[ ]

Assignment #3
Modulation Classification

Name1: Moustafa Ibrahim Tohamy
ID1: 3851

Name2: Zeyad Ezzat
ID2: 4492

Introduction:
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.

Steps:

Data has been extracted using pickle.
<u>The Code for dataset extraction:</u>

```
import pickle
import numpy as np
'''
    the data set has 10 modulations
    for each modulation we have 20 snr
    for each modulation/snr we have 1000 instance
    ************************************************************************
                        for example
    8 PSK modulation has 20 snr values from -20 to 18 with step value = 2
    for each value of the snr for 8 PSK modulation we have 1000 example
    and the same for each modulation
    so for every modulation we have 20(snrs)*6000=120000 example
    ************************************************************************
    the dataset =10 (modulation)* 120000(example for each modulation)=120000 example

'''
a=open("/content/gdrive/My Drive/a/RML2016.10b.dat",'rb')
u = pickle._Unpickler(a)
u.encoding = 'latin1'
Xd = u.load()

# Xd = pickle.load(a)
snrs,mods = map(lambda j: sorted(list(set(map(lambda x: x[j], Xd.keys())))), [1,0])
X = []
lbl = []

j=0
```

```
classes_index=0
mod_to_int={}
for mod in mods:
   j+=1
   print('mod = ',mod)
   mod_to_int[j] = mod

   for snr in snrs:
      '''
      classes are the labels dataset each modulation will have a number from 1 to 11
      mod to int is to map between numbers and clasees
      dataset_snrs is all snr as a one vector to be more easier to deal with

      '''
      classes_index+=1000
      print(Xd[mod,snr].shape)
      X.append(Xd[(mod,snr)])
      for i in range(Xd[(mod,snr)].shape[0]):  lbl.append((mod,snr))
X = np.vstack(X)
import matplotlib.pyplot as plt
y=np.zeros(shape=128)
for i in range(0,128):
   y[i]=i
plt.plot(y,X[-8000][0])
plt.plot(y,X[-8000][1])
plt.show()
```

Splitting Data

```
np.random.seed(2016)
n_examples = X.shape[0]
n_train = n_examples * 0.5
train_idx = np.random.choice(range(0,n_examples), size= int(n_train), replace=False)
test_idx = list(set(range(0,n_examples))-set(train_idx))
X_train = X[train_idx]
X_test =  X[test_idx]
# one hot encoding for multiclass classification since there are 11 classes (11 modulation techniques)
def to_onehot(yy):
      yy1 = np.zeros([len(yy), max(yy)+1])
      yy1[np.arange(len(yy)),yy] = 1
      return yy1
Y_train = to_onehot(list(map(lambda x: mods.index(lbl[x][0]), train_idx)))
Y_test = to_onehot(list(map(lambda x: mods.index(lbl[x][0]), test_idx)))
in_shp = list(X_train.shape[1:])
# print(X_train.shape, "     ", in_shp)
print(in_shp+[1])
classes = mods
```

***Note: The raw features were used in our training model as the derivatives provided so low accuracy.**

```
#Defining some varibales (Epochs ,Dropout rate , batch size)
dr = 0.5
# Set up some params for training
nb_epoch = 100    # number of epochs to train on
batch_size = 1024  # training batch size
```

1st - Fully connected neural network

```
#Build the N-Network
model = keras.models.Sequential()
model.add(Reshape(in_shp+[1], input_shape=in_shp))
model.add(Dropout(dr))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_normal', name="dense1"))
model.add(Dropout(dr))
model.add(Dense(128, activation='relu', kernel_initializer='he_normal', name="dense2"))
model.add(Dropout(dr))
model.add(Dense(64, activation='relu', kernel_initializer='he_normal', name="dense3"))
model.add(Dropout(dr))
model.add(Dense(64, activation='relu', kernel_initializer='he_normal', name="dense4"))
model.add(Dropout(dr))
model.add(Dense( len(classes), kernel_initializer='he_normal', name="dense5" ))
model.add(Activation('softmax'))
#Reshaping layer to output one of our 10 classes
model.add(Reshape([len(classes)]))

#Compile
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Perform training ...

