1.2.b) MLP Trainable Parameters and FLOP's

I implemented a 3 layer MLP model having 178 input nodes, 16 hidden nodes, and 5 output nodes with the hidden layer having a sigmoid activation function.

Thus, the number of trainable parameters including bias parameters can be calculated as (n+1)*m where n is the input size and m is the output size:

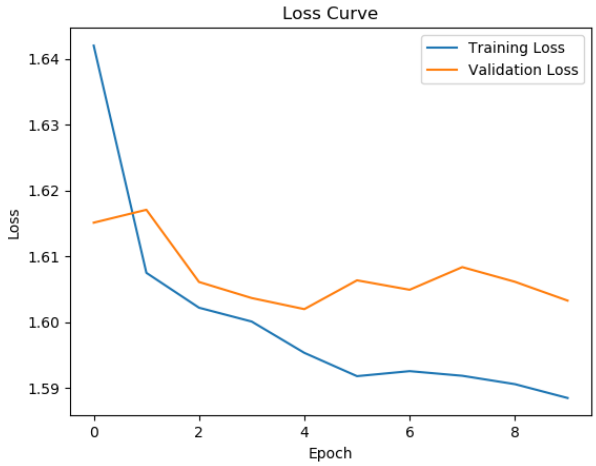
(178+1)*16+(16+1)*5 = 2949 Trainable Parameters

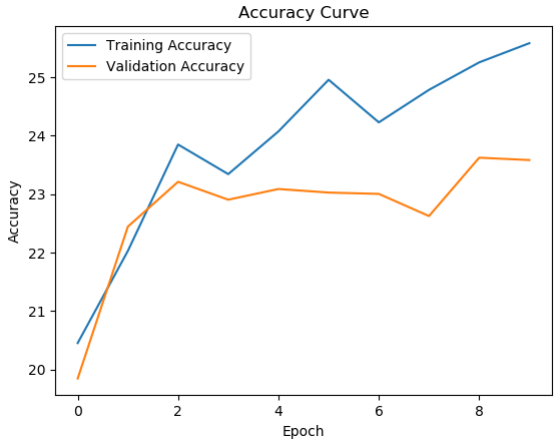
The number of FLOP's required for a forward and backward pass can be calculated as:

178*16 + 178*16 -1 + 4*(16+1) + 16*5 +16*5 -1 = 5922 Flops

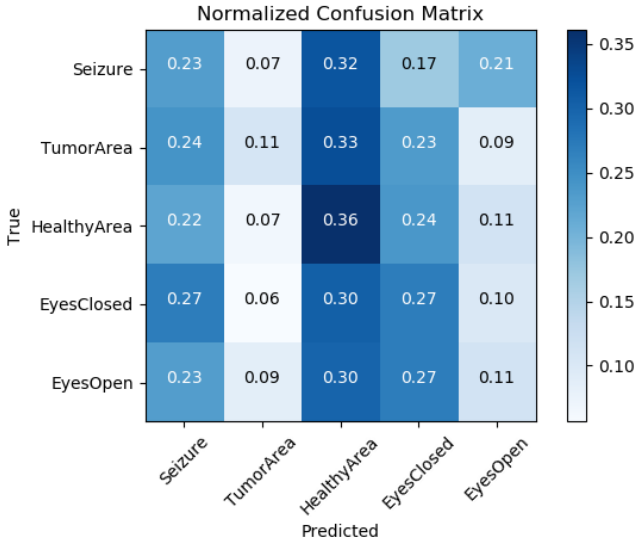
Here the sigmoid activation function has 4 flops and the flops for the fully connected layers can be calculated as (input_nodes * output_nodes) for one pass.

