HW 6: Important Figures and Questions

The threshold I utilized for this assignment is as follows:

- The threshold utilized to determine if the image is moving will be from the gyro data. Since the gyro data has been normalized, values that are greater than 0.5 or less than -0.5 clearly indicate motion. Values closer to zero are more unclear.
- Thus, a threshold has been set to +/- 0.15, where if the average of the absolute value of the horizontal gyro data added to the absolute value of the vertical gyro data has a magnitude below 0.15, we will classify this as not moving.
- One hot encoding will be utilized, where values that falling within the range specified above will be set to zero, meaning not moving, where values outside of this range will be set to one, meaning moving.

The following code shows the implementation of the threshold:

```
#take the absolute value element wise (per ele)
arr_pos = np.absolute(desired_output)
print(arr_pos)
```

```
[[0.14203974 0.61433608]
 [0.05582077 0.02093982]
 [0.18238127 0.59205631]
 ...
 [0.07146101 0.20568902]
 [0.02087806 0.01215516]
 [0.20259262 0.05642209]]
```

```
threshold = 0.15
moving = np.zeros(nTimes)
for t in range(nTimes):
  moving[t] = (desired_output[t,0] > threshold or desired_output[t,1] > threshold)
```

The following code shows the conversion to 1 hot encoding:

```
encode = np.zeros((len(moving), 2))
encode[moving == 1] = [1, 0]   # moving
encode[moving == 0] = [0, 1]   # not moving
encode = encode.astype(int)
print(encode)
```

```
[[1 0]
 [0 1]
 [1 0]
 ...
 [1 0]
 [0 1]
 [0 1]]
```

Since the output (target) is now the encoded values, it is now the output for the Time Series Generator:

```
#######         Build Time Series Generator      ##############

from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from tensorflow.keras.callbacks import EarlyStopping

#here we will use the desired_output (horizontal, vertical components of movement from the gyro)
predict_length = 5    #because LSTM will use this sequence
batch_size = 1
training_set_size = int(0.9 * nTimes)
test_set_size = nTimes - training_set_size
gen2 = TimeseriesGenerator(frames_list, encode,
                           length = predict_length, batch_size=batch_size)

training_gen2 = TimeseriesGenerator(frames_list[:training_set_size], encode[:training_set_size,:],
                                    length = predict_length, batch_size=batch_size)

validation_gen2 = TimeseriesGenerator(frames_list[:-test_set_size], encode[:-test_set_size,:],
                                      length = predict_length, batch_size=batch_size)

early_stop = EarlyStopping(monitor='val_loss',patience=2)

#note the shape of the output:  (batch_size, predict_length, height, width, colorchannels)
gen2[100][0].shape
```

A final softmax layer is added to the network:

```
#split into two parallel streams
dense1a = Dense(units=n_dense_units, activation='sigmoid', name='dense1a')(flat_layer)
dense1b = Dense(units=n_dense_units, activation='sigmoid', name='dense1b')(flat_layer)
dense3a = Dense(units=1, activation='linear', name='dense3a')(dense1a)
dense3b = Dense(units=1, activation='linear', name='dense3b')(dense1b)
concat = Concatenate(axis=1)([dense3a, dense3b])
finaldense = Dense(units=2, activation='softmax')(concat) #final layer, force softmax
# put the two streams back together so the training data will work as a numpy array
output_layer = finaldense  #the output is the result of the two streams
```

