16-720 Homework 5: Neural Networks for Recognition

Austin Windham

November 30, 2023

1 Theory

Q 1.1

In order to prove that $softmax(x) = softmax(x + c)$, we need to substitute $x_i + c$ for x_i below.

$$softmax\left(x_i\right) = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{1}$$

$$softmax\left(x_i + c\right) = \frac{\mathrm{e}^{x_i+c}}{\sum_j \mathrm{e}^{x_j+c}} \tag{2}$$

$$= \frac{\mathrm{e}^{c}\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{c}\mathrm{e}^{x_j}} \tag{3}$$

$$= \frac{\mathrm{e}^{c}\mathrm{e}^{x_i}}{\mathrm{e}^{c}\sum_j \mathrm{e}^{x_j}} \tag{4}$$

$$= \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{5}$$

So, the calculations above illustrate that $softmax(x) = softmax(x + c)$.

Using $c = -\max x_i$ is a good idea because this means that the value in the exponential $x + c$ is less than or equal to 0, so the max value of the numerator is 1, which prevents the exponential from increasing rapidly and stops overflow in the calculations.

Q 1.2

The range of each element in the function is from 0 to 1, and the sum over all the elements is 1.

One could say that "softmax(x) takes an arbitrary real valued vector x and turns it into a probability distribution."

The first step of the function $s_i = \mathrm{e}^{x_i}$ finds the outcome frequency of each element in exponential form, The second step $S = \sum s_i$ gives the total outcome frequency of all the elements. And the last step $\frac{s_i}{S}$ normalizes the frequency and obtains the probability for each outcome.

Q 1.3

We have the equation below for going through a multi-layer neural network.

$$y_n = W_n x_n + b_n \tag{6}$$

Then when we apply this to the $n + 1$ layer, we get the following results.

$$\begin{aligned} y_{n+1} &= W_{n+1} y_n + b_{n+1} & (7)\\ &= W_{n+1} \left(W_{n+1} x_n + b_n \right) + b_{n+1} & (8)\\ &= W_{n+1} W_n x_n + W_{n+1} b_n + b_n & (9) \end{aligned}$$

We can see that the equation above can be grouped to form the linear format below.

$$y_{n+1} = W' x_n + b' \tag{10}$$

And, if we were to keep moving through the neural network layers, then this linear relationship would persist showing that multi-layer neural networks without a non-linear activation function are in fact equivalent to linear regression.

Q1.4

The gradient of $\sigma\left(x\right)$ is derived below.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) &= \frac{\mathrm{d}}{\mathrm{d}x}\frac{1}{(1+\mathrm{e}^{-x})} && (11)\\
&= \frac{-1 * -\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^2} = \frac{\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^2} && (12)\\
&= \frac{1}{1+\mathrm{e}^{-x}}\left(\frac{1+\mathrm{e}^{-x}-1}{1+\mathrm{e}^{-x}}\right) && (13)\\
&= \frac{1}{1+\mathrm{e}^{-x}}\left(1+\frac{-1}{1+\mathrm{e}^{-x}}\right) && (14)\\
&= \sigma(x)\left(1-\sigma(x)\right) && (15)
\end{aligned}$$

Q1.5

Given $y = Wx + b$ and $\frac{\partial J}{\partial y} = \delta$, the three solutions were derived below, and transposes were used to ensure that matrix multiplication is performed correctly.

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} = \delta^T x \tag{16}$$

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} = \delta^T W \tag{17}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b} = \delta \tag{18}$$

Q1.6

1. From the graph of the derivative of the sigmoid function below, we can see that the range is from 0 to 0.25. Since this range is low, when it is applied to many layers in the neural network, the problem of vanishing gradient is likely to arise.

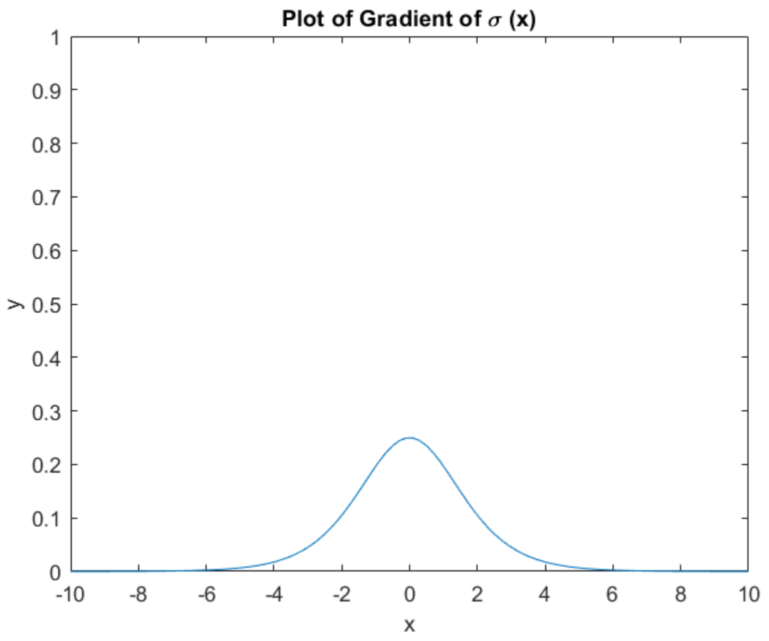


Figure 1: Sigma Gradient Plot

2. The range of the sigmoid function is from 0 to 1 and then range of the tanh function is from -1 to 1, so tanh might be preferred in scenarios where the data could have both positive and negative values. The tanh function might also be preferred because it helps mitigate the vanishing gradients problem since the function is centered at 0.

3. As you can see from the plot of the derivative of the tanh function below, the range is from 0 to 1. Since this is larger than the sigmoid function, the issue of vanishing gradients is less likely to happen.