1.2.c) MLP Base Case Plots



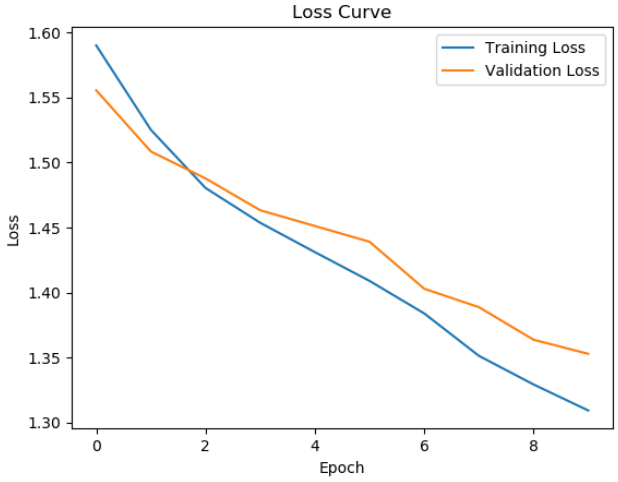


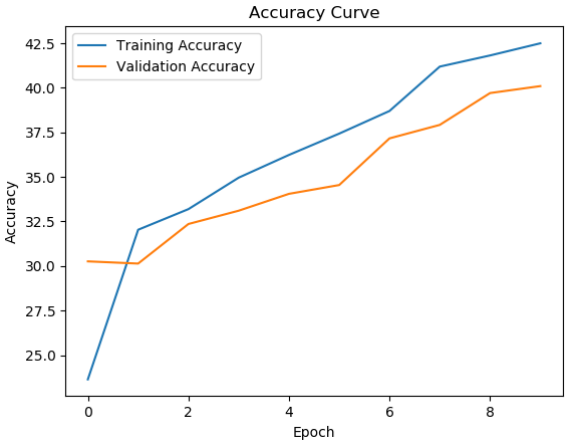
1.2.c) MLP Base Case Confusion Matrix

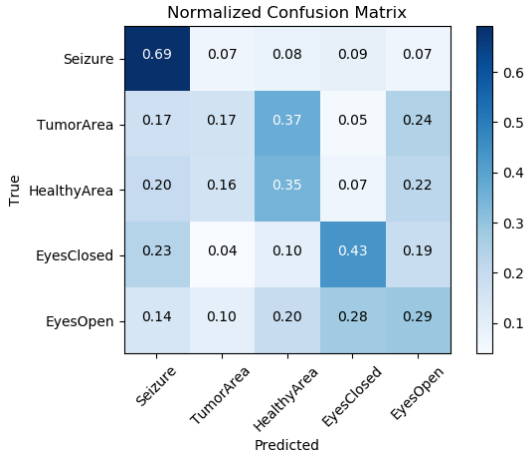


1.2.d) MLP Improvements

The basic MLP actually performs very poorly with only a ~25% accuracy. Thus, to improve the accuracy of my model, I added an additional hidden layer with a ReLu activation function to capture the higher level complexities in the dataset. This implementation doubled the performance of the model. The ReLu was used because it was noticed that in the base case sigmoid implementation the gradient dramatically vanished after 2 epochs as noticed in the large drop in the loss graph. The ReLu function overcomes this drawback as noticed in only a linear drop in loss in the below graph. Still, with this implementation accuracy is specifically poor for the Tumor Area class.







1.3.b) CNN Trainable Parameters and FLOP's

I implemented a 2 layer convolutional layer and 2 fully connected layer CNN model.

The number of trainable parameters can be calculated for a CNN layer as (n*m*l+1)*k, where n x m is filter size, l is input feature maps and k is output feature maps. Using this with the above knowledge of calculating the number of trainable parameters for the fully connected layer gives the below results:

(5*5-2+2)*6+(5*5*6+1)*16+ (16*41+1)*128+(128+1)*5 = 85273 Trainable parameters

The number of flops per CNN layer can be calculated as below:

n = output_channels * out_rows * out_columns
flops per instance = n + (n-1)

number of instances per filter = (((input_rows - output_channels + 2*padding) / stride) + 1)^2