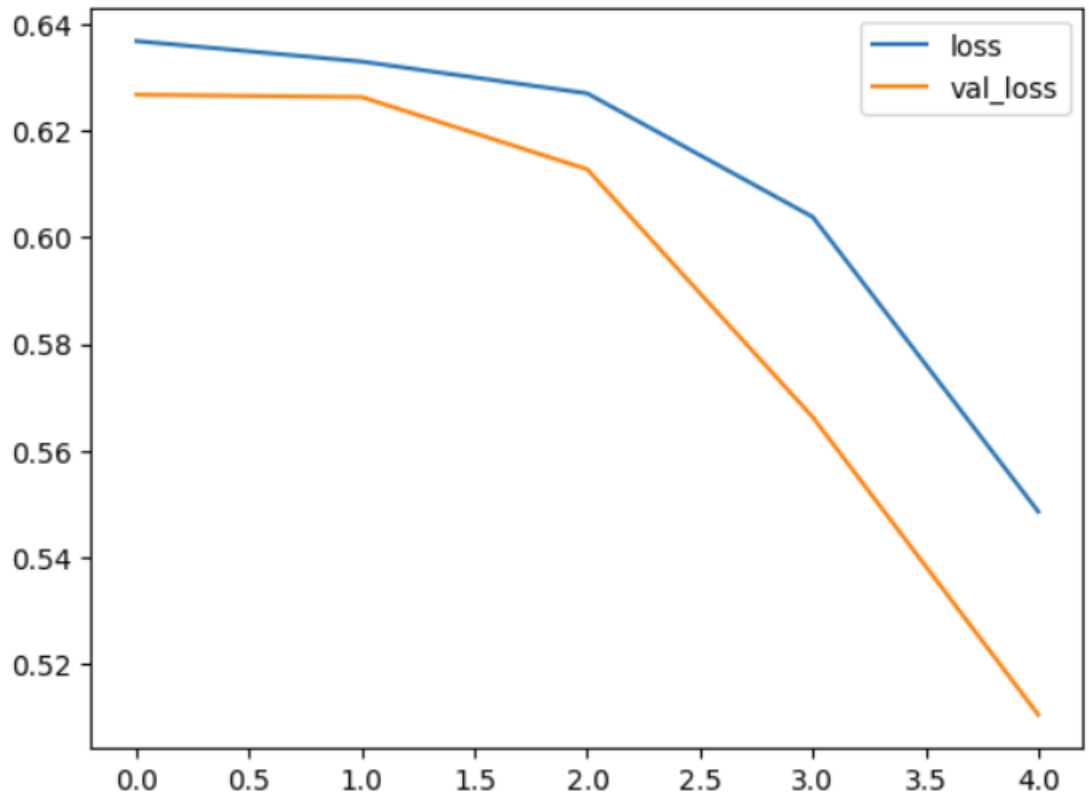
The output of the network is thus a probably vector p(x, t).

ML is utilized to decide whether we are moving or not:

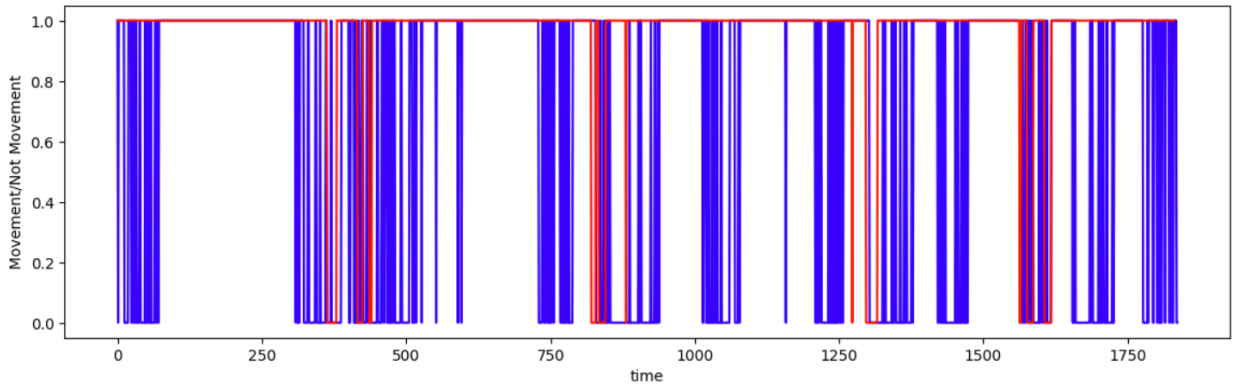
```
print(predicted_PMF) #output is p(x, t) which consists of 2 states
MLindex = np.argmax(predicted_PMF, axis=1) #axis=1, row wise
MLindex = MLindex.astype(int) #make sure it is integer
print("\n", MLindex) #pick max probability, gives winning index

pred_moving = np.zeros(nTimes) #initialize size
pred_moving = 1 - MLindex #flips since index 0 => moving = 1, and index 1 => moving = 0
print("\n", pred_moving)
```

```
[[0.58816475 0.41183525]
 [0.62747616 0.37252384]
 [0.64567393 0.35432607]
 ...
 [0.7947673  0.20523265]
 [0.8052848  0.19471522]
 [0.8150459  0.18495408]]

 [0 0 0 ... 0 0 0]

 [1 1 1 ... 1 1 1]
```

The losses are plotted:



The results comparing the true values (blue) to the predictions (red) are plotted:



Where 0 means not moving and 1 means moving.

Analysis:

From the losses and results plots, it is clear the network had trouble predicting the correct values.