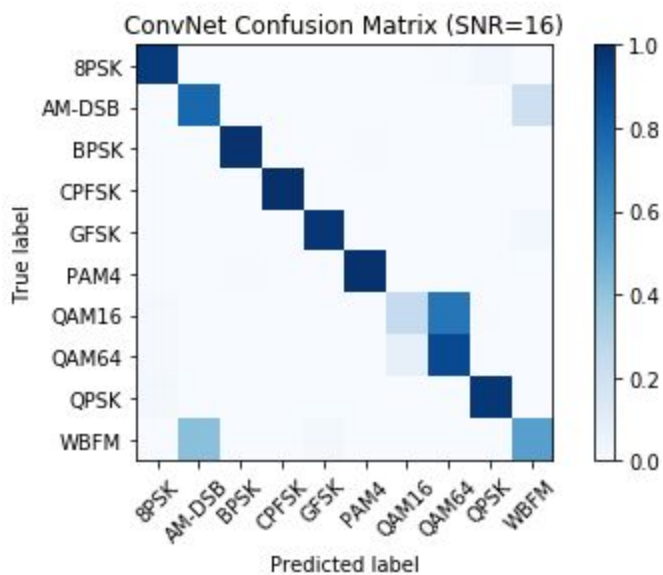
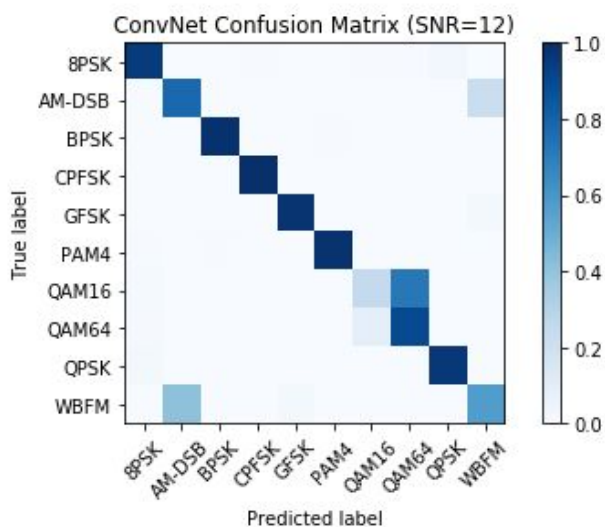
Confusion Matrices:



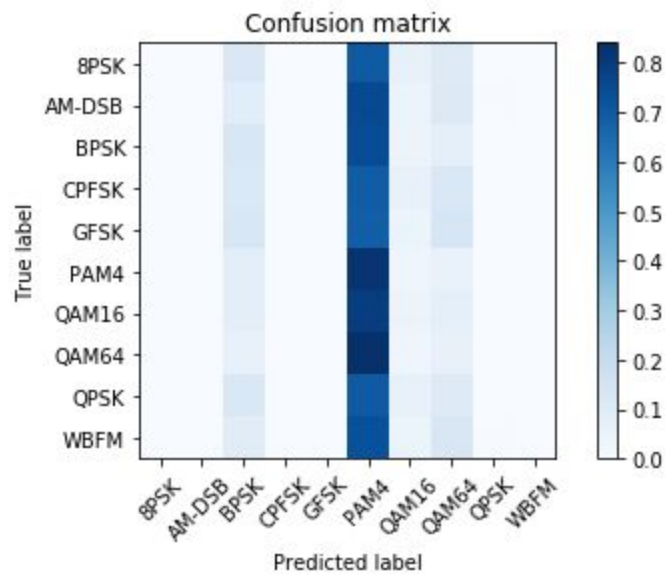
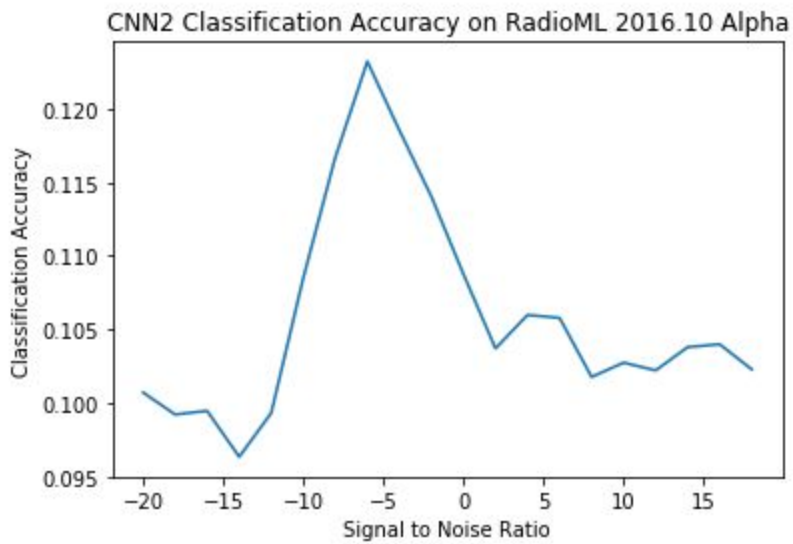