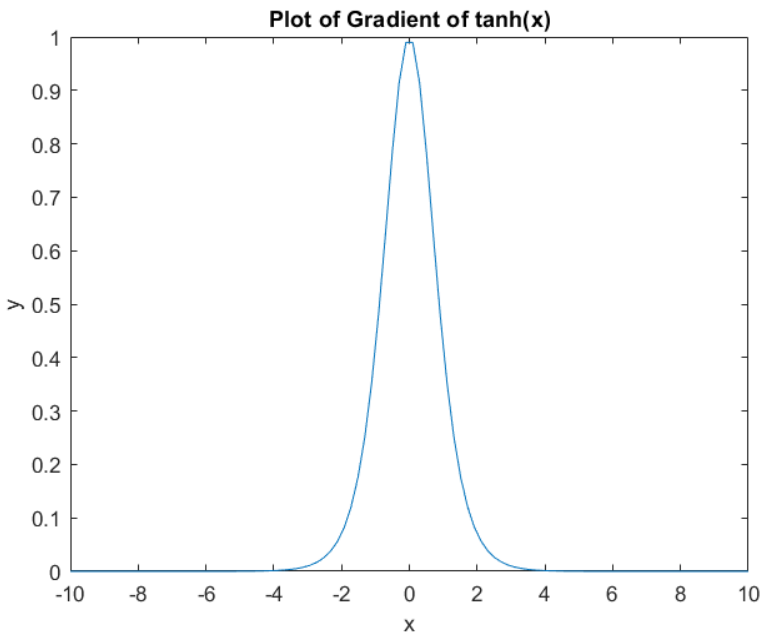


Figure 2: Sigma Gradient Plot

4. The steps to show how the tanh function can be written in terms of the sigmoid function are below.

$$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{19}$$

$$2\sigma(x) - 1 = \frac{2}{1+\mathrm{e}^{-x}} - \frac{1+\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}} = \frac{1-\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}} \tag{20}$$

And tanh is given below.

$$tanh(x) = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}} \tag{21}$$

So, the final result is

$$tanh(x) = 2\sigma(2x) - 1 \tag{22}$$

2 Implement a Fully Connected Network

2.1 Network Initialization

Q2.1.1

It is not a good idea to initialize a network with all zeros because all the inputs will then get set to zero when multiplied by the zero weights. This will then give us an output of all zeros. Since the outputs will all be the same, then the probability of what class the input belongs to will not vary also. Ultimately, initializing the network with all zeros causes zeros down the pipeline and does not allow the network to determine a differing probability distribution to select the most likely class.

Q 2.1.2 Code included in the submission.

Q 2.1.3

We initialize with random numbers in order to try to avoid obtaining the same results from each layer. And, we scale the initialization depending on layer size in order to try to prevent the variance from expanding too much during forward and back propagation.

Q 2.2 -2.5

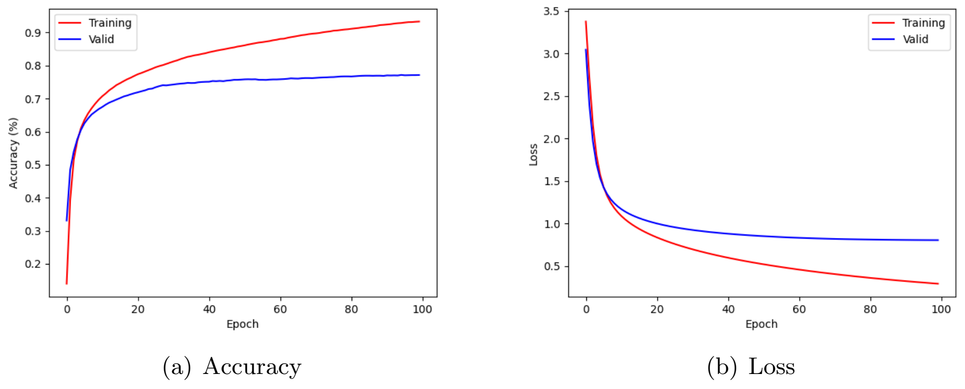
Code is included in the submission.

3 Training Models

Q 3.1

Code is included in the submission.

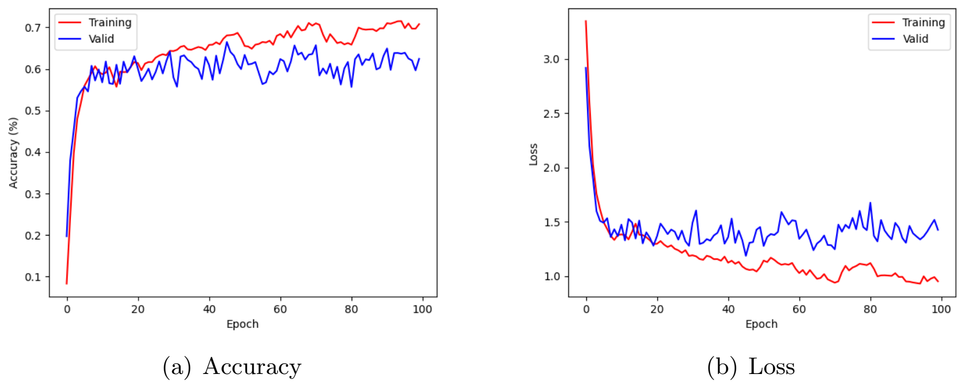
Q3.2 With 100 epochs, 32 batch size, 64 hidden layers, and a best learning rate of 0.0025, I was able to consistently achieve an accuracy over 75 %. The valid accuracy in this best learning rate case is 77.5 %. The training accuracy was approximately 94 %. The graphs of the loss and accuracy are below.



(a) Accuracy (b) Loss

Figure 3: Accuracy and Loss for Best Learning Rate

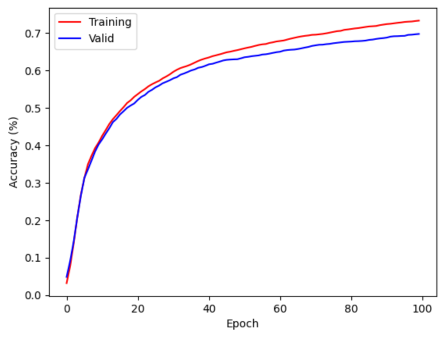
I then multiplied the learning rate by 10 for a learning rate of 0.025. The graphs of accuracy and loss can be seen below. The system was less accurate and had more loss with a Valid set accuracy of around 62.4 %. The graphs are also not smooth, and this is probably due to the fact that a faster learning rate is causing the network to not find a local minimum in time.