```
filepath = 'convmodrecnets_CNN2_0.5.wts.h5'
history=model.fit(X_train,
   Y_train,
   batch_size=batch_size,
   epochs=nb_epoch,
   verbose=2,
    validation_split=0.05,
   callbacks = [
       keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
save_best_only=True, mode='auto'),
       keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')
   ])
# we re-load the best weights once training is finished
model.load_weights(filepath)
```

Evaluation (Show simple version of performance)

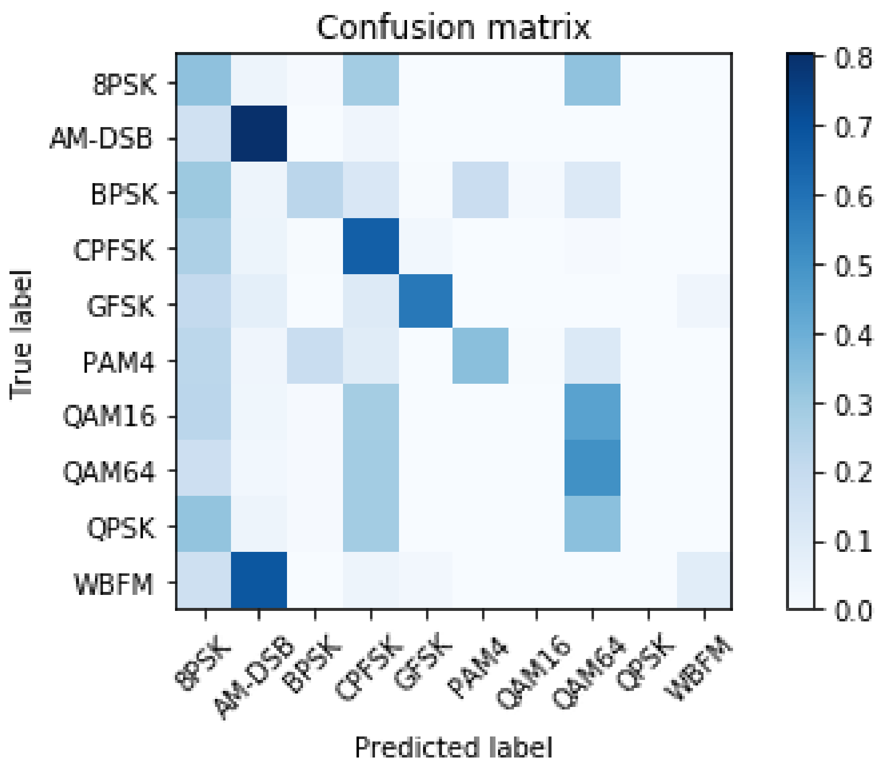
```
score = model.evaluate(X_test, Y_test, batch_size=batch_size)
print(model.metrics_names)
print(score)
```

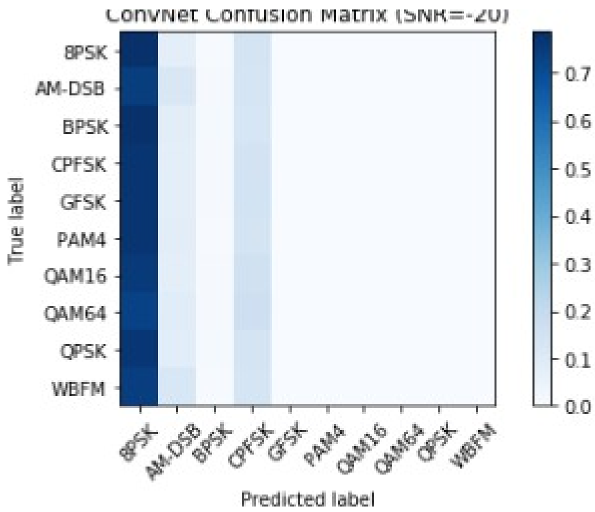
OUTPUT:

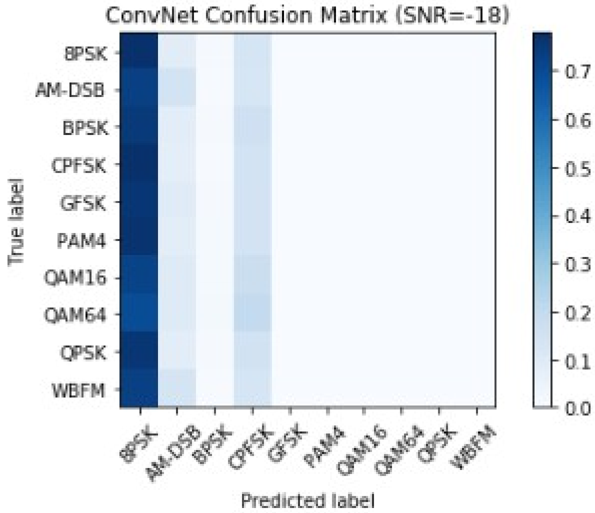
```
['loss', 'acc'] [1.2756312332344055, 0.46331166670481366]
```

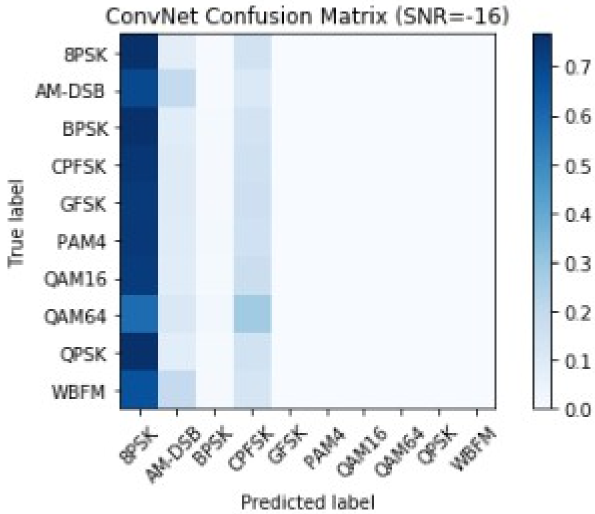
Overall Accuracy for each corresponding SNR value:

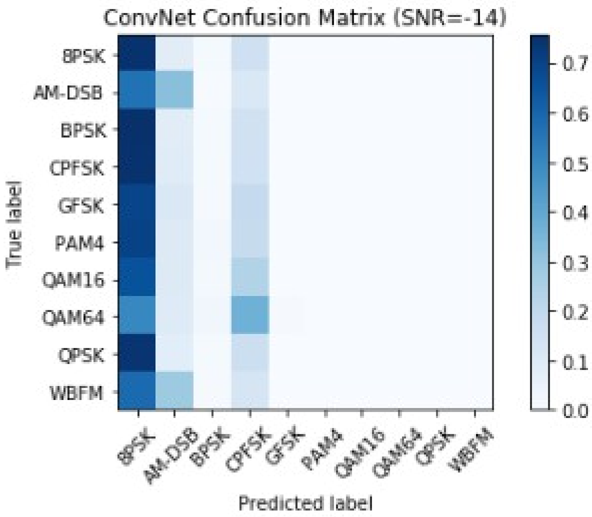
```
Overall Accuracy: 0.1051031995190702
Overall Accuracy: 0.10674194841641053
Overall Accuracy: 0.11280701169727064
Overall Accuracy: 0.12269363426313498
Overall Accuracy: 0.14665242640678192
Overall Accuracy: 0.21109963689663214
Overall Accuracy: 0.27046759366807155
Overall Accuracy: 0.3509035641513846
Overall Accuracy: 0.478809888718598
Overall Accuracy: 0.5091116173120729
Overall Accuracy: 0.46899654211568803
Overall Accuracy: 0.4675099866844208
Overall Accuracy: 0.463636061321539
Overall Accuracy: 0.46979287489643745
Overall Accuracy: 0.46533010256580476
Overall Accuracy: 0.4671100764881942
Overall Accuracy: 0.46211969908794354
Overall Accuracy: 0.46574888041134516
Overall Accuracy: 0.4626213430296561
Overall Accuracy: 0.4621969570305969
```

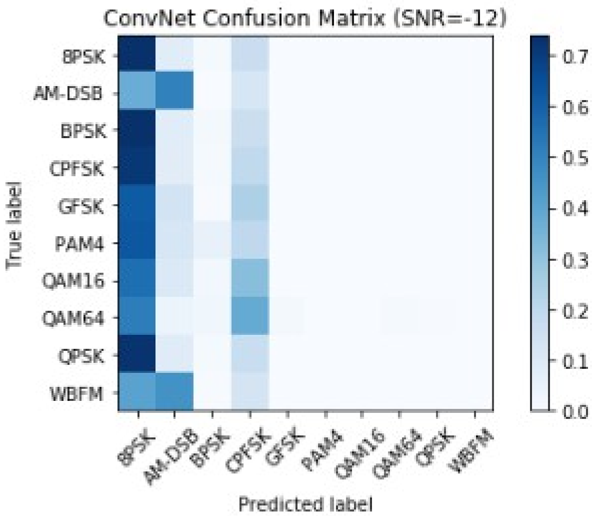


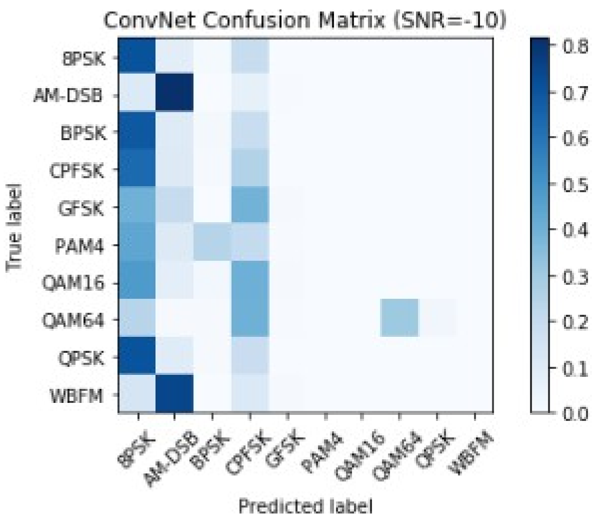

ConvNet Confusion Matrix (SNR=-20)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM

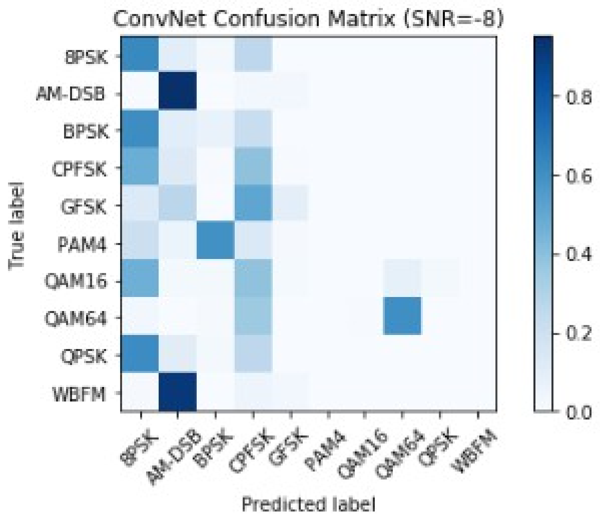

ConvNet Confusion Matrix (SNR=-18)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM

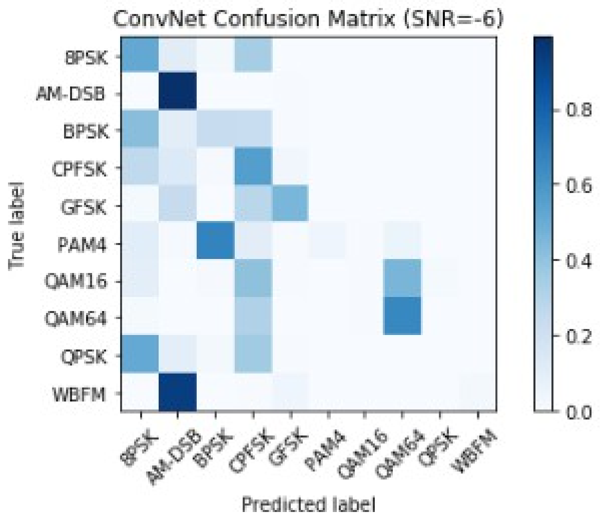