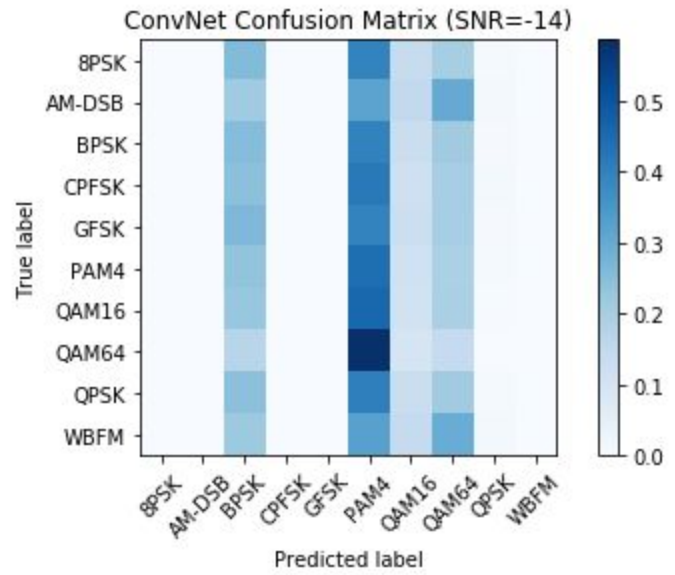
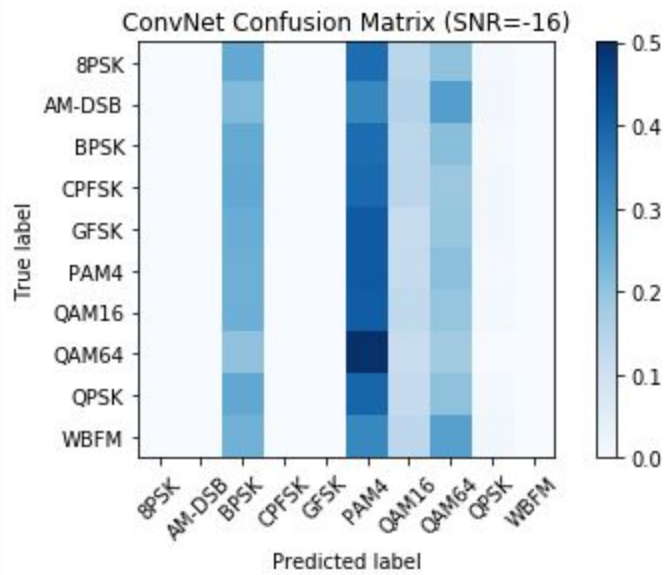
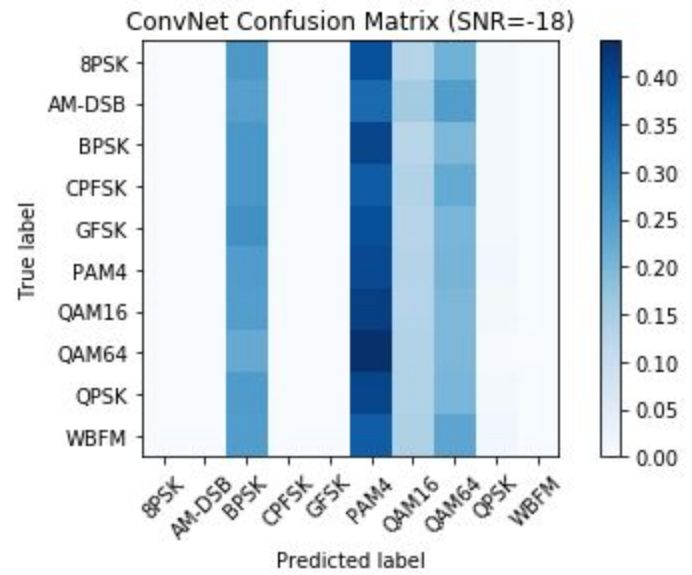
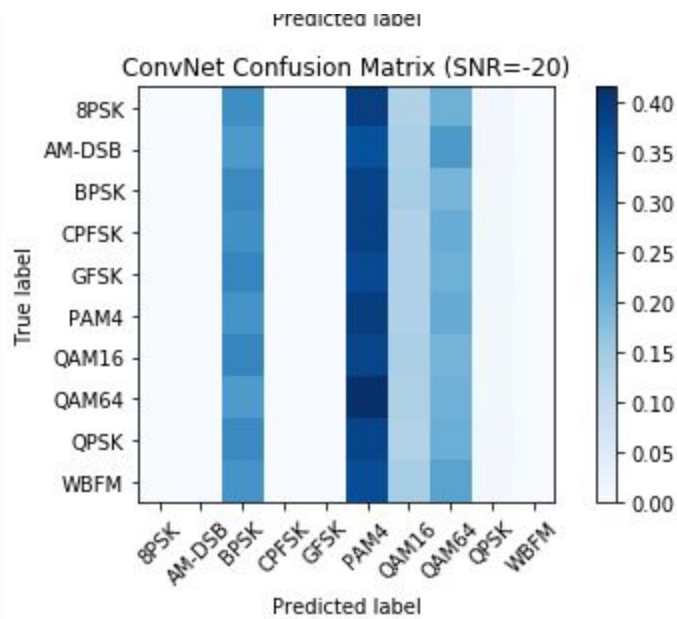


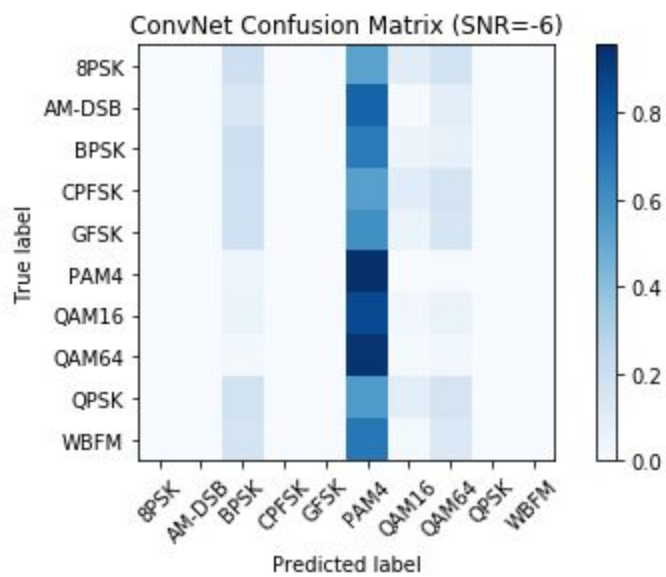