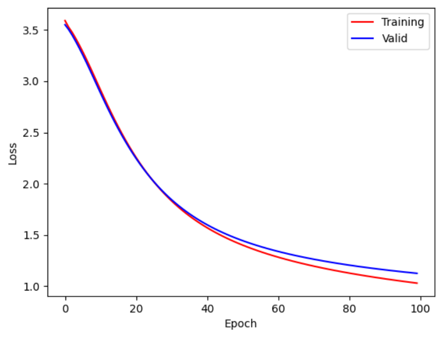
(a) Accuracy (b) Loss

Figure 4: Accuracy and Loss for Best Learning Rate X 10

I then divided the best learning rate by 10 for a learning rate of 0.00025. The results can be seen below. The accuracy and loss were both worse in this case also with the accuracy reaching 69.7 %. With the slower learning rate, the network was not able to reach the required 75 % accuracy within 100 epochs.



(a) Accuracy



(b) Loss

Figure 5: Accuracy and Loss for Best Learning Rate X 0.01

Q3.3

The visualization of the the trained and untrained layer 1 weights can be seen below. The untrained weights look unorganized like random noise because it is a random uniform distribution. But, after learning there appears to be some patterns. This shows how the network was able to take random weights and then learn to adjust the weights for better results.

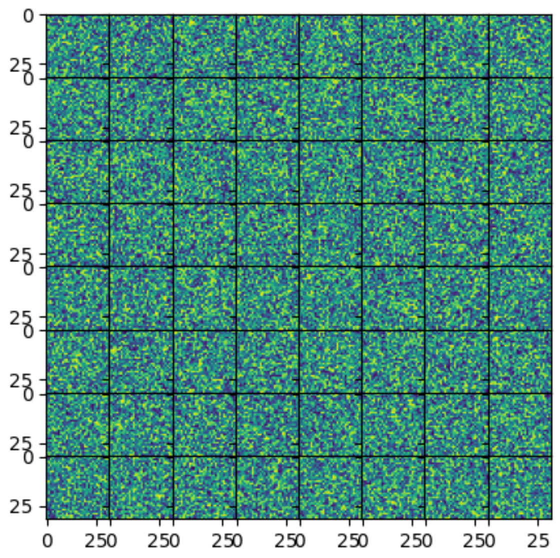


Figure 6: Weights Untrained

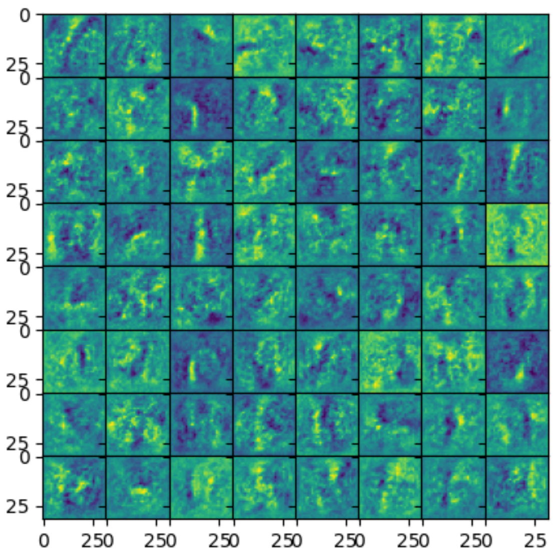


Figure 7: Learned Weights

Q3.4

The confusion matrix is below. We can see that the most likely confusion was the number "0" and the letter "O". This is understandable since these two are extremely similar. "S" and "5" were the next most confused pairs, which also makes sense since they have very similar shapes. After those two, "1" and "I" were confused the most, which they also have similar shapes. Overall, the confusion matrix make sense because the network is mainly confused on similar looking alphanumerical characters.

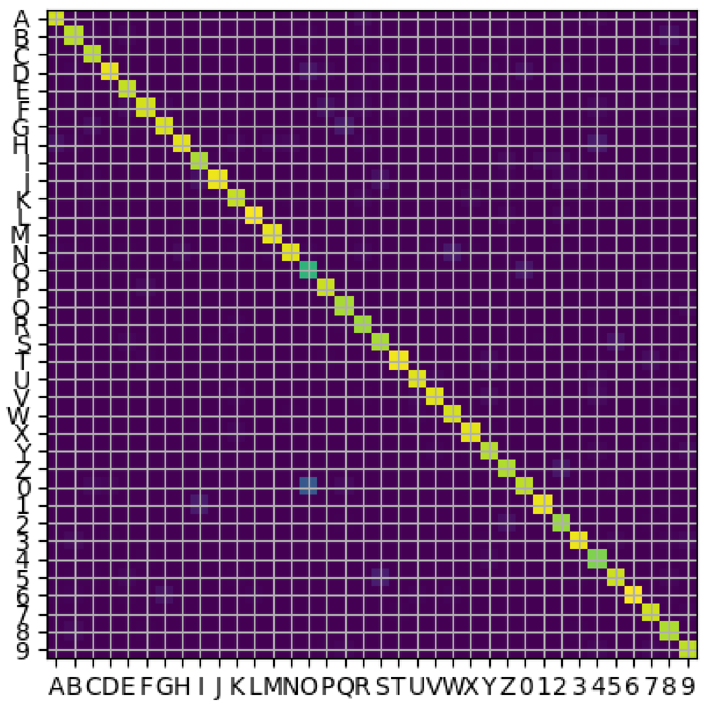


Figure 8: Confusion Matrix

4 Extract Text from Images

Q 4.1

Two big assumptions that our method makes are that an alphanumerical character is fully connected, and that these characters are not touching one another. These would create errors because we find characters by connected character pixels, and if the pixels weren't connected, the method might group the character into multiple characters. And if multiple characters are connected together, then the method might group two characters as one. An example of of an image with a characters that are unconnected and an example of an image with multiple characters touching one another are below.