ConvNet Confusion Matrix (SNR=-16)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM

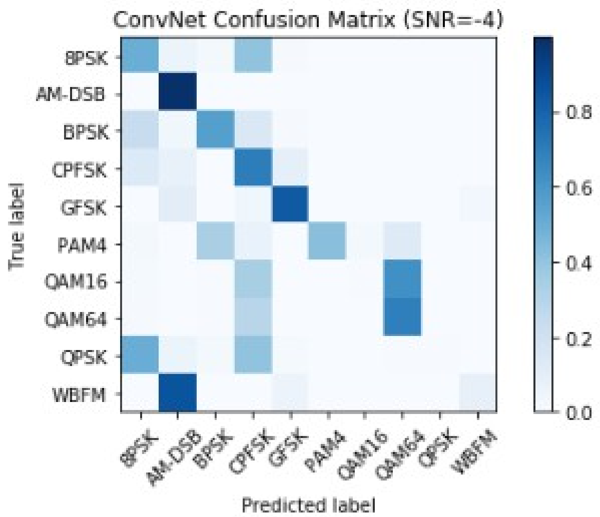

ConvNet Confusion Matrix (SNR=-14)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

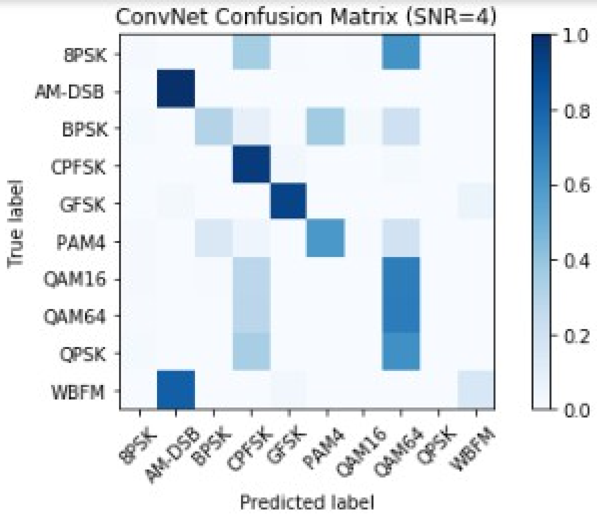

ConvNet Confusion Matrix (SNR=-12)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

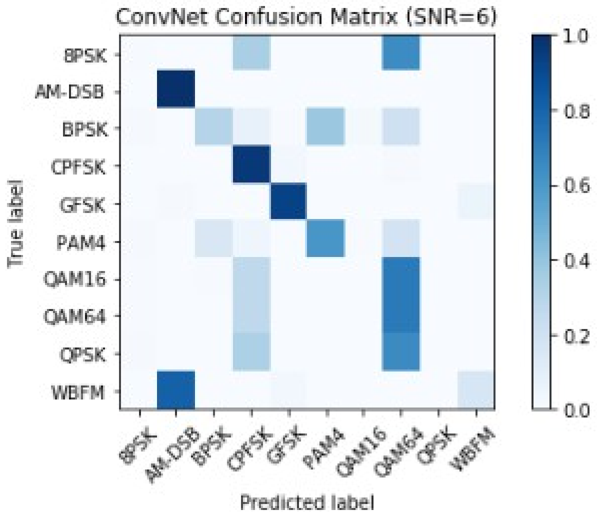

ConvNet Confusion Matrix (SNR=-10)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

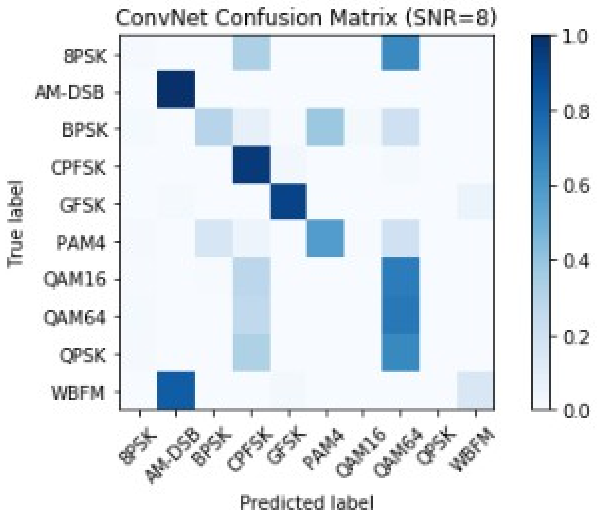

ConvNet Confusion Matrix (SNR=-8)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8

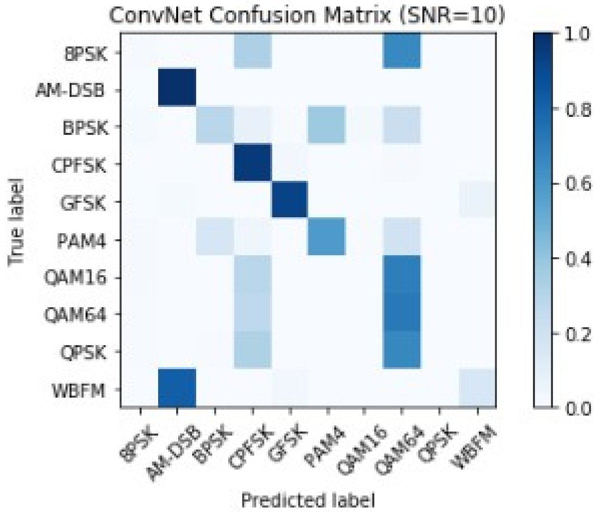

ConvNet Confusion Matrix (SNR=-6)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8

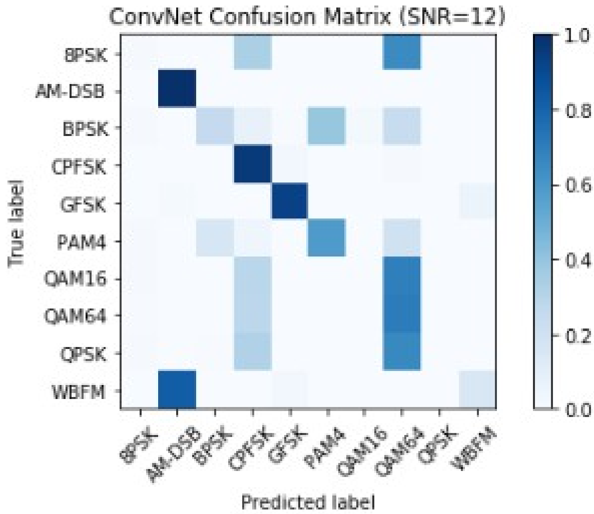

ConvNet Confusion Matrix (SNR=-4)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8

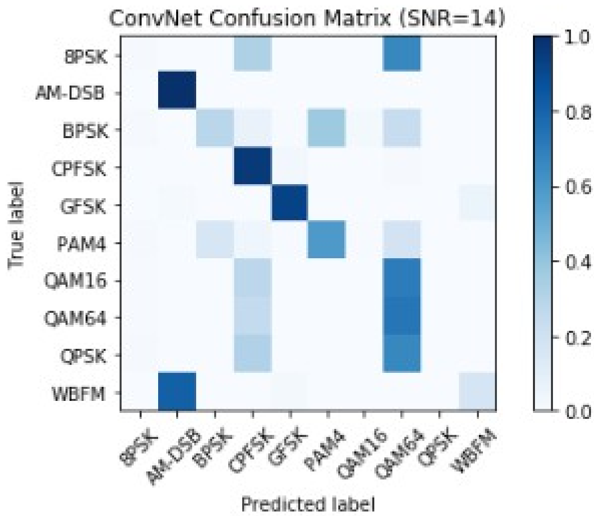

ConvNet Confusion Matrix (SNR=4)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0

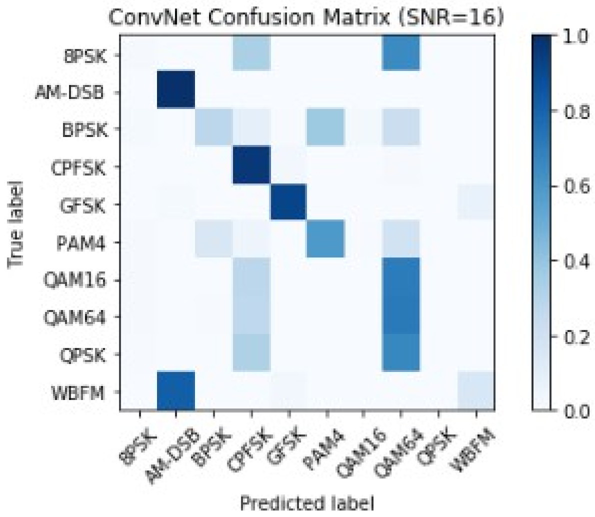

ConvNet Confusion Matrix (SNR=6)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0

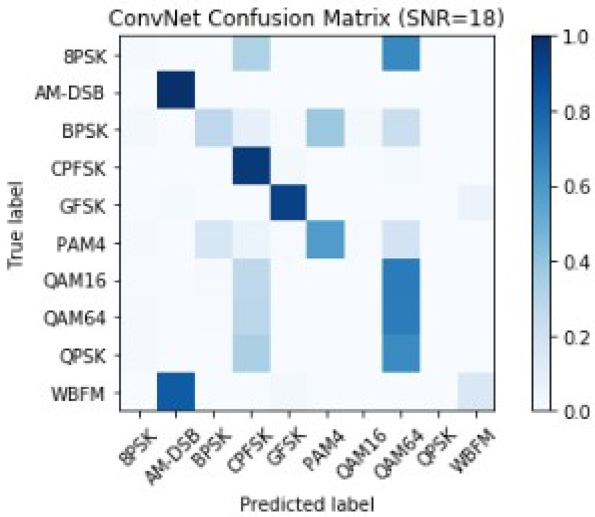

ConvNet Confusion Matrix (SNR=8)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0


ConvNet Confusion Matrix (SNR=10)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0


ConvNet Confusion Matrix (SNR=12)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0


ConvNet Confusion Matrix (SNR=14)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0





CNN

CNN using PDF Architecture

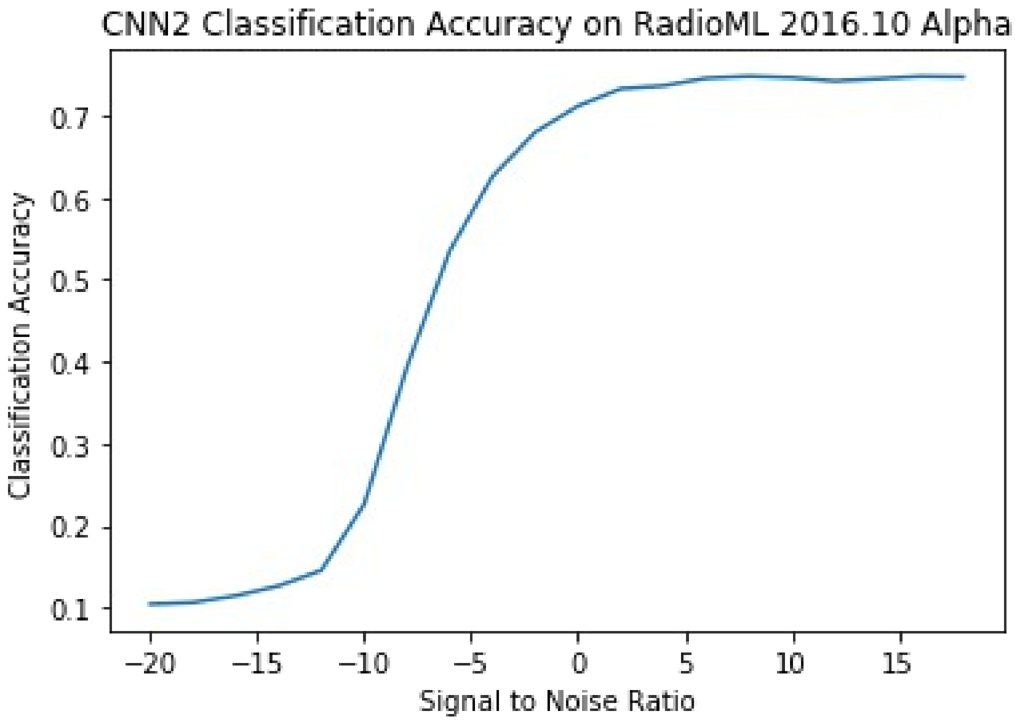
```
model2 = keras.models.Sequential()
model2.add(Reshape(in_shp+[1], input_shape=in_shp))
# no padding for the height and add padding to width (2 more columns)
model2.add(ZeroPadding2D((0, 2)))
model2.add(Conv2D(64, (1, 3), padding='valid', activation='relu', name="conv1",
          kernel_initializer='glorot_uniform', data_format="channels_last"))
# Adding dropout to inputs to next layer to avoid over fitting
model2.add(Dropout(dr))
model2.add(ZeroPadding2D((0, 2)))
```

