Homework 5

Thomas Zhang

November 2023

Question 1

1.1)

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}}$$
$$softmax(x_i + c) = \frac{e^{x_i}e^{c}}{\sum_j e^{x_j}e^{c}}$$
$$softmax(x_i + c) = \frac{e^{x_i}e^{c}}{e^{c}\sum_j e^{x_j}}$$
$$softmax(x_i + c) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$
$$softmax(x_i + c) = softmax(x_i)$$

Using $c = -maxx_i$ helps because then the max x_i value evaluates to 1 in the numerator. While the numerator for the smallest x_i evaluates to some small number. If $c = 0$ then it is possible for the numerator to evaluate to some large number if the largest x_i is large.

1.2)

- The range of each element is the interval $(0, 1)$. The sum over all i of $softmax(x_i)$ is 1.
- Softmax takes an arbitrary real valued vector x and turns it into a vector of probabilities.
- $s = e^{x_i}$ amplifies the difference between large and small values. The summation: $S = \sum s_i$ is the total weights of the amplified values. $softmax(x_i) = \frac{1}{S}s_i$ normalizes the weights so that the sum of all weights equals one.

1.3)

If a network has no non-linear activation functions, then each of its perceptrons is represented by the equation:

$$y(x) = \mathbf{w}^t\mathbf{x} + w_0$$

And the activation function, $f(y)$, is some linear activation function. Since $f(y)$ is linear it can be represented in the form $f(y) = yu + b$ where a and b are some constant.

Hence the activation function with respect to the inputs is $f(y(x)) = a(\mathbf{w}^t\mathbf{x} + w_0) + b$. This can be rewritten as $f(y(x)) = \mathbf{c}^t\mathbf{x}+d$ where $\mathbf{c} = a\times\mathbf{w}$ and $d = aw_0+b$. Hence each layer of perceptron/activation function in a neural network can be reduced to a linear equation. Which means optimizing the model is the same as performing linear regression.

1.4)

$$
\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\frac{d\sigma(x)}{dx} &= \frac{d}{dx}\frac{1}{1+e^{-x}} \\
\frac{d\sigma(x)}{dx} &= -1(1+e^{-x})^{-2}\frac{d}{dx}(1+e^{-x}) \\
\frac{d\sigma(x)}{dx} &= -1(1+e^{-x})^{-2}(-e^{-x}) \\
\frac{d\sigma(x)}{dx} &= \frac{e^{-x}}{(1+e^{-x})}^2 \\
\frac{d\sigma(x)}{dx} &= \frac{1+e^{-x}-1}{(1+e^{-x})}^2 \\
\frac{d\sigma(x)}{dx} &= \frac{1+e^{-x}}{(1+e^{-x})}^2 - \frac{1}{(1+e^{-x})}^2 \\
\frac{d\sigma(x)}{dx} &= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\
\frac{d\sigma(x)}{dx} &= \sigma(x) - \sigma(x)^2
\end{aligned}
$$

1.5)

Let I be an identity matrix, then $I_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

$$
\begin{aligned}
\frac{dy_i}{dW_{j,k}} &= \frac{d}{dW_{j,k}}\sum_{l=1}^{d}(x_l W_{i,l}) + b_i \\
\frac{dy_i}{dW_{j,k}} &= I_{i,j}\sum_{l=1}^{d} I_{k,l}x_l \\
\frac{dy_i}{dW_{j,k}} &= I_{i,j}x_k
\end{aligned}
$$

$$
\begin{aligned}
\frac{dy_i}{dx_j} &= \frac{d}{dx_j}\sum_{k=1}^{d}(x_k W_{i,k}) + b_i \\
\frac{dy_i}{dx_j} &= \sum_{k=1}^{d} I_{j,k}W_{i,k} \\
\frac{dy_i}{dx_j} &= W_{i,j}
\end{aligned}
$$

$$
\begin{aligned}
\frac{dy_i}{db_j} &= \frac{d}{db_j}\sum_{k=1}^{d}(x_k W_{i,k}) + b_i \\
\frac{dy_i}{db_j} &= I_{i,j}
\end{aligned}
$$

