Timothy Horrell
ML Applications in ECE
Homework 5

Question 1:

A)

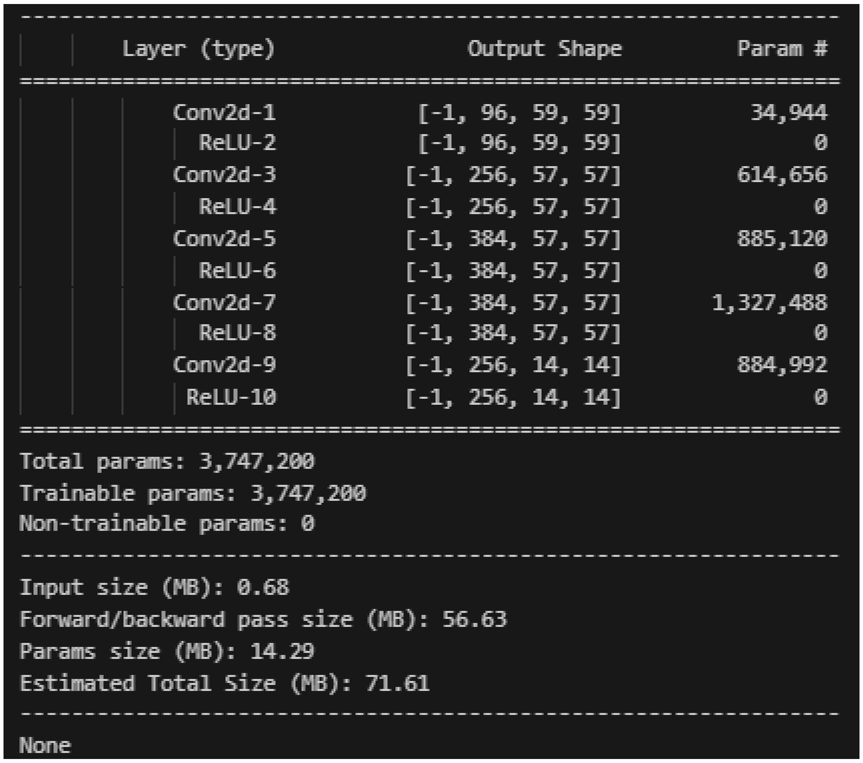
Formula for trainable parameters = input channel x output channel x filter size + bias

The number of parameters in each layer are as follows:
Input layer - None
First Layer - 3 * 96 * (11*11) + 96 = 34,944 parameters
Second Layer – 96 * 256 * (5*5) + 256 = 614,656 parameters
Third Layer – 256 * 384 * (3*3) + 384 = 885,129 parameters
Fourth Layer – 384 * 384 * (3*3) + 384 = 1,327,488 parameters
Fifth Layer – 384 * 256 * (3*3) + 256 = 884,992 parameters

Total parameters: 3,747,200 parameters

B)