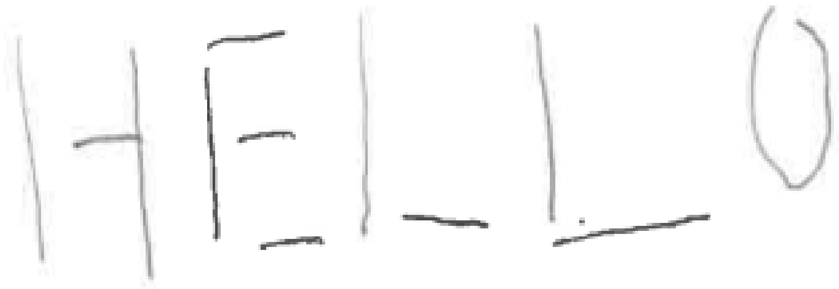


Figure 9: Unconnected Single Characters

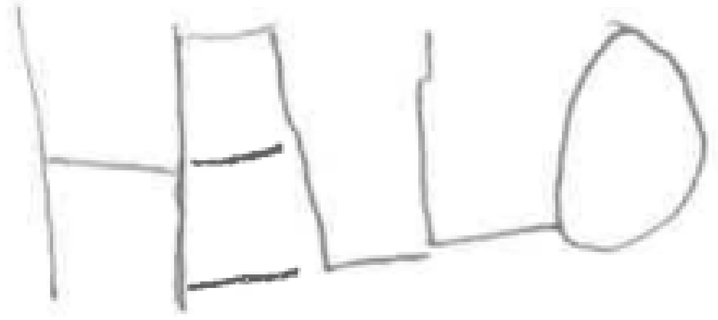


Figure 10: Multiple Touching Characters

Q 4.2 Code is included in the submission.

Q4.3

As you can see from the four figures below, the characters were detected correctly in the four different images. I have a variable called pad in my findletters function. If you chnage the variable, the red boxes will not be as tight. I showed the images with the pad at 0 because 4.4 worked better with that, but if you want a better visualization you can set the pad at 15.

