Vincent La
903178639-vla6-hw5

1.2 Multi Layer Perceptron

1.2.b

Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur when a new single data point comes in to the model?

Note that on Calculating the Number of "trainable parameters", the following Piazza Posts were consulted:

1. *https://piazza.com/class/jjjilbkqk8m1r4?cid=928*
2. *https://piazza.com/class/jjjilbkqk8m1r4?cid=928*

Note that the number of "trainable parameters" come from Weights, biases, etc. which are being trained/learned in the model.

We can calculate this using code (note that formatting taken from this SO post):

```
print(sum(p.numel() for p in best_model.parameters() if p.requires_grad))
```

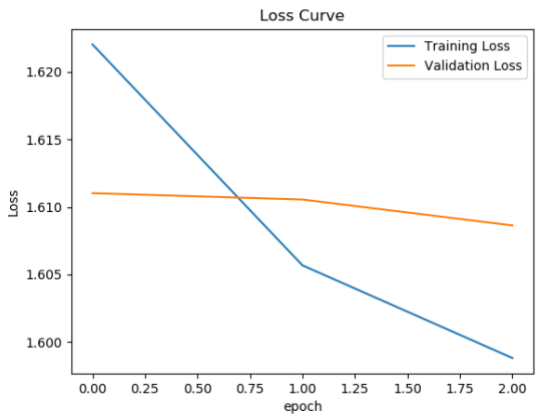
This returns 2,965 trainable parameters.

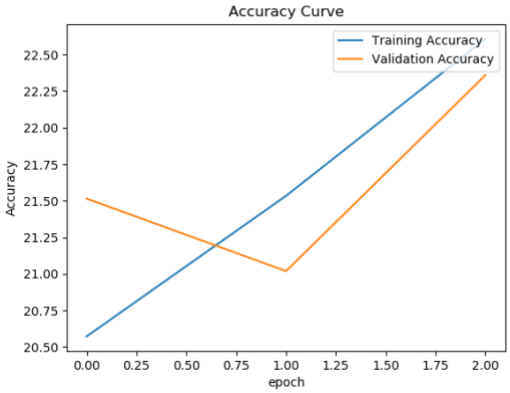
For the number of floating point operations, note that a sigmoid function is 4 operations. Furthermore, a dot product between two vectors of size n returns 2n -1 operations, since we have n multiplications and n-1 summations.

Thus, if there are 2,965 trainable parameters, we have close to ~20,000 floating point operations.

1.2.c

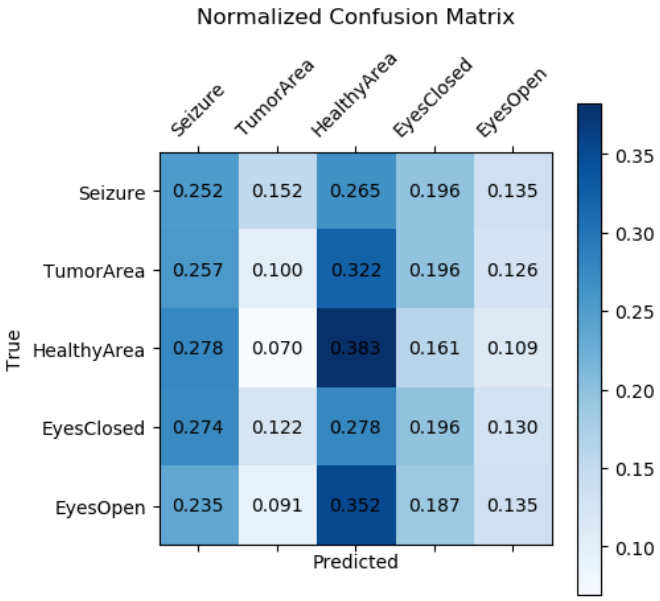
Learning Curves for MLP:





1.2.d

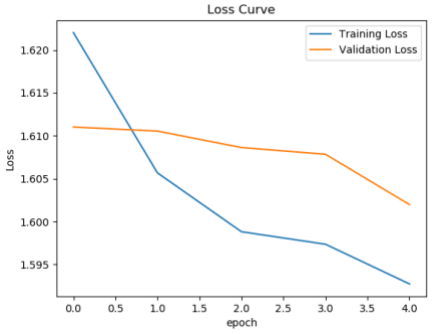
Confusion Matrix for MLP

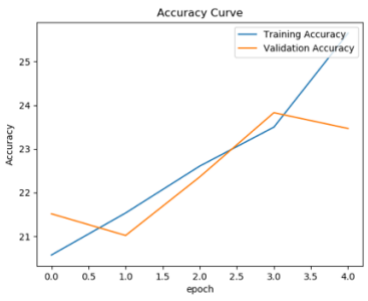


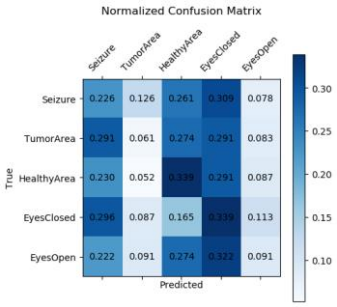
1.2.e

Modify your model class MyMLP in mymodels.py to improve the performance. It still must be a MLP type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots.

The first thing I tried was increasing the **number of epochs to 5** without changing anything else. Below are the graphs that resulted:

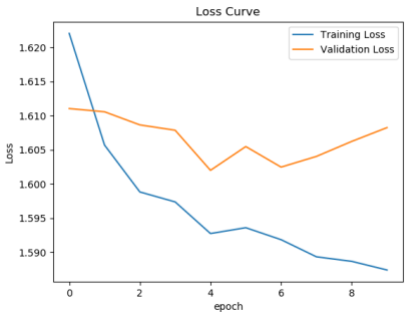


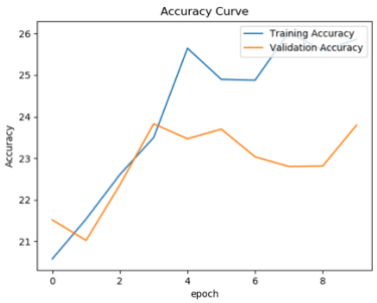


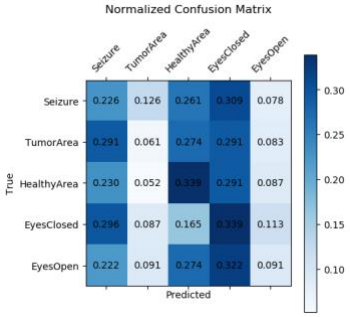


As you can see, even with changing epochs to 5 there is some improvements, Loss goes down closer to 1.6, and validation accurate moves up to around 23.5%.

Now, I try increasing the **number of epochs to 10** without changing anything else. Below are the graphs that results:

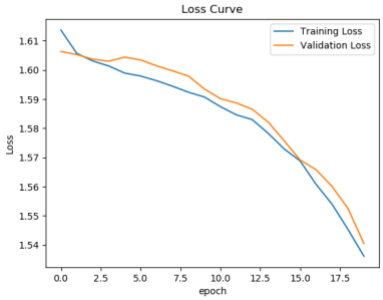


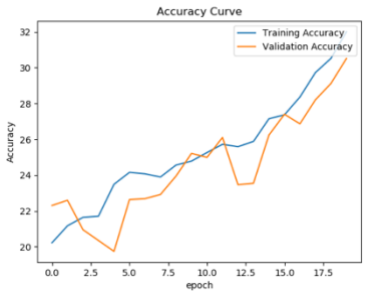


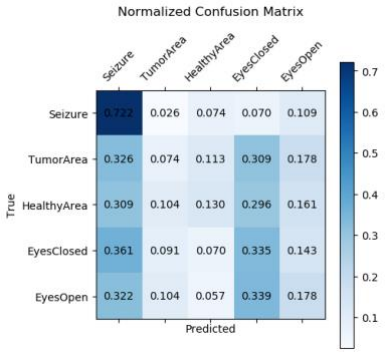


As you can see, not much difference just changing number of epochs to 10. In fact, we see Training performance improve but not Accuracy, which makes sense as we are now more likely to overfit the data. So next, we will change the epochs back to 5 and add more layers.