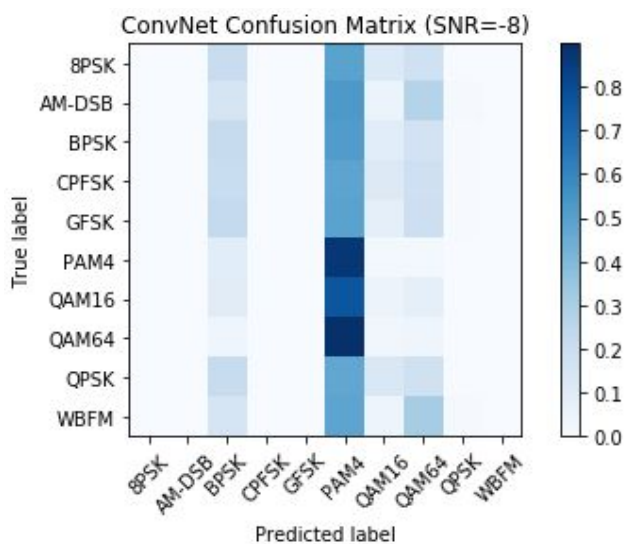
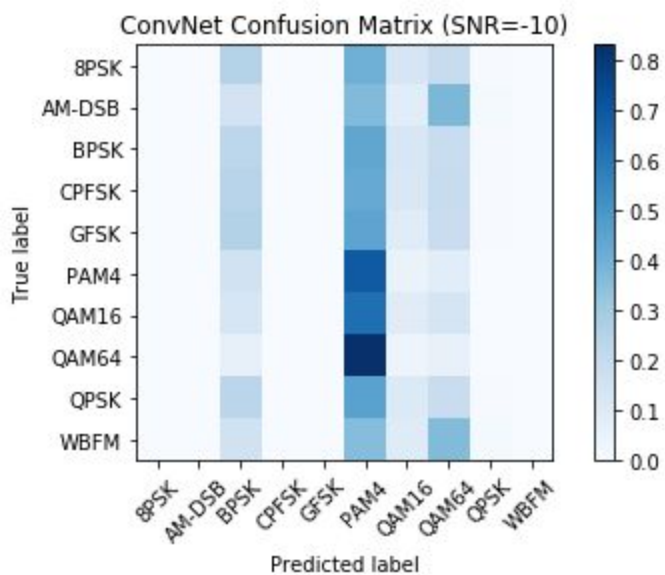
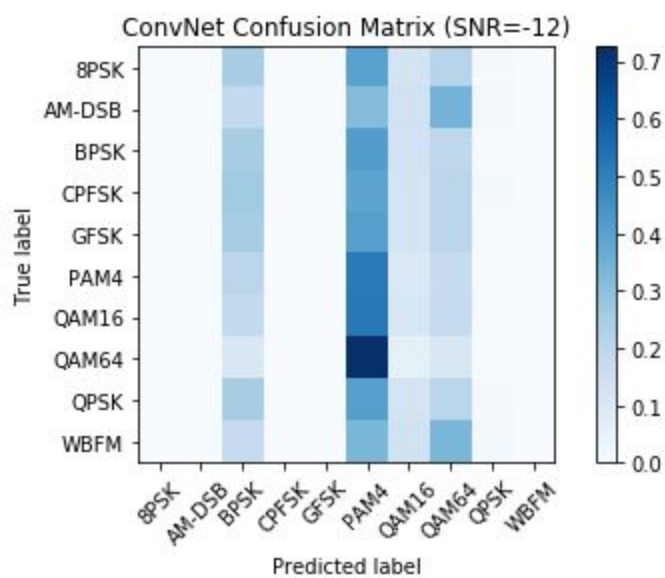


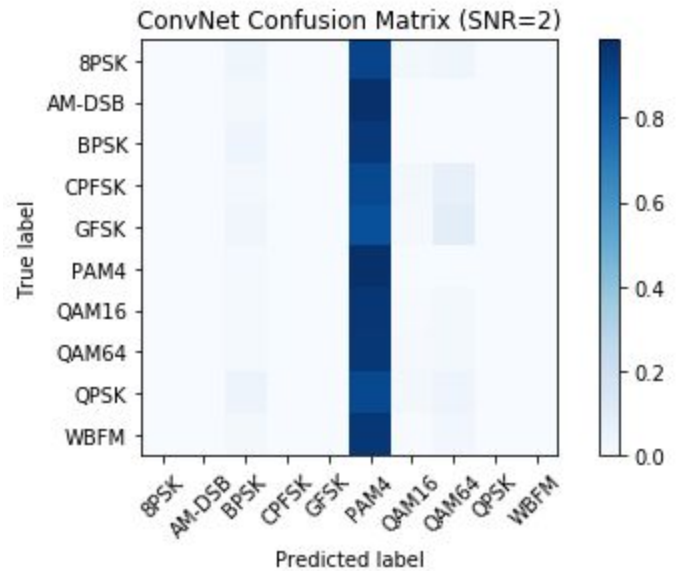