$$
\begin{aligned}
\frac{dJ}{dW} &= \begin{bmatrix} \frac{dJ}{dW_{1,1}} & \frac{dJ}{dW_{1,2}} & \cdots & \frac{dJ}{dW_{1,d}} \\ \frac{dJ}{dW_{2,1}} & \frac{dJ}{dW_{2,2}} & \cdots & \frac{dJ}{dW_{2,d}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dJ}{dW_{k,1}} & \frac{dJ}{dW_{k,2}} & \cdots & \frac{dJ}{dW_{k,d}} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{1,1}} & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{1,2}} & \cdots & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{1,d}} \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{2,1}} & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{2,2}} & \cdots & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{2,d}} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{k,1}} & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{k,2}} & \cdots & \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dW_{k,d}} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \delta_i I_{i,1} x_1 & \sum_{i=1}^{k} \delta_i I_{i,1} x_2 & \cdots & \sum_{i=1}^{k} \delta_i I_{i,1} x_d \\ \sum_{i=1}^{k} \delta_i I_{i,2} x_1 & \sum_{i=1}^{k} \delta_i I_{i,2} x_2 & \cdots & \sum_{i=1}^{k} \delta_i I_{i,2} x_d \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{k} \delta_i I_{i,k} x_1 & \sum_{i=1}^{k} \delta_i I_{i,k} x_2 & \cdots & \sum_{i=1}^{k} \delta_i I_{i,k} x_d \end{bmatrix} \\
&= \begin{bmatrix} \delta_1 x_1 & \delta_1 x_2 & \cdots & \delta_1 x_d \\ \delta_2 x_1 & \delta_2 x_2 & \cdots & \delta_2 x_d \\ \vdots & \vdots & \ddots & \vdots \\ \delta_k x_1 & \delta_k x_2 & \cdots & \delta_k x_d \end{bmatrix} \\
&= \delta x^T
\end{aligned}
$$

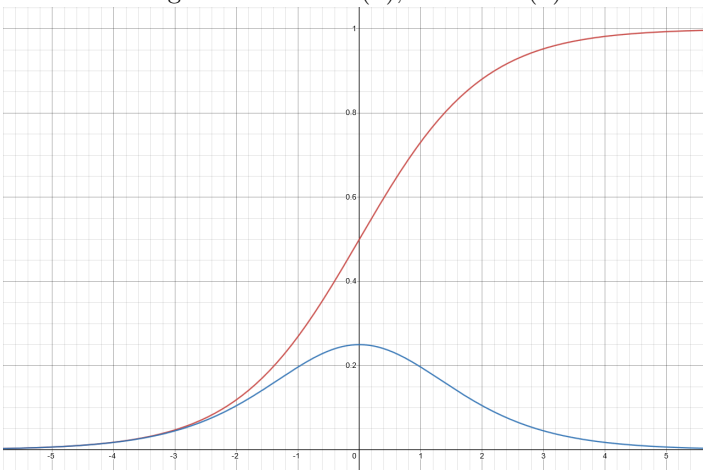
$$
\begin{aligned}
\frac{dJ}{dx} &= \begin{bmatrix} \frac{dJ}{dx_1} \\ \frac{dJ}{dx_2} \\ \vdots \\ \frac{dJ}{dx_d} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dx_1} \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dx_2} \\ \vdots \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{dx_d} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \delta_i W_{i,1} \\ \sum_{i=1}^{k} \delta_i W_{i,2} \\ \vdots \\ \sum_{i=1}^{k} \delta_i W_{i,d} \end{bmatrix} \\
&= W^T \delta
\end{aligned}
$$

$$
\begin{aligned}
\frac{dJ}{db} &= \begin{bmatrix} \frac{dJ}{db_1} \\ \frac{dJ}{db_2} \\ \vdots \\ \frac{dJ}{db_k} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{db_1} \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{db_2} \\ \vdots \\ \sum_{i=1}^{k} \frac{dJ}{dy_i}\frac{dy_i}{db_k} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{k} \delta_i I_{i,1} \\ \sum_{i=1}^{k} \delta_i I_{i,2} \\ \vdots \\ \sum_{i=1}^{k} \delta_i I_{i,k} \end{bmatrix} \\
&= \delta
\end{aligned}
$$