Epochs to 20 and Adding 1 More Hidden Layers (2 Total):
Next, we set Epochs to 20 and add 1 more Hidden Layers (2 Total)







Now we set Epochs to 20, but add one more hidden layer. Notice that now Validation Accuracy tracks with Training Accuracy and is up to 32, with Loss down to 1.54.

1.3 Convolutional Neural Network

1.3.b

Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur when a new single data point comes in to the model?

Note that the number of “trainable parameters” come from Weights, biases, etc. which are being trained/learned in the model.

We can calculate this using code (note that formatting taken from this SO post):

```
print(sum(p.numel() for p in best_model.parameters() if p.requires_grad))
```

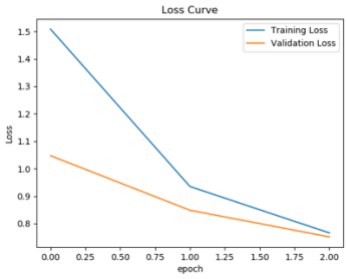
This returns 85,273 trainable parameters.

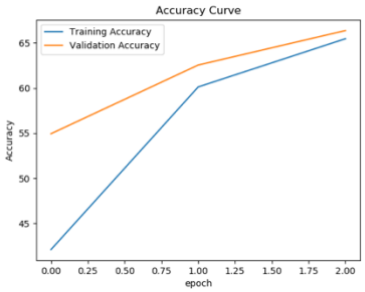
For the number of floating point operations, note that a sigmoid function is 4 operations. Furthermore, a dot product between two vectors of size n returns 2n -1 operations, since we have n multiplications and n-1 summations.

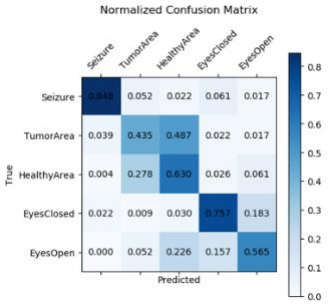
Thus, if there are 85,273 trainable parameters, we have close to ~200,000 floating point operations.

1.3.c

Plot and attach the learning curves and the confusion matrix for your CNN model in your report.





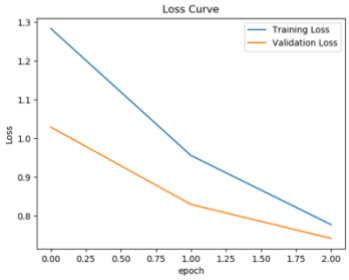


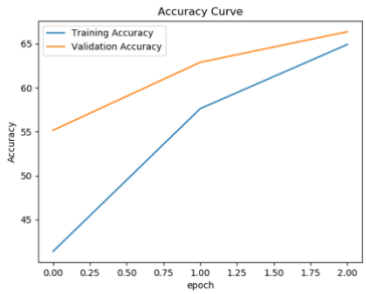
1.3.d

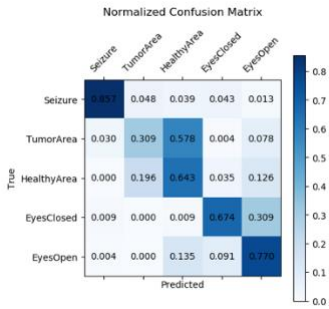
Modify your model class MyCNN in mymodels.py to improve the performance. It still must be a CNN type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots. (See this Piazza Post as potentially useful for benchmarking).