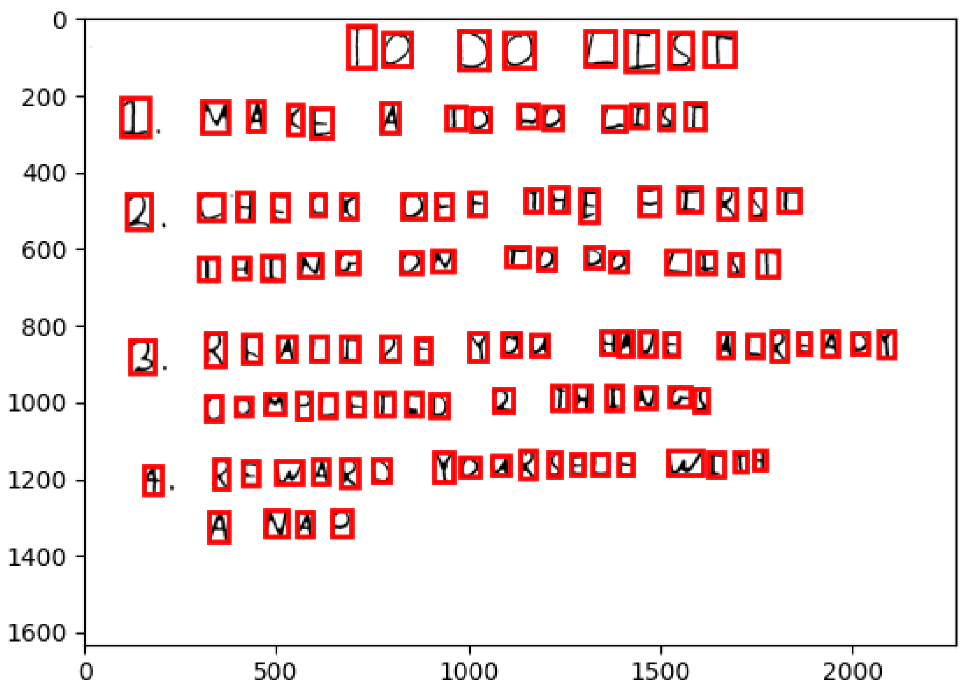


Figure 11: Character Boxing for List Picture

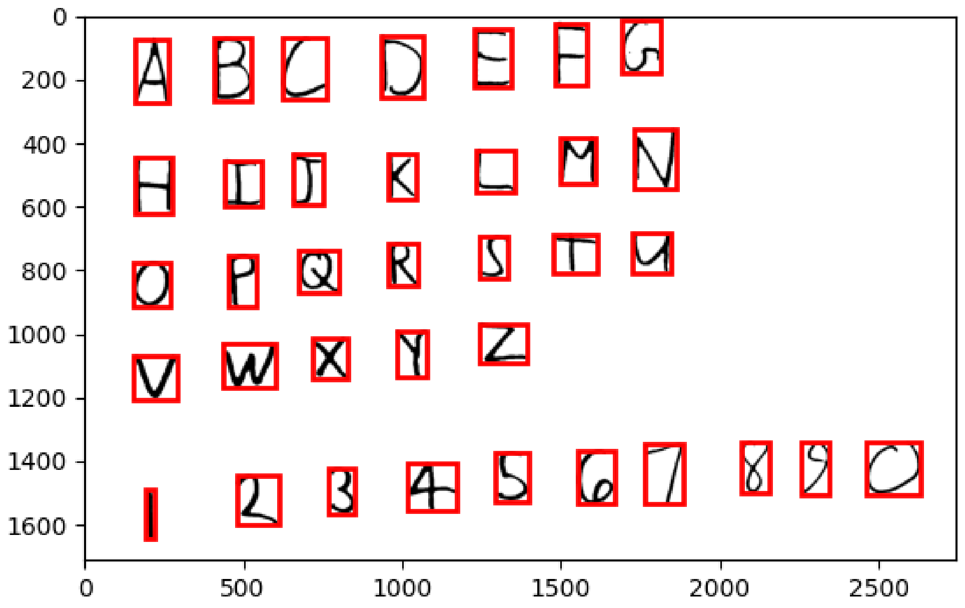


Figure 12: Character Boxing for Letter Picture

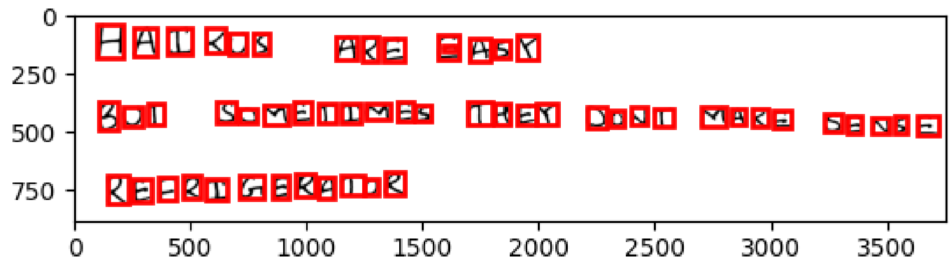


Figure 13: Character Boxing for Haiku Picture

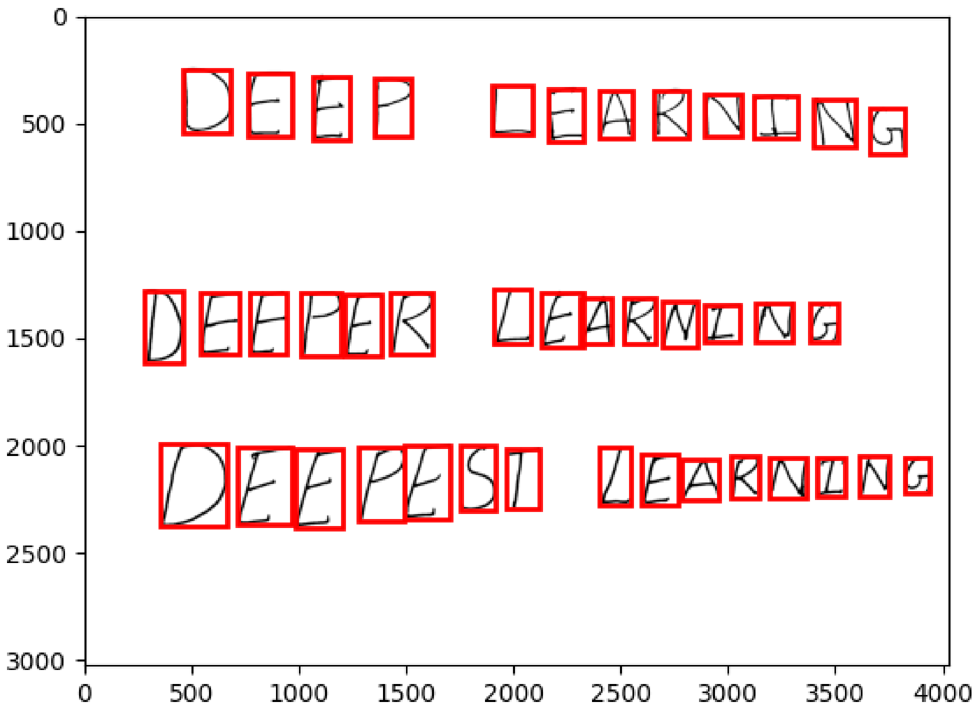
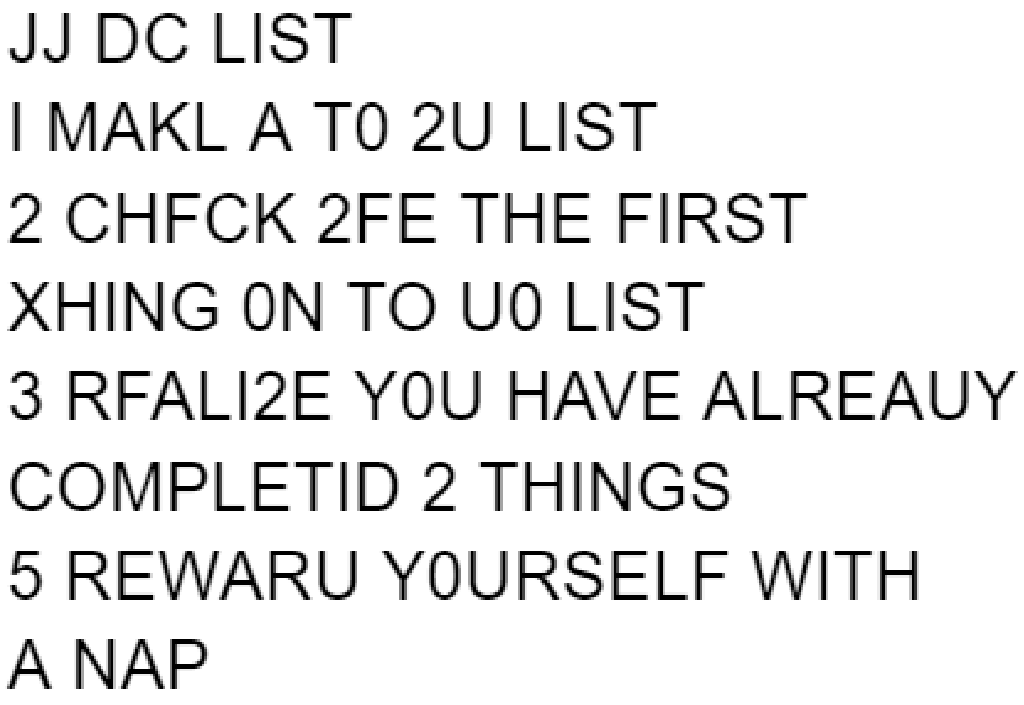