```
model2.add(Conv2D(16, (2, 3), padding='valid', activation='relu', name="conv2",
              kernel_initializer='glorot_uniform', data_format="channels_last"))
model2.add(Dropout(dr))
# The coming layer is dense to we need to flatten our inputs
model2.add(Flatten())
model2.add(Dense(128, activation='relu', kernel_initializer='he_normal', name="dense1"))
model2.add(Dropout(dr))
model2.add(Dense( len(classes), kernel_initializer='he_normal', name="dense2" ))
model2.add(Activation('softmax'))
model2.add(Reshape([len(classes)]))
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.summary()

filepath = 'convmodrecnets_CNN2_0.5.wts.h5'
history=model2.fit(X_train,
   Y_train,
   batch_size=batch_size,
   epochs=nb_epoch,
   verbose=2,
    validation_split=0.05,
   callbacks = [
      keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
save_best_only=True, mode='auto'),
      keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='auto')
   ])
# we re-load the best weights once training is finished
model2.load_weights(filepath)
```

OutPut:

Last Epoch result:

```
Epoch 54/100 - 13s - loss: 1.2126 - acc: 0.4863 - val_loss: 1.1303 -
val_acc: 0.5225

[1.0480517408156396, 0.5663266666603088]

Overall Accuracy: 0.10436844566161245
Overall Accuracy: 0.10640785781103836
Overall Accuracy: 0.11457326623787782
Overall Accuracy: 0.1266115259685899
Overall Accuracy: 0.145384153260797
Overall Accuracy: 0.22595689396715413
Overall Accuracy: 0.39325656257259484
Overall Accuracy: 0.5348899355312824
Overall Accuracy: 0.625708002931965
Overall Accuracy: 0.6798204475412033
Overall Accuracy: 0.7117198777990399
Overall Accuracy: 0.7327896138482024
Overall Accuracy: 0.7364570516444415
Overall Accuracy: 0.7457166528583264
Overall Accuracy: 0.7481992896737146
Overall Accuracy: 0.7464582640505487
Overall Accuracy: 0.7422275480993276
Overall Accuracy: 0.7449991706750705
Overall Accuracy: 0.7480068052173333
Overall Accuracy: 0.7474001003176727
```



<u>Then we tried to modify the architectureand parameters as :</u>

<u>#Defining some varibales (Epochs ,Dropout rate , batch size)</u>
<u>dr = 0.05</u>
<u># Set up some params for training</u>
<u>nb_epoch = 100   # number of epochs to train on</u>
<u>batch_size = 1024 # training batch size</u>

<u>Model Code:</u>