Having 3 Fully-Connected (aka Dense) Layers

The first change we try is to keep everything the same, but add one more Dense Layer, so that the model is exactly the same as what is presented in the lab.







As you can see above, adding in 1 more Fully Connected layer did not change much. Training Loss improved slightly, but Validation Loss and Accuracy are about the same.

1.4 Recurrent Neural Network

1.4.b

Calculate the number of "trainable" parameters in the model with providing the calculation details. How many floating-point computation will occur when a new single data point comes in to the model?

Note that the number of “trainable parameters” come from Weights, biases, etc. which are being trained/learned in the model.

We can calculate this using code (note that formatting taken from this SO post):

```
print(sum(p.numel() for p in best_model.parameters() if p.requires_grad))
```

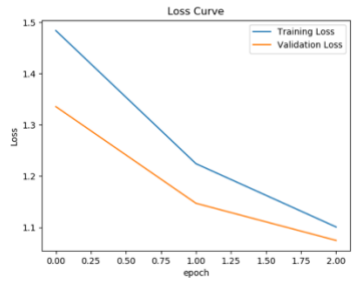
This returns 997 trainable parameters.

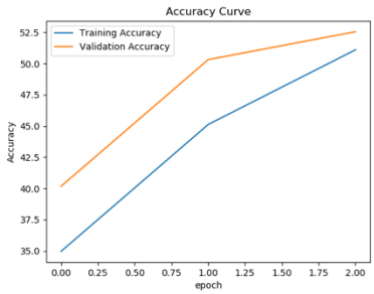
For the number of floating point operations, note that a tanh function is 4 operations. Furthermore, a dot product between two vectors of size n returns 2n -1 operations, since we have n multiplications and n-1 summations.

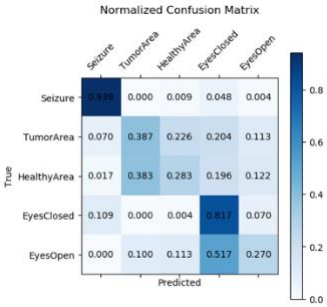
Thus, if there are 997 trainable parameters, we have close to ~10,000 floating point operations.

1.4.c

Plot and attach the learning curves and the confusion matrix for your RNN model in your report.





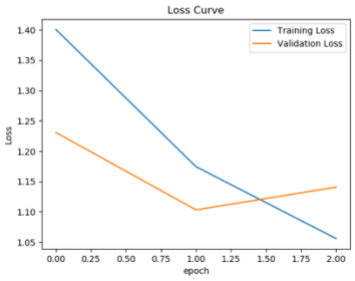


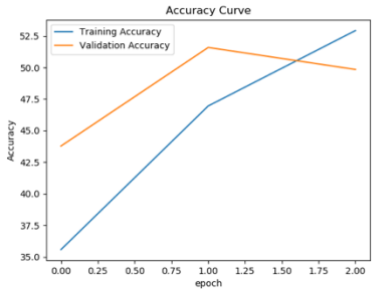
1.4.d

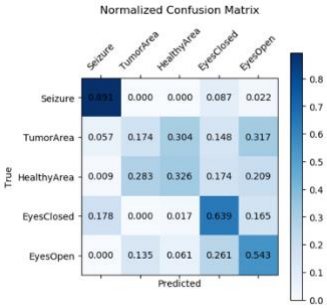
Modify your model class MyRNN in mymodels.py to improve the performance. It still must be a RNN type of architecture. Explain your architecture and techniques used. Briefly discuss about the result with plots.