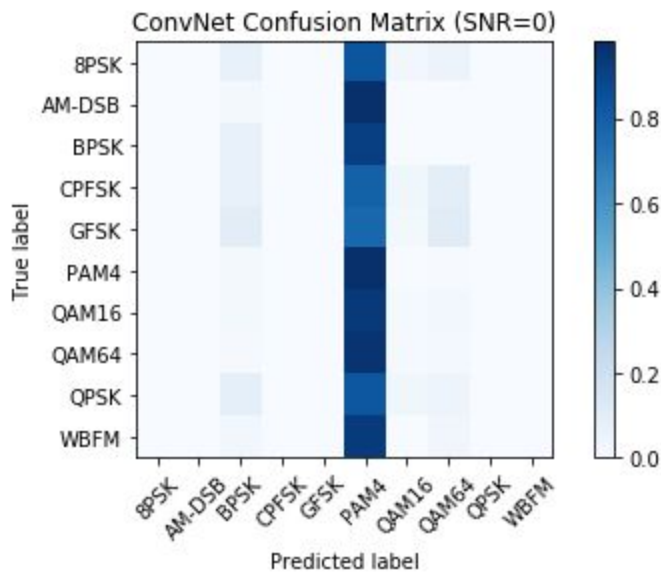
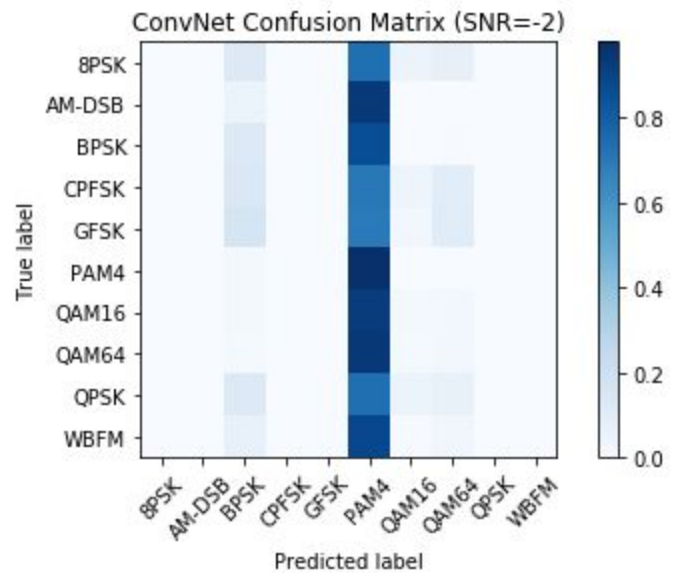
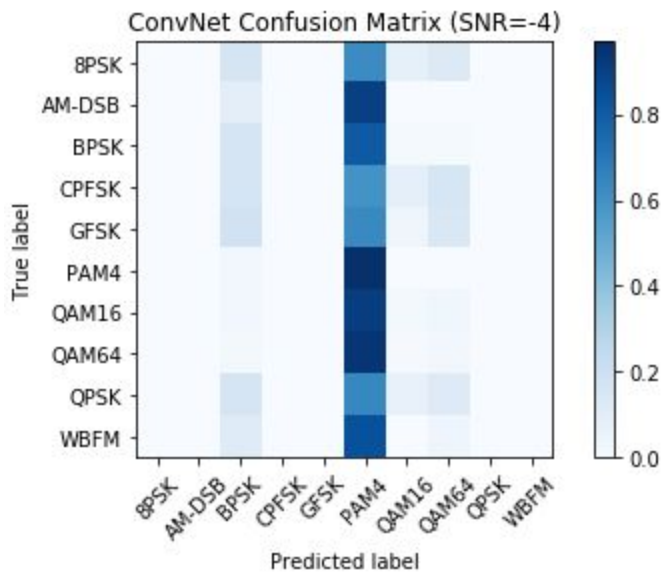


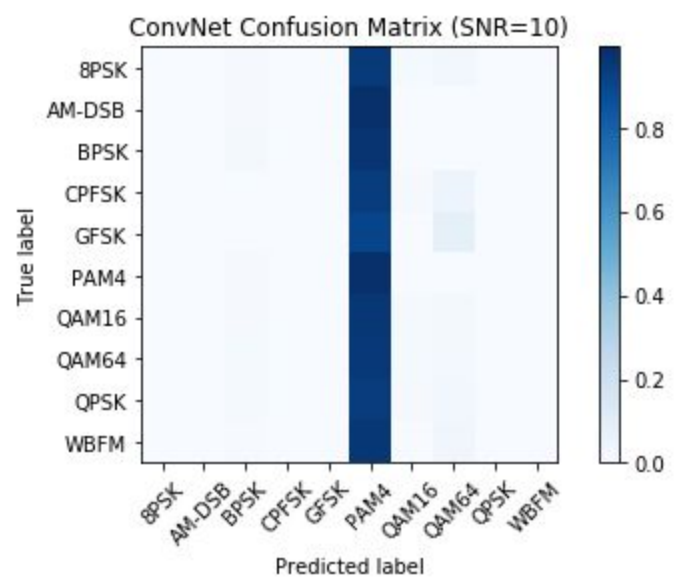
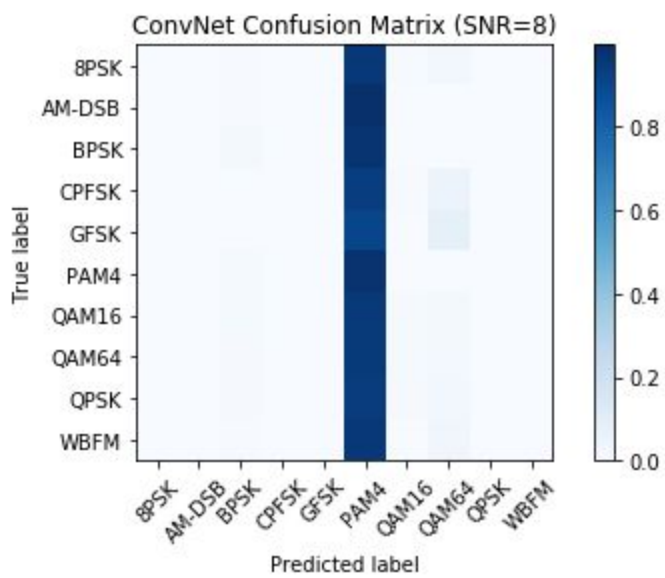