AlexNet_CNN Summary:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 59, 59]          34,944
              ReLU-2           [-1, 96, 59, 59]               0
            Conv2d-3          [-1, 256, 57, 57]         614,656
              ReLU-4          [-1, 256, 57, 57]               0
            Conv2d-5          [-1, 384, 57, 57]         885,120
              ReLU-6          [-1, 384, 57, 57]               0
            Conv2d-7          [-1, 384, 57, 57]       1,327,488
              ReLU-8          [-1, 384, 57, 57]               0
            Conv2d-9          [-1, 256, 14, 14]         884,992
             ReLU-10          [-1, 256, 14, 14]               0
================================================================
Total params: 3,747,200
Trainable params: 3,747,200
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.68
Forward/backward pass size (MB): 56.63
Params size (MB): 14.29
Estimated Total Size (MB): 71.61
----------------------------------------------------------------
None
```

The parameters line up exactly as expected from my calculation in part A.

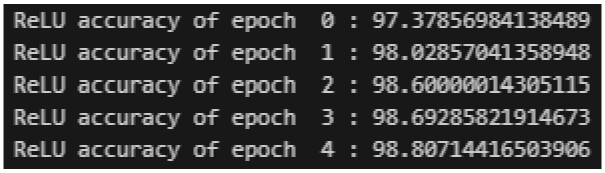
Question 2:

The CNN was trained with mini-batch SGD, with a batch size of 5. The validation loss was evaluated after each epoch, while training loss was evaluated after each batch.

The __ network gives the best prediction accuracy on the testing dataset. The activation functions have their own strengths and weaknesses. Because the sigmoid is bound to 0 and 1, it is resistant to 'blowing up' the activation. The Relu however holds on to attributes of parameters far from the y-axis. For example, far reaching values on the sigmoid x-axis would equal 0.001 or 0.00001, which are both essentially 0.

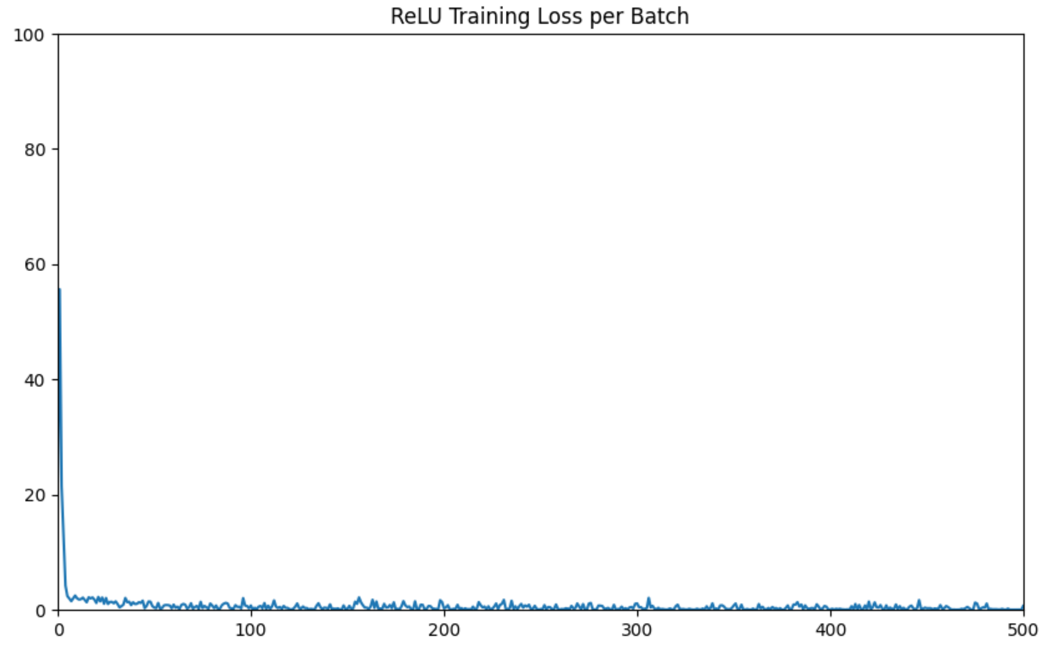
Based on these results, the sigmoid function took longer to learn, and didn't end up performing as strongly as the ReLU function. Additionally, it had longer runtime, meaning the ReLU is stronger in all facets here.

ReLU Testing Accuracy:

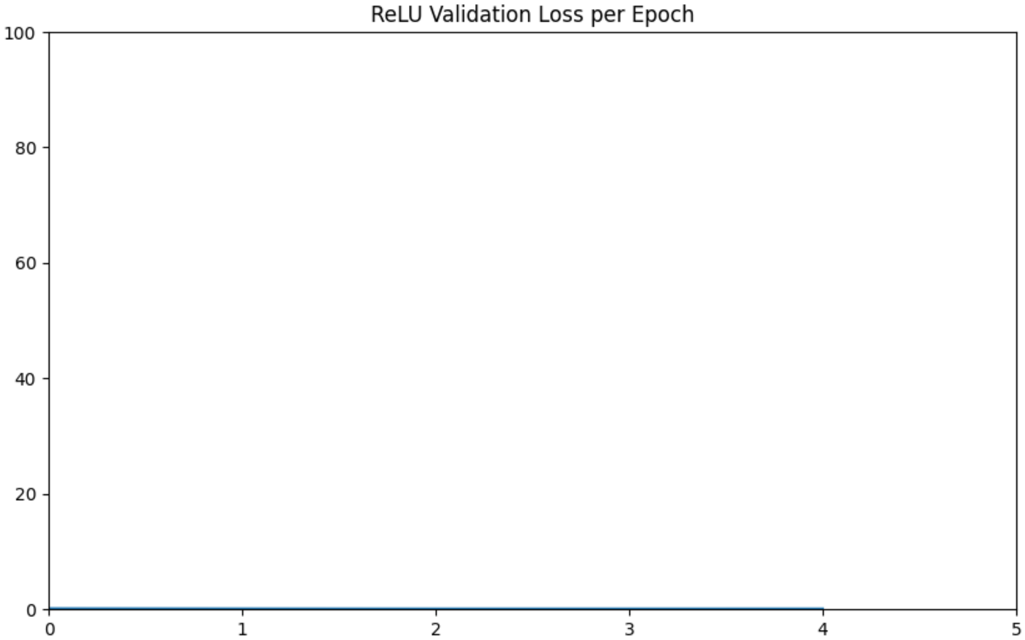