```
model3 = keras.models.Sequential()
model3.add(Reshape(in_shp+[1], input_shape=in_shp))
# no padding for the height and add padding to width (2 more columns)
model3.add(ZeroPadding2D((0, 2)))
model3.add(Conv2D(64, (1, 3), padding='valid', activation='relu', name="conv1",
        kernel_initializer='glorot_uniform', data_format="channels_last"))
# Adding dropout to inputs to next layer to avoid over fitting
model3.add(Dropout(dr))
model3.add(ZeroPadding2D((0, 2)))
model3.add(Conv2D(64, (2, 3), padding='valid', activation='relu', name="conv2",
        kernel_initializer='glorot_uniform', data_format="channels_last"))
model3.add(Dropout(dr))
# The coming layer is dense to we need to flatten our inputs
model3.add(Flatten())
model3.add(Dense(128, activation='relu', kernel_initializer='he_normal', name="dense1"))
model3.add(Dropout(dr))
model3.add(Dense( len(classes), kernel_initializer='he_normal', name="dense2" ))
```

model3.add(Activation('softmax'))
model3.add(Reshape([len(classes)]))
model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

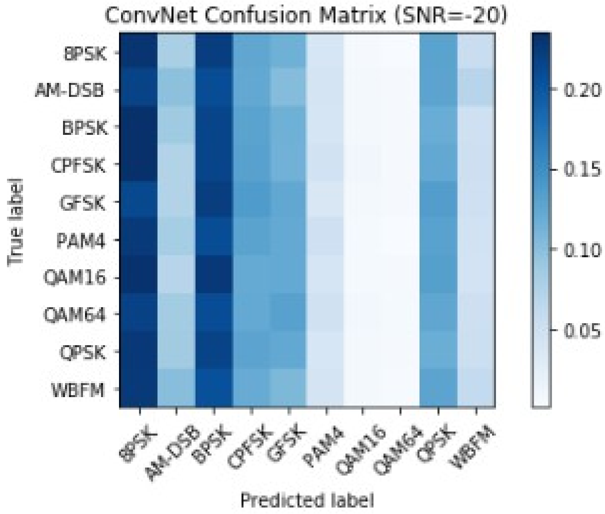
```
Train on 570000 samples, validate on 30000 samples
Epoch 1/100 - 14s - loss: 1.9088 - acc: 0.2465 - val_loss: 1.6231 - val_acc:
0.3500
Epoch 2/100 - 14s - loss: 1.4913 - acc: 0.3967 - val_loss: 1.3809 - val_acc:
0.4438
Epoch 3/100 - 14s - loss: 1.3094 - acc: 0.4631 - val_loss: 1.2695 - val_acc:
0.4779
Epoch 4/100 - 14s - loss: 1.2324 - acc: 0.4863 - val_loss: 1.2272 - val_acc:
0.4869
Epoch 5/100 - 14s - loss: 1.2044 - acc: 0.4940 - val_loss: 1.2255 - val_acc:
0.4787
Epoch 6/100 - 13s - loss: 1.1844 - acc: 0.5012 - val_loss: 1.1840 - val_acc:
0.4957
Epoch 7/100 - 13s - loss: 1.1689 - acc: 0.5066 - val_loss: 1.1785 - val_acc:
0.5038
Epoch 8/100 - 13s - loss: 1.1526 - acc: 0.5134 - val_loss: 1.1567 - val_acc:
0.5145
Epoch 9/100 - 13s - loss: 1.1376 - acc: 0.5187 - val_loss: 1.1464 - val_acc:
0.5081
Epoch 10/100 - 13s - loss: 1.1265 - acc: 0.5235 - val_loss: 1.1360 -
val_acc: 0.5149
Epoch 11/100 - 14s - loss: 1.1184 - acc: 0.5276 - val_loss: 1.1448 -
val_acc: 0.5076
Epoch 12/100 - 13s - loss: 1.1126 - acc: 0.5304 - val_loss: 1.1380 -
val_acc: 0.5201
Epoch 13/100 - 13s - loss: 1.1053 - acc: 0.5345 - val_loss: 1.1332 -
val_acc: 0.5269
Epoch 14/100 - 13s - loss: 1.0980 - acc: 0.5378 - val_loss: 1.1163 -
val_acc: 0.5309
Epoch 15/100 - 13s - loss: 1.0893 - acc: 0.5426 - val_loss: 1.1165 -
val_acc: 0.5341
Epoch 16/100 - 13s - loss: 1.0774 - acc: 0.5500 - val_loss: 1.1073 -
val_acc: 0.5391
Epoch 17/100 - 14s - loss: 1.0676 - acc: 0.5550 - val_loss: 1.0947 -
val_acc: 0.5433
Epoch 18/100 - 13s - loss: 1.0606 - acc: 0.5586 - val_loss: 1.1009 -
val_acc: 0.5376
Epoch 19/100 - 13s - loss: 1.0538 - acc: 0.5625 - val_loss: 1.0915 -
val_acc: 0.5471
Epoch 20/100 - 14s - loss: 1.0457 - acc: 0.5658 - val_loss: 1.0899 -
val_acc: 0.5465
Epoch 21/100 - 13s - loss: 1.0400 - acc: 0.5688 - val_loss: 1.0796 -
val_acc: 0.5489
Epoch 22/100 - 13s - loss: 1.0356 - acc: 0.5709 - val_loss: 1.1050 -
val_acc: 0.5414
Epoch 23/100 - 14s - loss: 1.0311 - acc: 0.5728 - val_loss: 1.0797 -
val_acc: 0.5503
Epoch 24/100 - 13s - loss: 1.0275 - acc: 0.5740 - val_loss: 1.0836 -
val_acc: 0.5480
Epoch 25/100 - 13s - loss: 1.0220 - acc: 0.5772 - val_loss: 1.0808 -
val_acc: 0.5509
Epoch 26/100 - 13s - loss: 1.0173 - acc: 0.5792 - val_loss: 1.0796 -
val_acc: 0.5504
Epoch 27/100 - 13s - loss: 1.0120 - acc: 0.5813 - val_loss: 1.0771 -
```

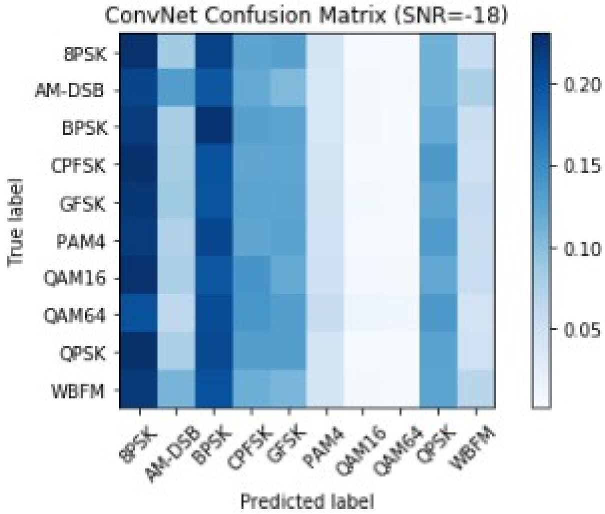
```
val_acc: 0.5535
Epoch 28/100 - 13s - loss: 1.0066 - acc: 0.5842 - val_loss: 1.0786 -
val_acc: 0.5541
Epoch 29/100 - 14s - loss: 1.0000 - acc: 0.5871 - val_loss: 1.0800 -
val_acc: 0.5526