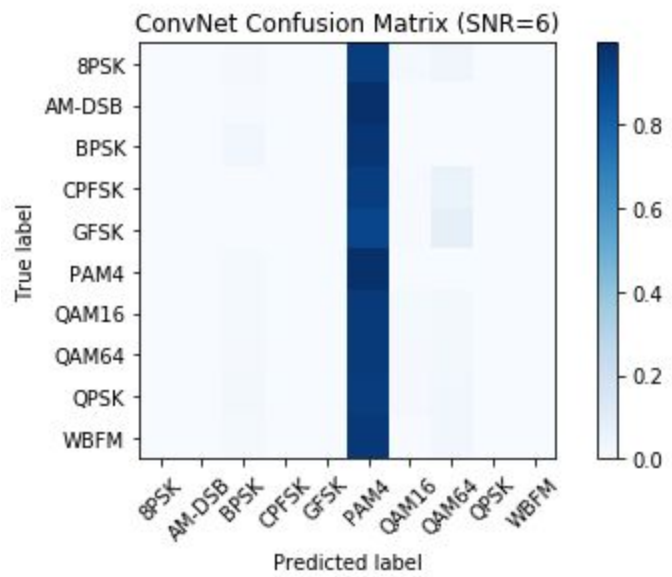
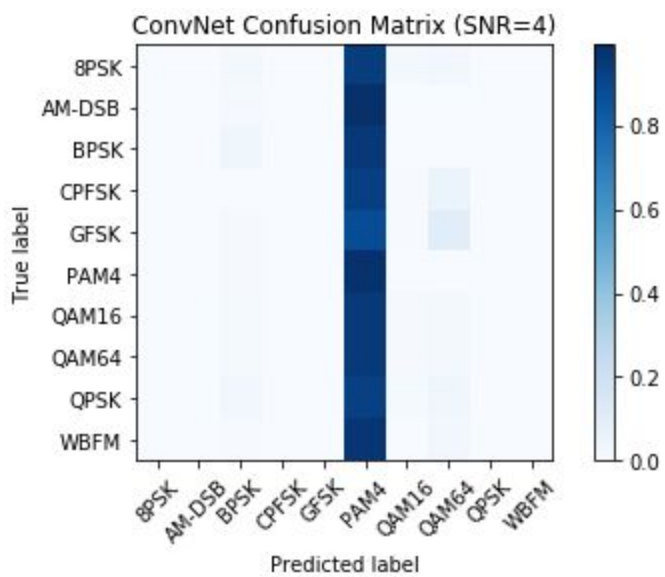
Derivation Plots:

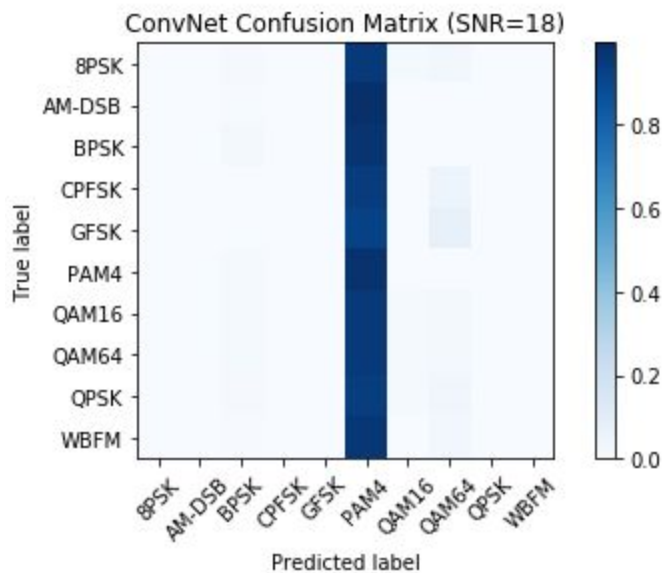
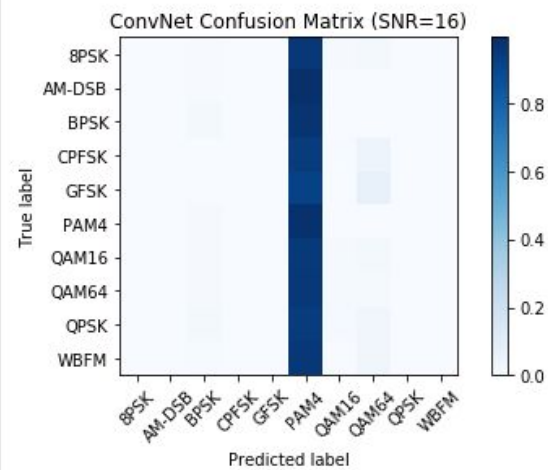