2 Layers and 3 Epochs

The first RNN model we try next is an RNN model with everything the same as the initial HW Defaults but with 2 Layers:





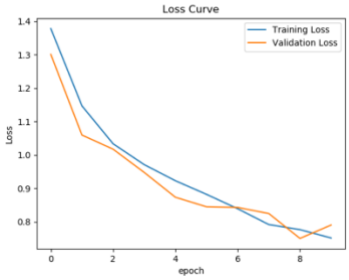


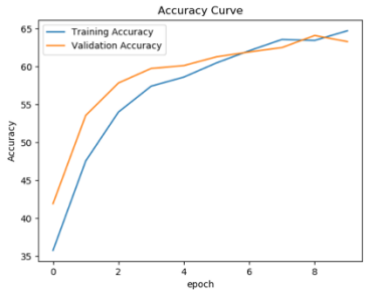
We can actually see with 2 layers with 3 epochs, it seems like accuracy and Loss did not actually improve, but that makes sense as we likely need to increase the epochs. Note that the number of trainable parameters increases to 2,629.

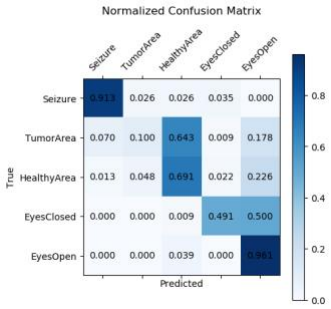
5 Layers and 10 Epochs

Next, we try 5 Layers and 10 Epochs

Now the trainable parameters increases to 7,525





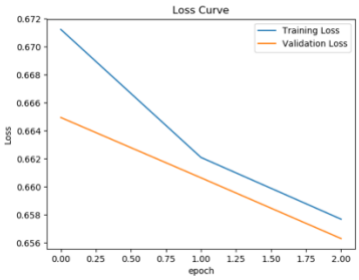


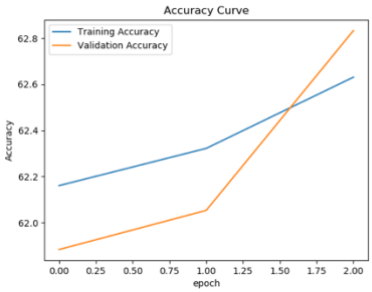
So you can see now that we changed number of layers to 5 and Epochs to 10, the model improves dramatically. Loss goes down to under 0.8 and Validation accuracy is around 62, which is a marketed improvement from the previous models.

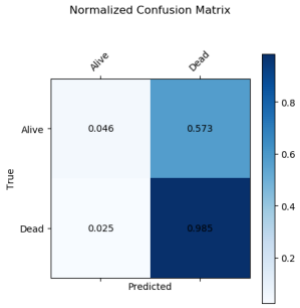
2.3 MyVariable RNN

2.3.a Plots

With 3 Epochs:



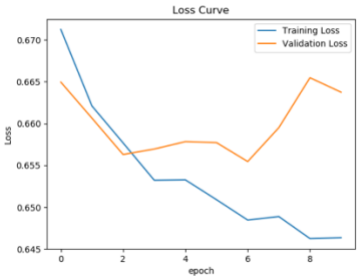


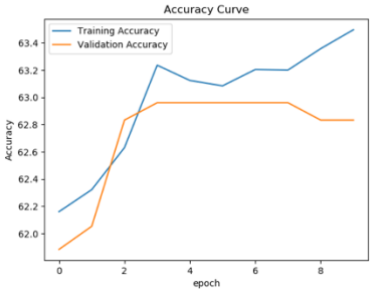


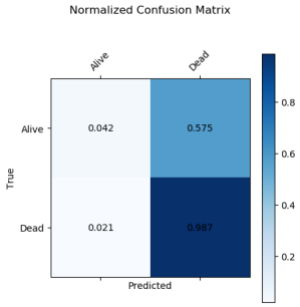
Note that loss and accuracy improve with the number of epochs. You can also see from the confusion matrix that the model is actually not performing well on Alive patients.

2.3.b Plots

So to improve our model, we try 10 Epochs:







As you can see, which makes sense, changing number of epochs improved performance slightly, but not a lot.