Figure 14: Character Boxing for Deep Picture

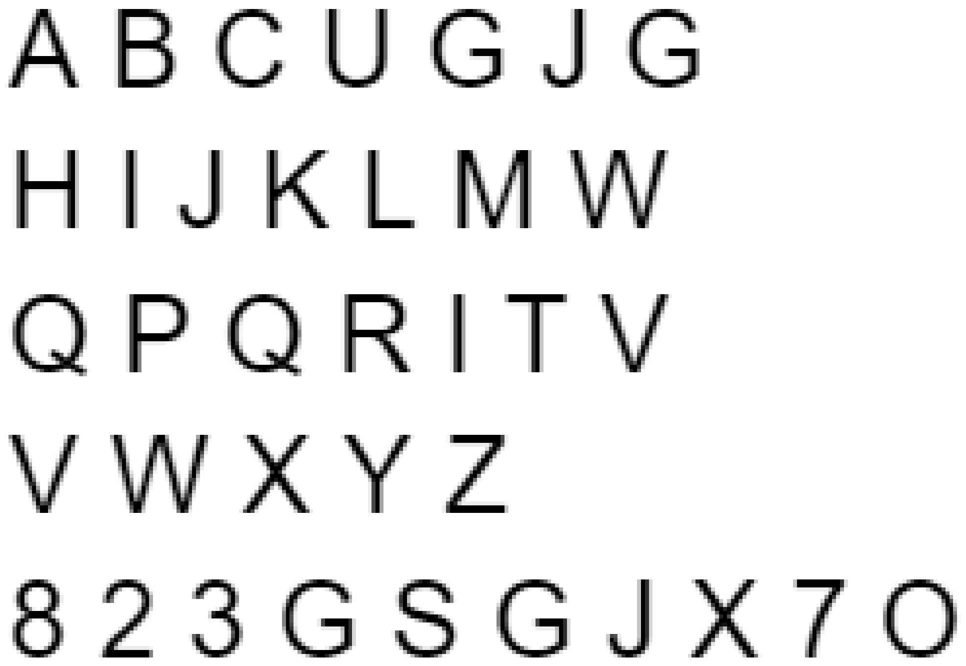
Q4.4

I printed the results in the terminal and then added spaces afterwards. Pictures of the predicted characters are below. As you can see, the neural network was able to correctly decipher most of the letters, but it still has a good deal of inaccuracy, which is expected since our network only reached a little over 75 % in question 3.



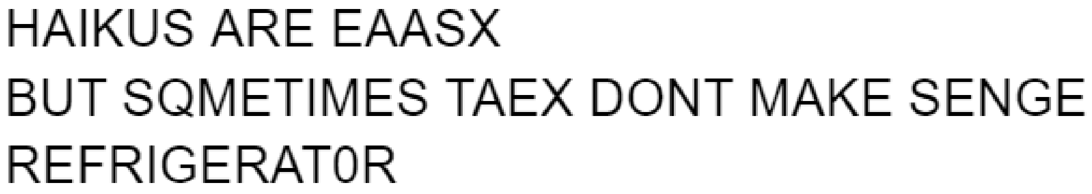
```
JJ DC LIST
I MAKL A T0 2U LIST
2 CHFCK 2FE THE FIRST
XHING 0N TO U0 LIST
3 RFALI2E Y0U HAVE ALREAUY
COMPLETID 2 THINGS
5 REWARU Y0URSELF WITH
A NAP
```

Figure 15: Output for List Picture



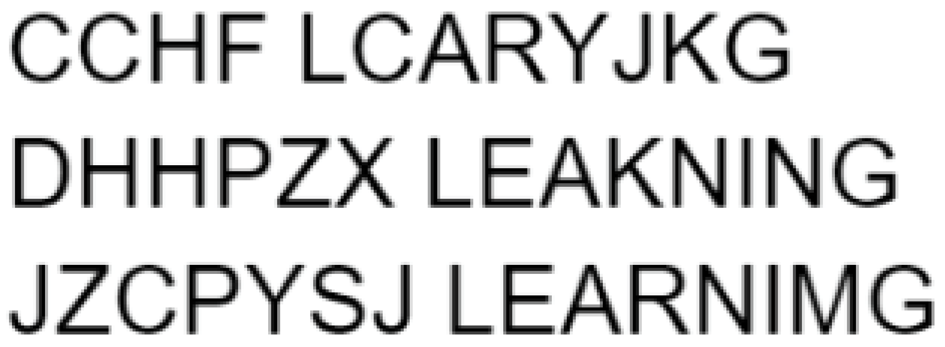
```
A B C U G J G
H I J K L M W
Q P Q R I T V
V W X Y Z
8 2 3 G S G J X 7 O
```

Figure 16: Output for Letters Picture



```
HAIKUS ARE EAASX
BUT SQMETIMES TAEX DONT MAKE SENGE
REFRIGERAT0R
```

Figure 17: Output for Haiku Picture



```
CCHF LCARYJKG
DHHPZX LEAKNING
JZCPYSJ LEARNIMG
```

Figure 18: Output for Deep Picture

5 Image Compression with Autoencoders

Q5.1

Code is included in the submission.

5.2 Training the Autoencoder

Q5.2

With the default parameters, the train loss curve can be seen below. The loss drops very quickly initially in the curve. The curve also is different than the best curve in question 3 because this curve has several bumps and does not look like a smooth function.

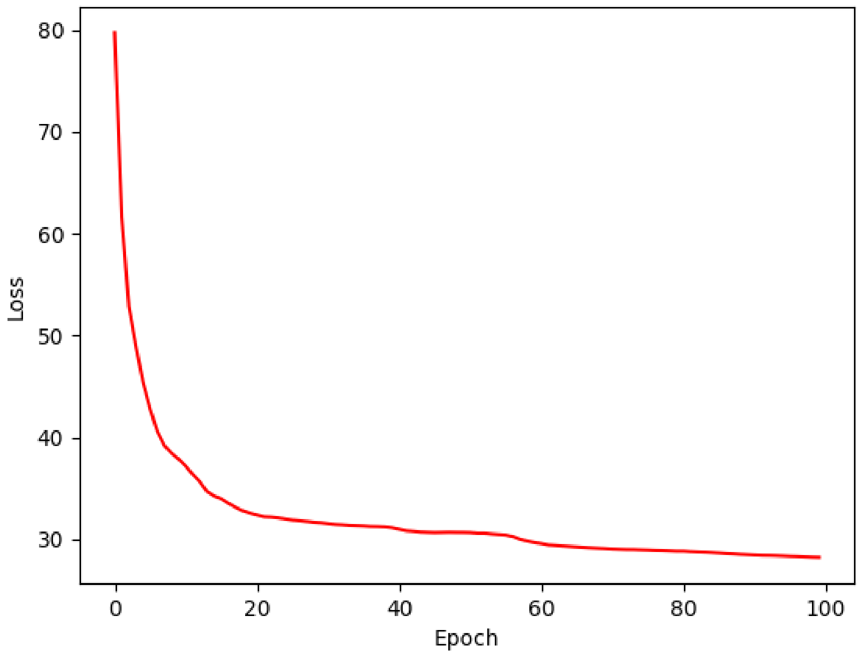


Figure 19: Loss Figure

5.3 Evaluating the Autoencoder

Q 5.3.1

The validation images and reconstructed images for two examples of classes A, B, 0, 6, and 8 are below. Someone can still decipher what the character in the reconstructed image are, but the images are much blurrier than the originals, which shows that the autoencoder was not able to completely reconstruct the characters.