```
ReLU accuracy of epoch  0 : 97.37856984138489
ReLU accuracy of epoch  1 : 98.02857041358948
ReLU accuracy of epoch  2 : 98.60000014305115
ReLU accuracy of epoch  3 : 98.69285821914673
ReLU accuracy of epoch  4 : 98.80714416503906
```

ReLU Training Loss:

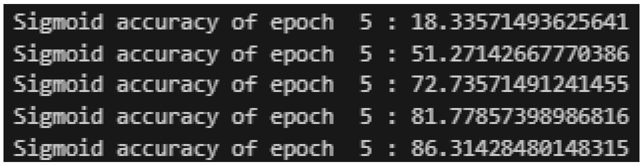


ReLU Validation Loss:



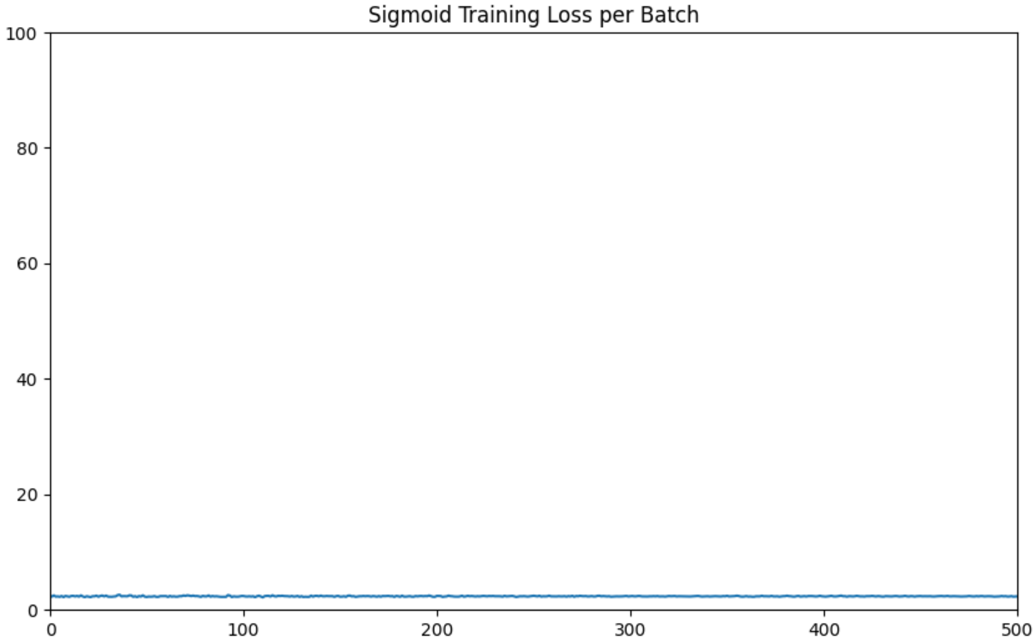
Sigmoid Testing Accuracy:

**The sigmoid accuracy goes from epoch 0 to 4, like ReLU. This was a typo in my code.

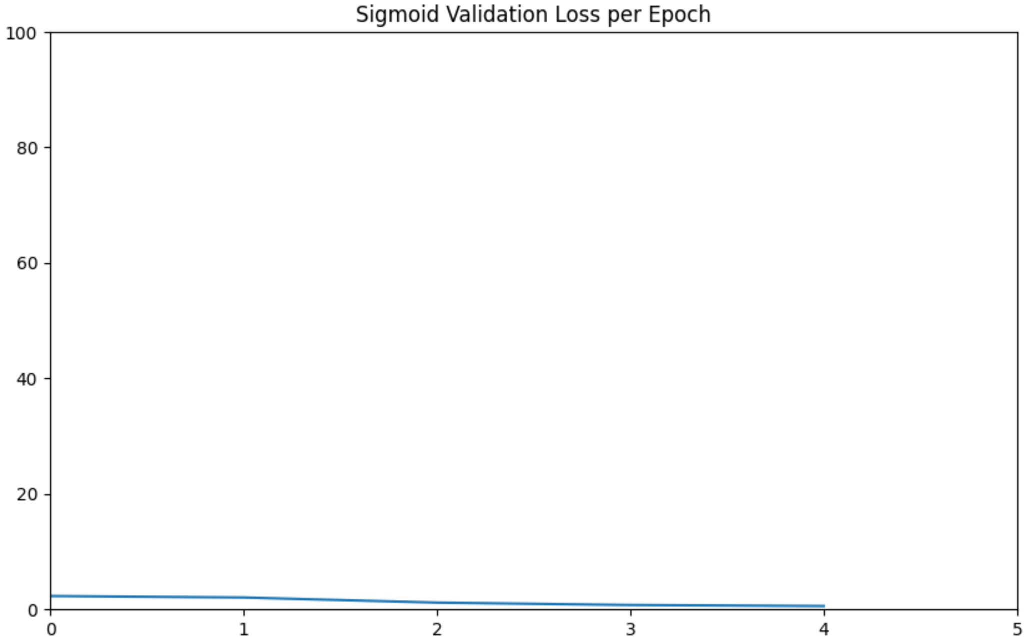


```
Sigmoid accuracy of epoch  5 : 18.33571493625641
Sigmoid accuracy of epoch  5 : 51.27142667770386
Sigmoid accuracy of epoch  5 : 72.73571491241455
Sigmoid accuracy of epoch  5 : 81.77857398986816
Sigmoid accuracy of epoch  5 : 86.31428480148315
```

Sigmoid Training Loss:



Sigmoid Validation Loss:


Sigmoid Validation Loss per Epoch