Epoch 30/100 - 13s - loss: 0.9911 - acc: 0.5909 - val_loss: 1.0702 -
val_acc: 0.5540
Epoch 31/100 - 13s - loss: 0.9837 - acc: 0.5944 - val_loss: 1.0645 -
val_acc: 0.5594
Epoch 32/100 - 13s - loss: 0.9739 - acc: 0.5992 - val_loss: 1.0648 -
val_acc: 0.5560
Epoch 33/100 - 13s - loss: 0.9663 - acc: 0.6026 - val_loss: 1.0734 -
val_acc: 0.5593
Epoch 34/100 - 14s - loss: 0.9584 - acc: 0.6063 - val_loss: 1.0580 -
val_acc: 0.5653
Epoch 35/100 - 14s - loss: 0.9512 - acc: 0.6093 - val_loss: 1.0583 -
val_acc: 0.5652
Epoch 36/100 - 13s - loss: 0.9453 - acc: 0.6113 - val_loss: 1.0776 -
val_acc: 0.5547
Epoch 37/100 - 13s - loss: 0.9430 - acc: 0.6124 - val_loss: 1.0658 -
val_acc: 0.5657
Epoch 38/100 - 13s - loss: 0.9369 - acc: 0.6152 - val_loss: 1.0655 -
val_acc: 0.5662
Epoch 39/100 - 13s - loss: 0.9323 - acc: 0.6171 - val_loss: 1.0682 -
val_acc: 0.5604
Epoch 40/100 - 13s - loss: 0.9276 - acc: 0.6198 - val_loss: 1.0702 -
val_acc: 0.5650
Epoch 41/100 - 14s - loss: 0.9241 - acc: 0.6201 - val_loss: 1.0706 -
val_acc: 0.5628
Epoch 42/100 - 14s - loss: 0.9188 - acc: 0.6230 - val_loss: 1.0790 -
val_acc: 0.5608
Epoch 43/100 - 14s - loss: 0.9152 - acc: 0.6251 - val_loss: 1.0799 -
val_acc: 0.5628
Epoch 44/100 - 14s - loss: 0.9104 - acc: 0.6271 - val_loss: 1.0899 -
val_acc: 0.5628
```

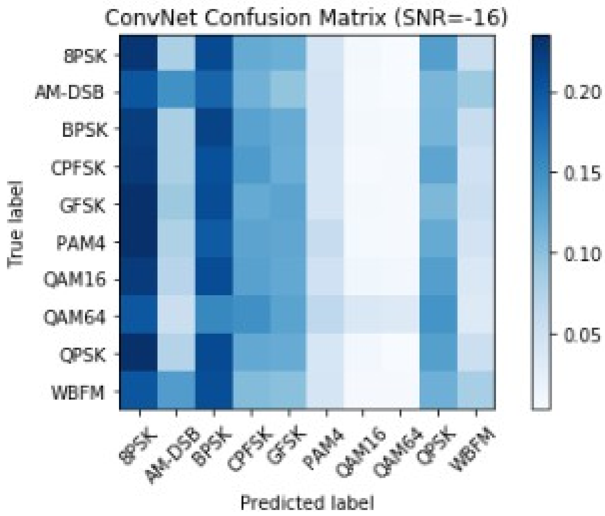
Print SNRS Accuracies

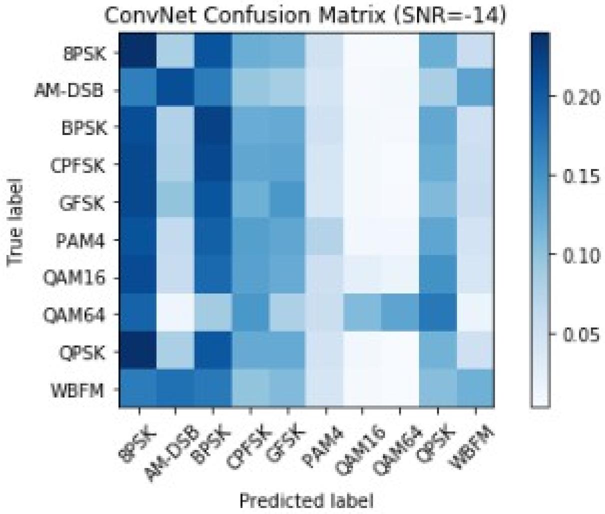
```
Overall Accuracy: 0.10289893794669695
Overall Accuracy: 0.10590672190298009
Overall Accuracy: 0.11583963741793582
Overall Accuracy: 0.14144593644309011
Overall Accuracy: 0.18433348908617583
Overall Accuracy: 0.27122822212598685
Overall Accuracy: 0.36730494806358477
Overall Accuracy: 0.4687176403781274
Overall Accuracy: 0.596021856466982
Overall Accuracy: 0.7141900040198311
Overall Accuracy: 0.7783596871118273
Overall Accuracy: 0.8067909454061252
Overall Accuracy: 0.8161351468191945
Overall Accuracy: 0.8202485501242751
```

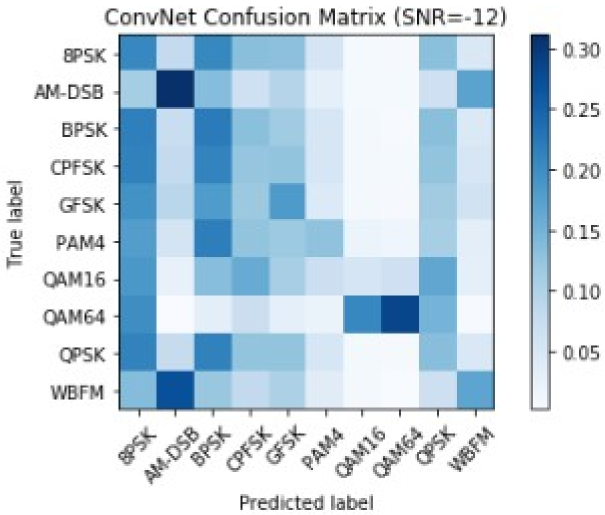
Graphs:

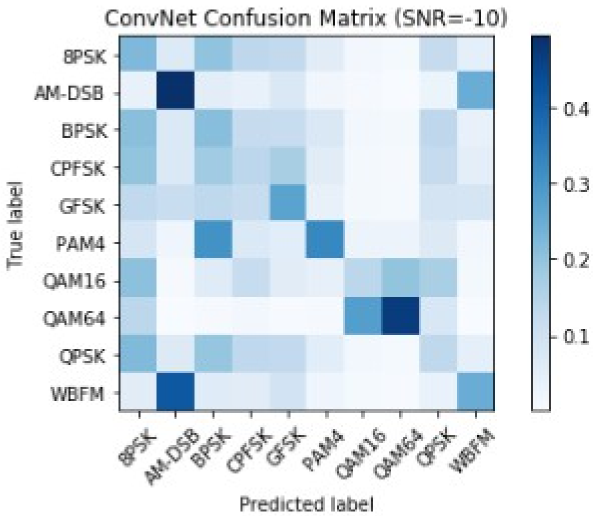

ConvNet Confusion Matrix (SNR=-20)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.20
0.15
0.10
0.05

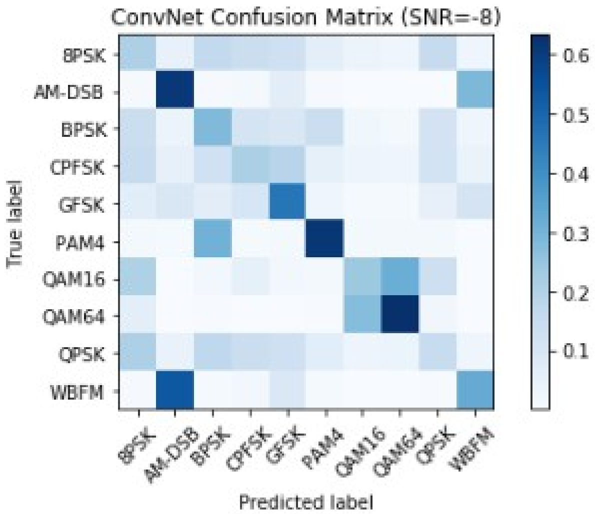

ConvNet Confusion Matrix (SNR=-18)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.20
0.15
0.10
0.05

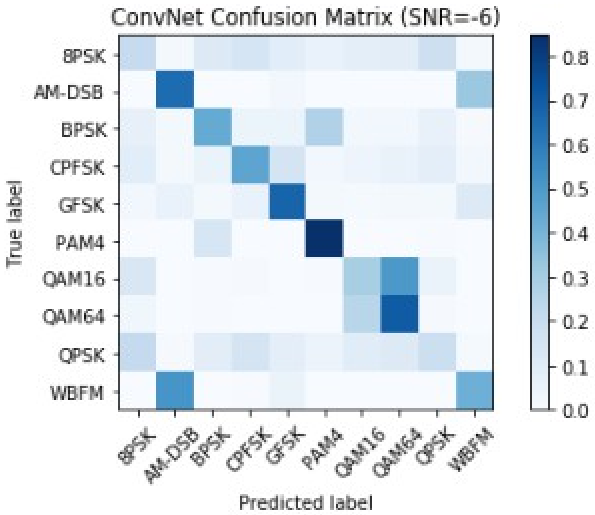

ConvNet Confusion Matrix (SNR=-16)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.20
0.15
0.10
0.05

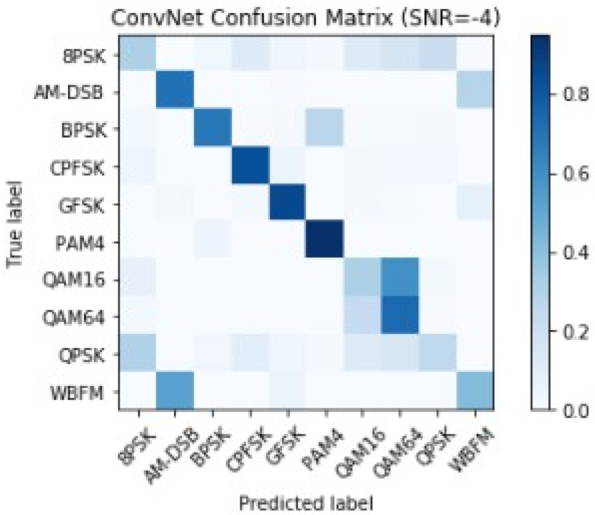