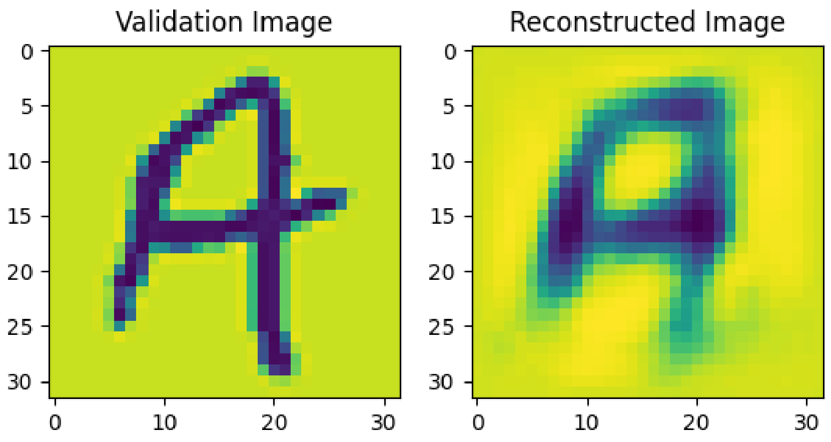


Figure 20: Class A

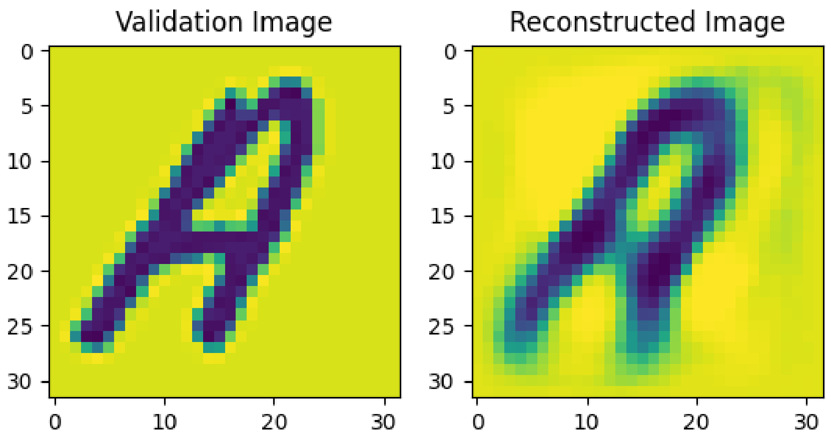


Figure 21: Class A

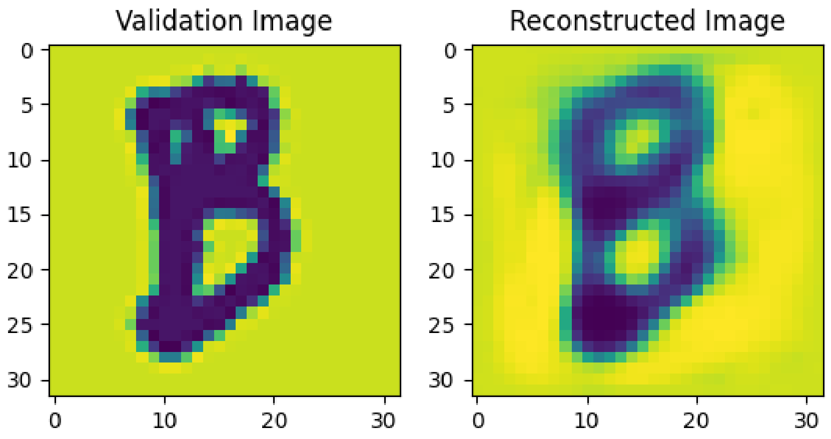


Figure 22: Class B

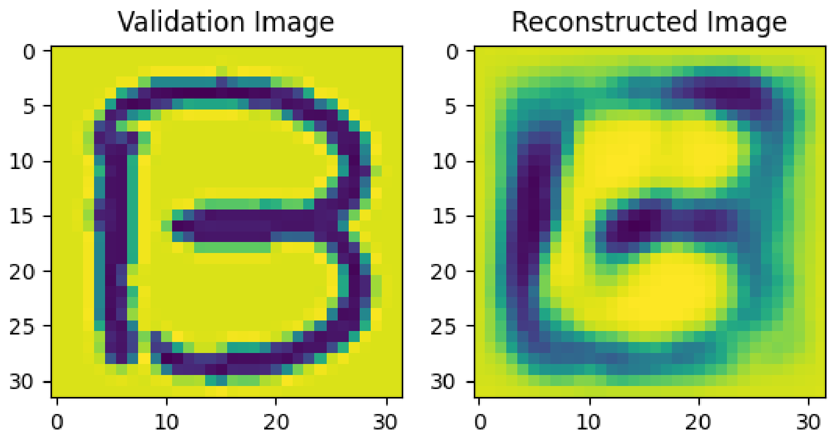


Figure 23: Class B

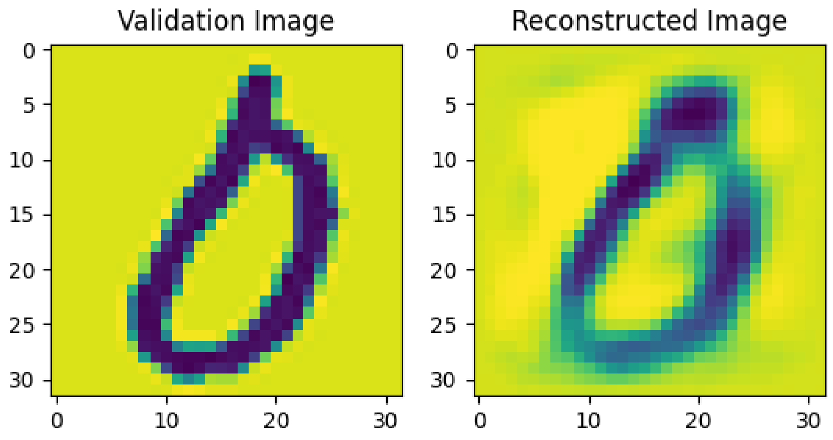


Figure 24: Class 0

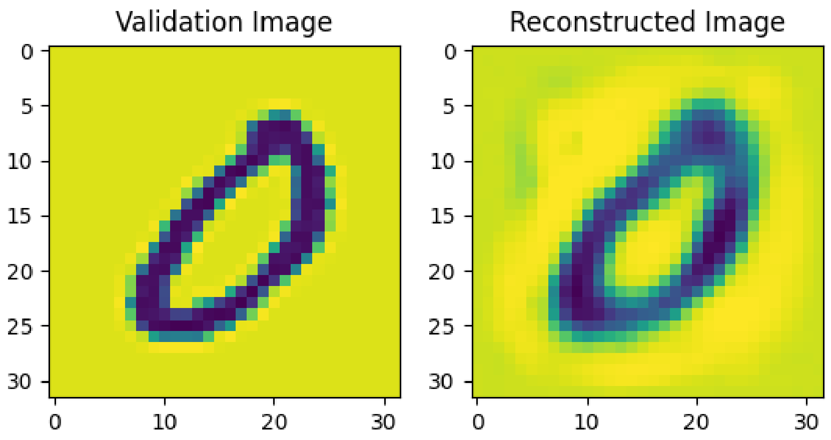


Figure 25: Class 0