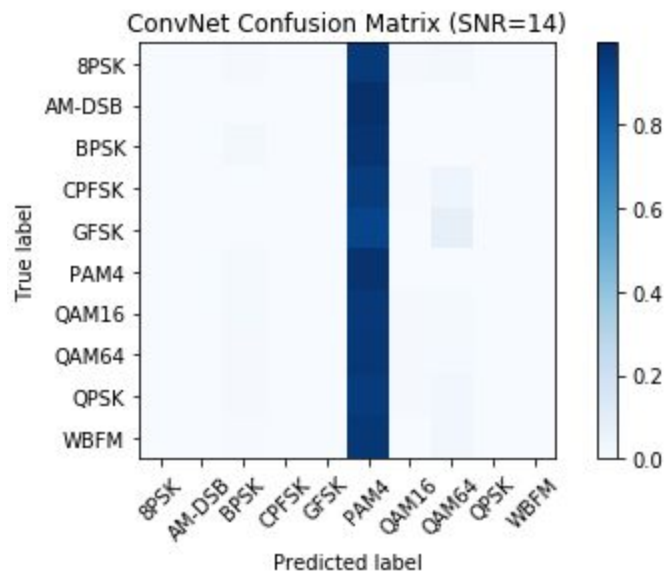
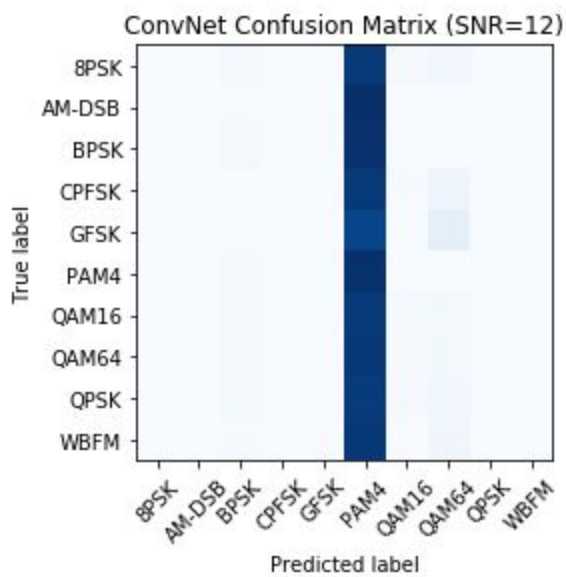














Conclusion:

We notice that the derivation gives a significantly lower accuracy than the raw data.

We notice that the accuracy considerably increases with the SNR increase.

Batch size being in the 600 range gives out the best results.

The epochs never reach 100 it only gets to 50 tops.