ConvNet Confusion Matrix (SNR=-14)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.20
0.15
0.10
0.05

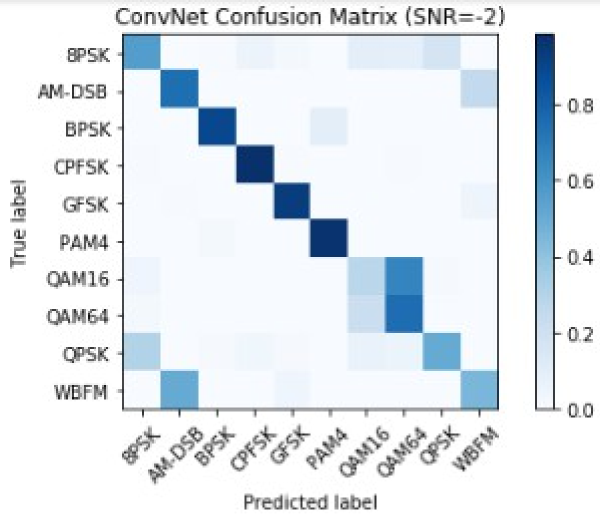

ConvNet Confusion Matrix (SNR=-12)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.30
0.25
0.20
0.15
0.10
0.05

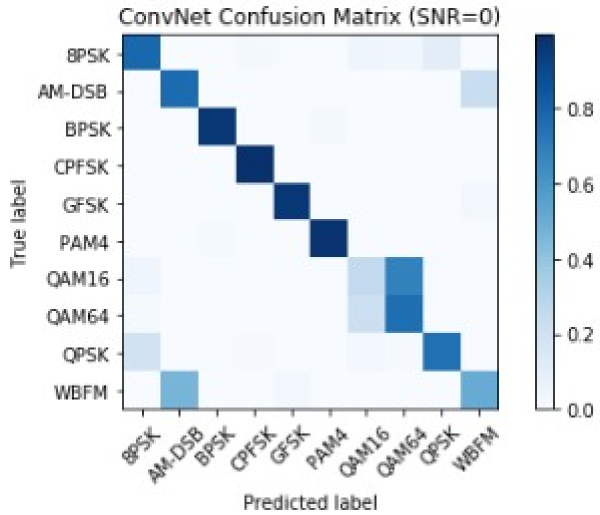

ConvNet Confusion Matrix (SNR=-10)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.4
0.3
0.2
0.1

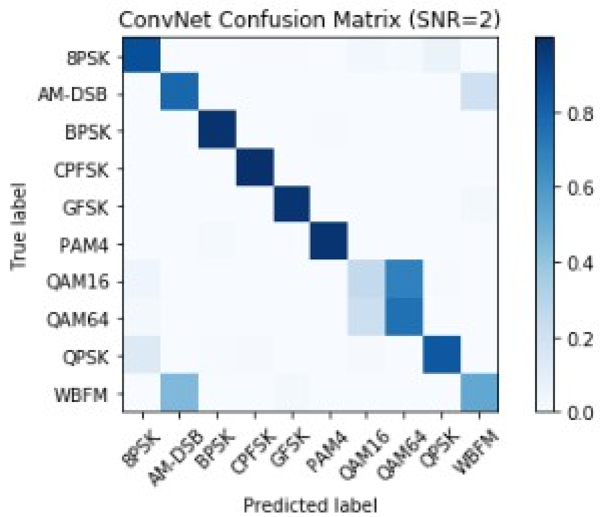

ConvNet Confusion Matrix (SNR=-8)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.1
0.2
0.3
0.4
0.5
0.6

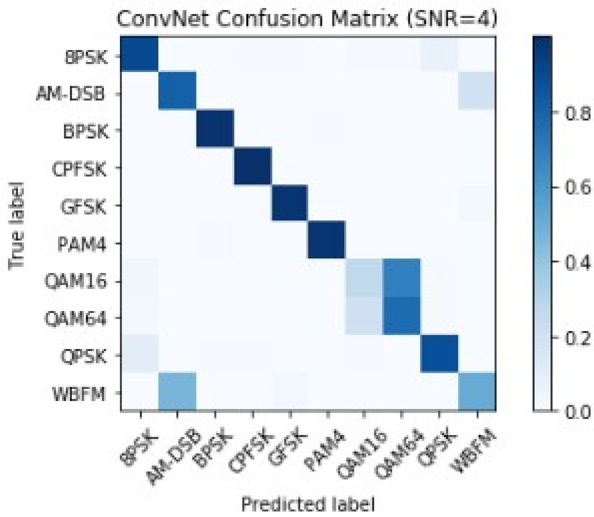

ConvNet Confusion Matrix (SNR=-6)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

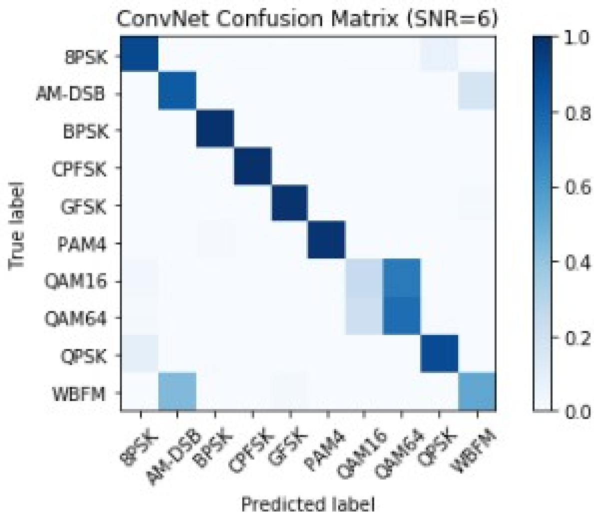

ConvNet Confusion Matrix (SNR=-4)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8

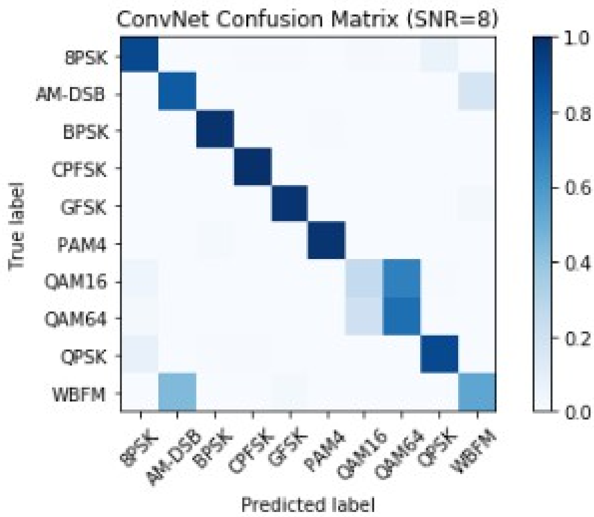

ConvNet Confusion Matrix (SNR=-2)
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
True label
Predicted label
0.0
0.2
0.4
0.6
0.8

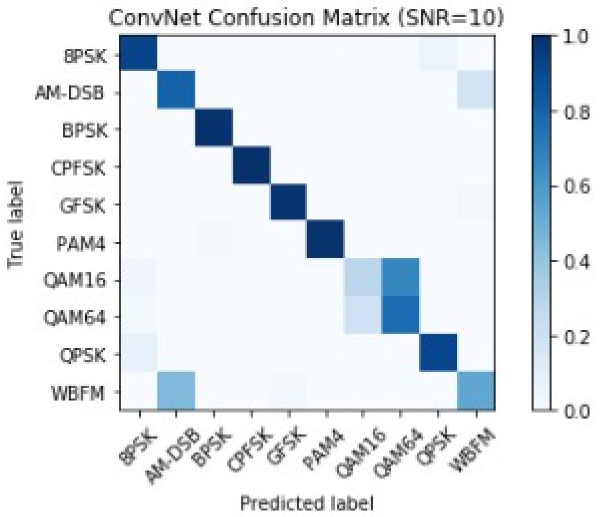

ConvNet Confusion Matrix (SNR=0)
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
True label
Predicted label
0.0
0.2
0.4
0.6
0.8


ConvNet Confusion Matrix (SNR=2)
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
True label
Predicted label
0.0
0.2
0.4
0.6
0.8

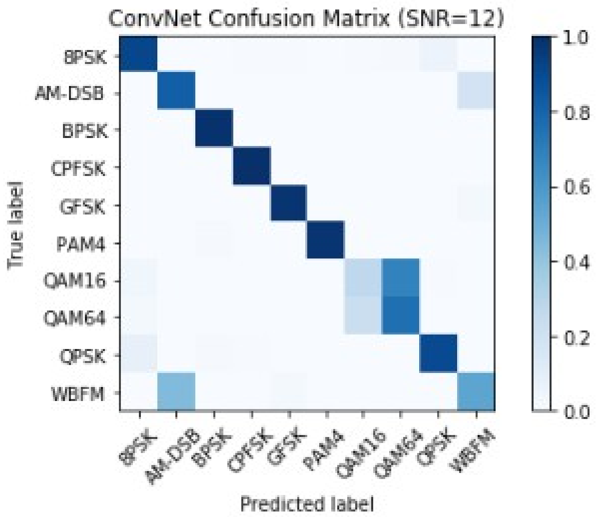

ConvNet Confusion Matrix (SNR=4)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8


ConvNet Confusion Matrix (SNR=6)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0

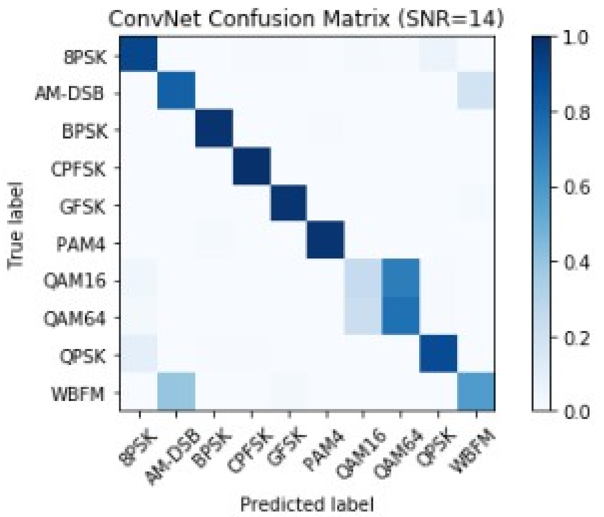

ConvNet Confusion Matrix (SNR=8)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
0.0
0.2
0.4
0.6
0.8
1.0



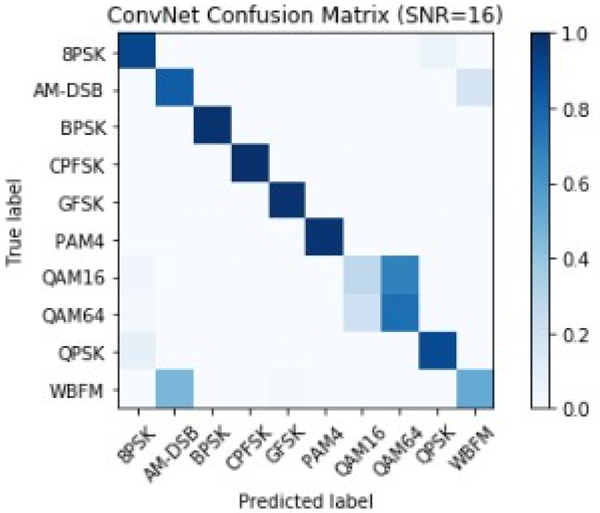
ConvNet Confusion Matrix (SNR=10)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0

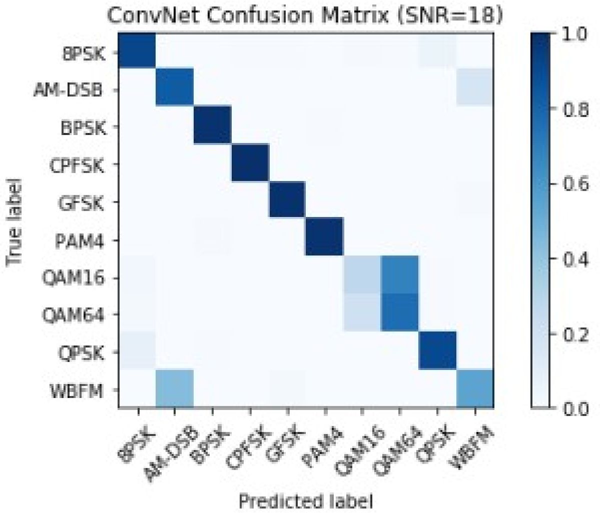


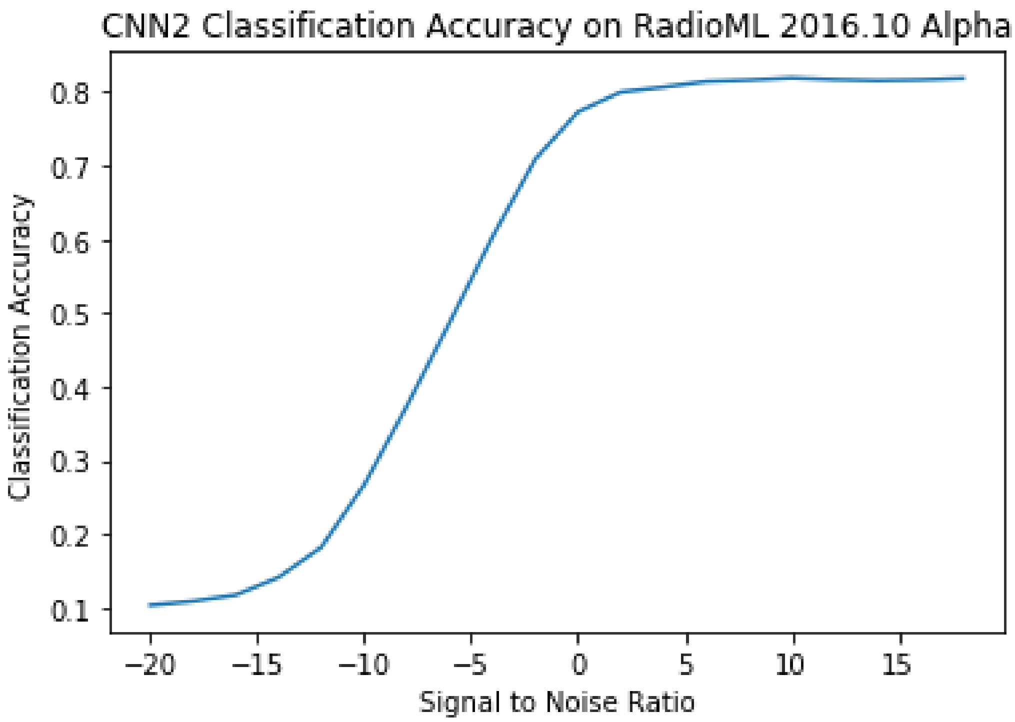
ConvNet Confusion Matrix (SNR=12)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0



ConvNet Confusion Matrix (SNR=14)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0


ConvNet Confusion Matrix (SNR=16)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0


ConvNet Confusion Matrix (SNR=18)
True label
Predicted label
8PSK
AM-DSB
BPSK
CPFSK
GFSK
PAM4
QAM16
QAM64
QPSK
WBFM
1.0
0.8
0.6
0.4
0.2
0.0


CNN2 Classification Accuracy on RadioML 2016.10 Alpha
Classification Accuracy
Signal to Noise Ratio
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
−20
−15
−10
−5
0
5
10
15

After trying some more model modifications:

```
model4 = keras.models.Sequential()
model4.add(Reshape(in_shp+[1], input_shape=in_shp))
# no padding for the height and add padding to width (2 more columns)
model4.add(ZeroPadding2D((0, 2)))
model4.add(Conv2D(256, (1, 3), padding='valid', activation='relu', name="conv1",
        kernel_initializer='glorot_uniform', data_format="channels_last"))
# Adding dropout to inputs to next layer to avoid over fitting
model4.add(Dropout(dr))
model4.add(ZeroPadding2D((0, 2)))
model4.add(Conv2D(256, (2, 3), padding='valid', activation='relu', name="conv2",
        kernel_initializer='glorot_uniform', data_format="channels_last"))
model4.add(Dropout(dr))
# The coming layer is dense to we need to flatten our inputs
model4.add(Flatten())
model4.add(Dense(256, activation='relu', kernel_initializer='he_normal', name="dense1"))
model4.add(Dropout(dr))
model4.add(Dense( len(classes), kernel_initializer='he_normal', name="dense2" ))
model4.add(Activation('softmax'))
model4.add(Reshape([len(classes)]))
model4.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

2

Last epoch result:

```
Epoch 25/100 - 111s - loss: 0.8556 - acc: 0.6522 - val_loss: 1.1813 -
val_acc: 0.5553
```