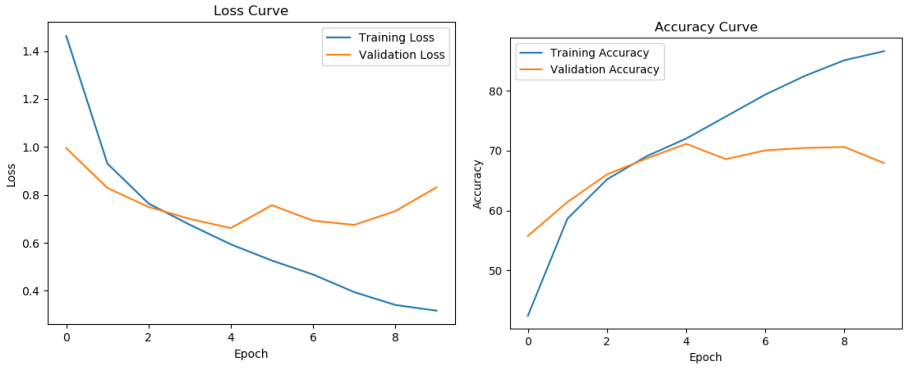
flops per filter = number of instances per filter * flops per instance
total flops per layer = flops per filter * number of filters + number of filters* in_rows * in_columns

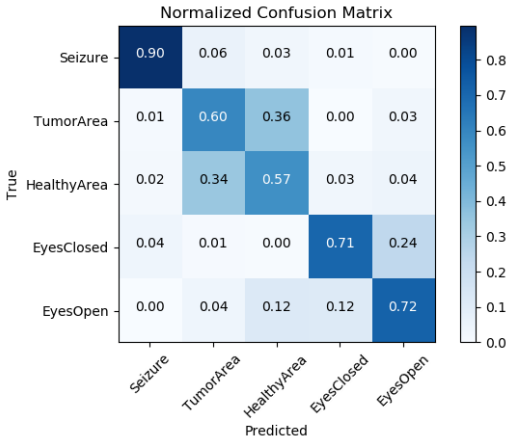
Thus, the total number of FLOPs can be calculated as:

CNN1 = $(6*5*5 + 6*5*5-1)*(178 – 6 +1)^2 *6 +16*178*1 = 53695474$
CNN2 = $(16*41 + 16*41-1)*(178*6 – 16 +1)^2 *16 +16*178*6 = 23258394672$
FC = 16*41*128 + 16*41*128 -1 + 2*(128+1) + 128*5 +128*5 -1 = 169472

Giving a total of 23312259618 FLOP's or 23 GFLOP's. This a huge increase in magnitude compared to the prior MLP implementation.

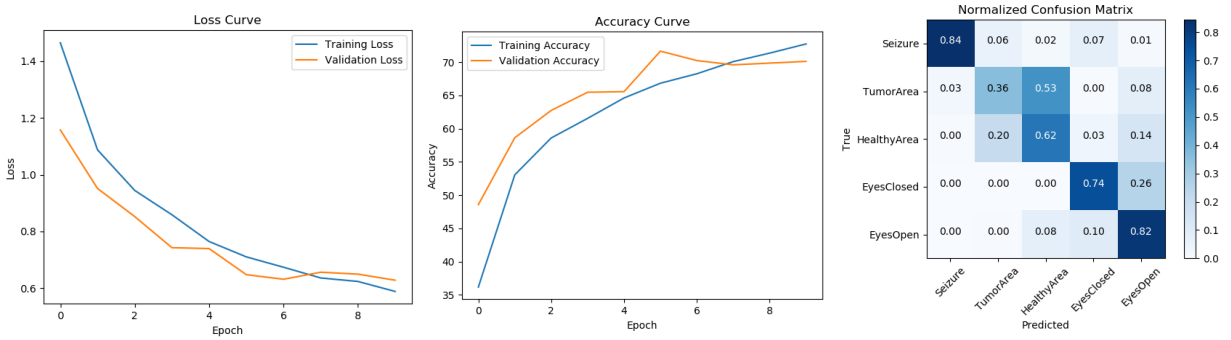
1.3.c) CNN Base Case Plots





1.3.d) CNN Improvements

I noticed that in the base case implementation that the model was overfitting to the training data as noticed by the near 90% training accuracy and poor loss curve on the validation set. Thus, to reduce overfitting I added a dropout layer after the last convolutional layer, and reduced the size of the fully connected layer to 60 hidden units and reduced the number of filters on the second convolutional layer to 8 channels to reduce model complexity. This though only gave a small improvement in performance mostly due to reduce accuracy with predicting seizures but there are notable improvements increases in accuracy for the other cases. This indicate that the model isn't overfitting to the training data.