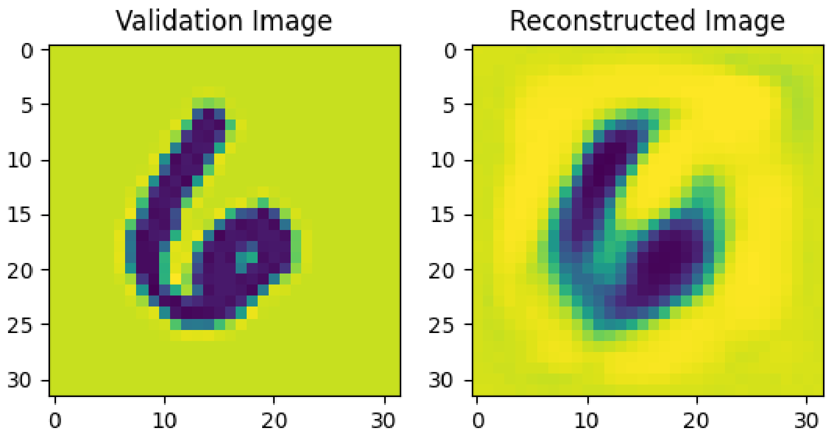


Figure 26: Class 6

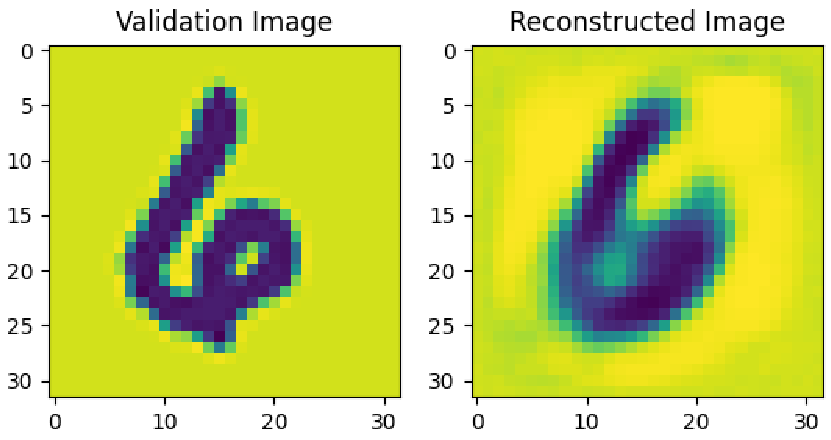


Figure 27: Class 6

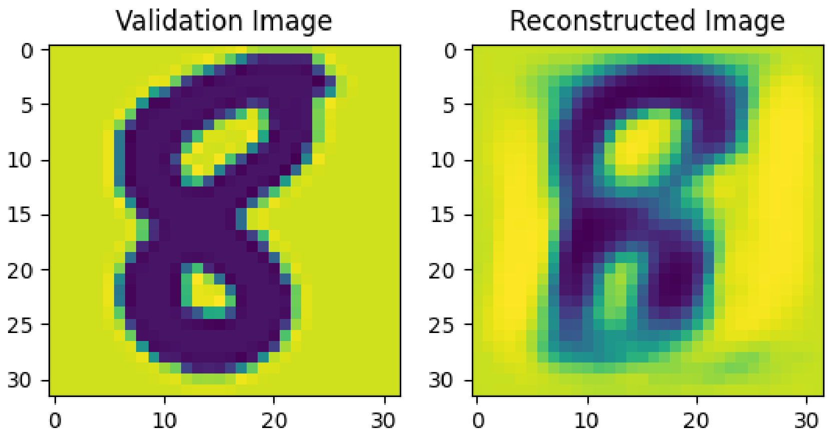


Figure 28: Class 8

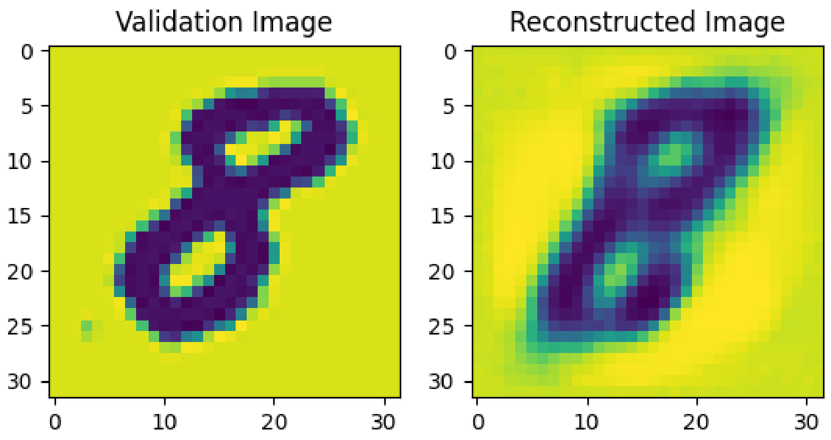


Figure 29: Class 8

Q5.3.2

From the autoencoder, the obtained PSNR value is 15.79. Code is included in the submission also.

6 PyTorch Extra Credit

I did not complete the extra credit.