1.4.b) RNN Trainable Parameters and FLOP's

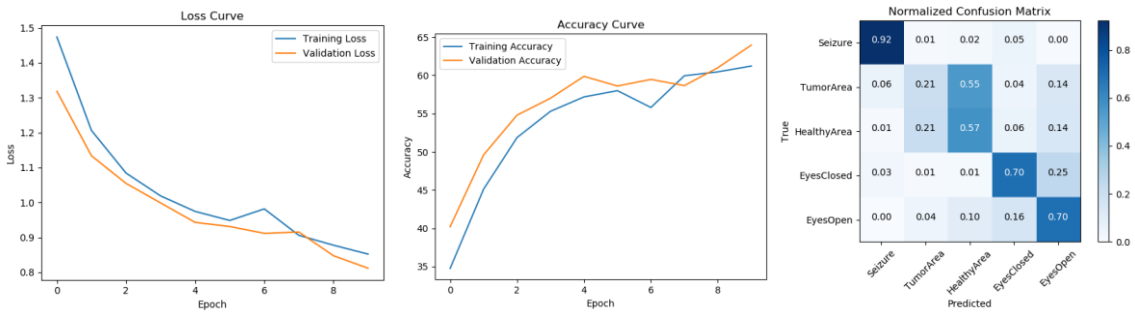
I implemented a 1 layer GRU layer and 1 fully connected layer RNN model.

The number of trainable parameters can be calculated for a RNN GNU layer as $3\times(n^2+nm+n)$, where m is the input dimension and n is the output dimension. Using this with the above knowledge of calculating the number of trainable parameters for the fully connected layer gives the below results:

3*(16*16+16+16) +(16+1)*5 = 997 Trainable parameters

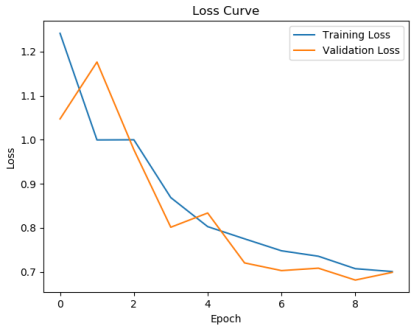
I couldn't figure out how to calculate FLOP's for RNN.

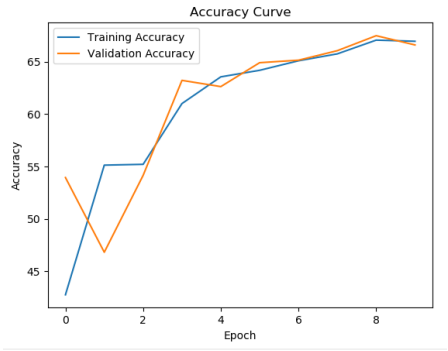
1.4.c) RNN Base Case Plots

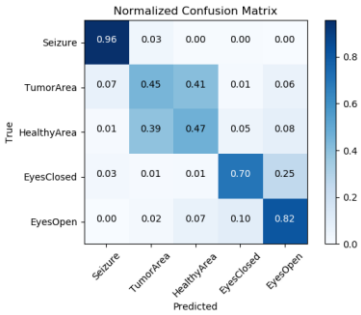


1.3.d) RNN Improvements

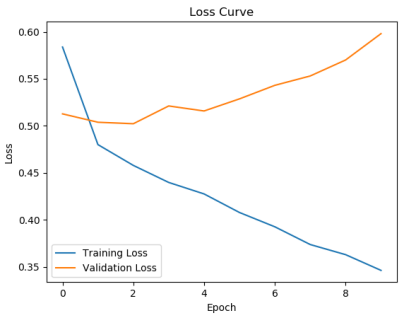
I noticed that the base model was underfitting and not complex enough to capture the patterns in the dataset. Thus, I decided to increase model complexity by adding a fully connected layer before the GRU layer as well as increasing the size of the GRU layer to 50 hidden units. This led to an increase in accuracy as shown below:

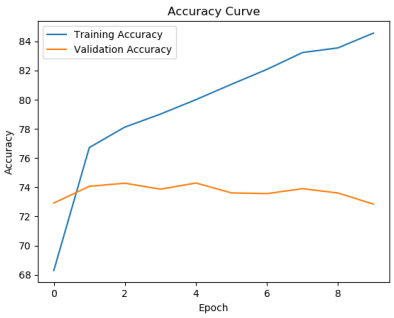


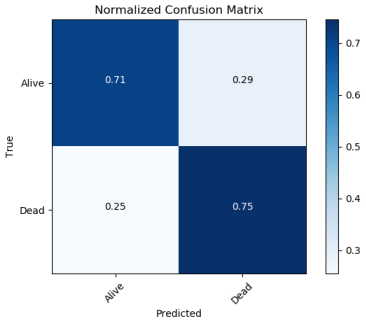




2.3.a) Mortality RNN Base Case Plots

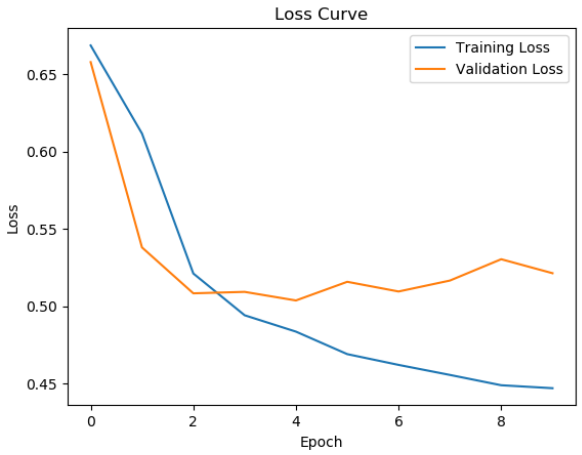


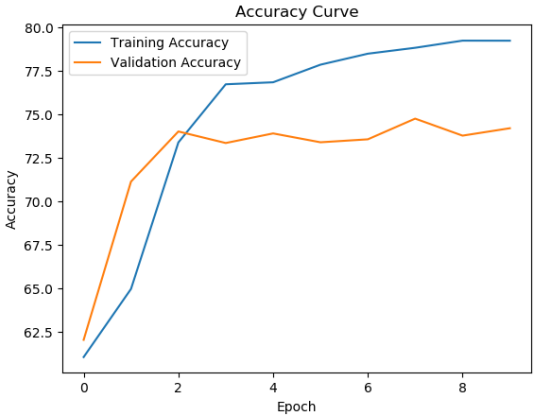


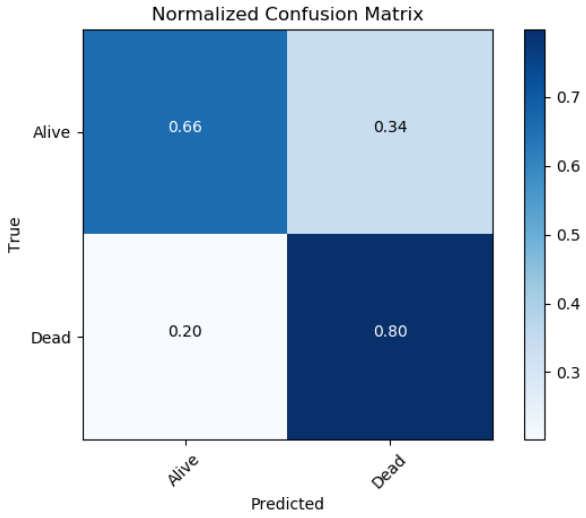


1.3.d) Mortality RNN Improvements

I noticed that the model was overfitting the data as shown by the high training accuracy, so I redcuced model complexity. I added dropouts to reduce overtraining. I further decreased the size of the first fully connected layer to 10 units while increasing the size of the GRU layer to 20 units to give more weight in the model to the RNN. Additionally, I added a sigmoid function after the first FC layer. As shown below, the overtraining was reduced but there is still a steep drop in loss.


Loss Curve
Training Loss
Validation Loss
Loss
Epoch
0.65
0.60
0.55
0.50
0.45
0
2
4
6
8


Accuracy Curve
Training Accuracy
Validation Accuracy
Accuracy
Epoch
80.0
77.5
75.0
72.5
70.0
67.5
65.0
62.5
0
2
4
6
8


Normalized Confusion Matrix
Alive
Dead
True
0.66
0.34
0.20
0.80
Alive
Dead
Predicted
0.7
0.6
0.5
0.4
0.3