1.6)

1. The sigmoid function squishes values between 0 and 1. As a result even large changes in x, result in small changes in the output of the activation function. Additionally as x gets further from 0, the derivative of $\sigma(x)$ gets smaller and smaller, and the largest the derivative of $\sigma(x)$ can get is 0.25 at $x = 0$. Thus if multiple layers all use the sigmoid function as an activation function, the gradient becomes smaller and smaller via the derivative chain rule. This leads to the vanishing gradient problem.

Figure 1: Red = $\sigma(x)$, Blue = $\sigma'(x)$



2. The $tanh(x)$ function has an output range of $[-1, 1]$. The $\sigma(x)$ function has an output range of $[0, 1]$. This is preferable to the sigmoid function because it has both positive and negative output values. Which allows perceptrons to positively or negatively affect next layer. Additionally, the derivative of the $tanh(x)$ function has a maximum value of 0.5 at $x = 0$, which is twice as large as the maximum value of the sigmoid derivative. Hence, the rate at which the gradient shrinks across multiple layers is also slower.

3. The output range for the derivative of $tanh(x)$ is $[0, 1]$, which is 4 times as large as the output range of the sigmoid derivative $[0, 0.25]$. Additionally the output of $tanh(x)$ is 1 for $x = 0$. Put together, this means that a $tanh(x)$ activation function decreases the gradient less compared to $\sigma(x)$. Hence $tanh(x)$ has less of a vanishing gradient problem.

4. Given:

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

Then:

$$\begin{aligned}
2\sigma(2x) &= \frac{2}{1 + e^{-2x}} \\
2\sigma(2x) - 1 &= \frac{2}{1 + e^{-2x}} - 1 \\
2\sigma(2x) - 1 &= \frac{2}{1 + e^{-2x}} - \frac{1 + e^{-2x}}{1 + e^{-2x}} \\
2\sigma(2x) - 1 &= \frac{1 - e^{-2x}}{1 + e^{-2x}} \\
2\sigma(2x) - 1 &= tanh(x)
\end{aligned}$$

This shows that $tanh(x)$ is equivalent to scaling the $\sigma(x)$ by two fold in both the x and y direction, then shifting the scaled function down by 1.

Question 2

2.1

2.1.1

Initiating a network with all zeros results in a network that can only output 0. Additionally since all the perceptrons have 0 as their weights, none of deeper weights will be updated during learning since the gradient is δ times the output of the previous layer. However since the weights are all 0, the output of the previous layer will be all 0. Hence most of the weights won't update. If all the weights are the same, but not zero, then all the weights will be symmetrically updated resulting in the model only learning a single feature.

2.1.3

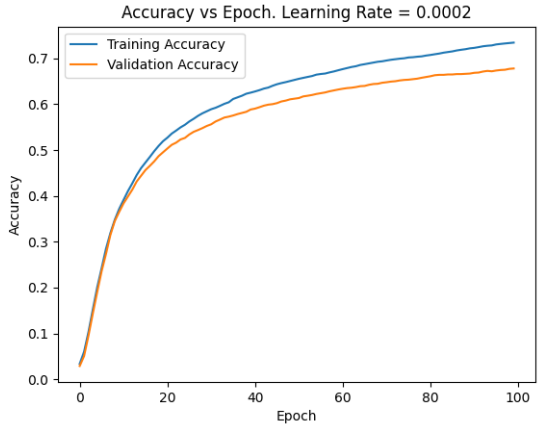
We initialize with random numbers so that the perceptrons do not all learn the same weights. A second reason to use random weights is that we increase the chance of reaching a local minima faster since the system is likely to be less stable than a system with identical weights are weights following some pattern. Additionally, we normalize the initial weights to minimize the chance of gradients vanishing. Without normalized initial weights, the activation values get closer and closer to 0 for deeper layers. Where as for normalized initial weights, the activation value distribution is similiar between layers.

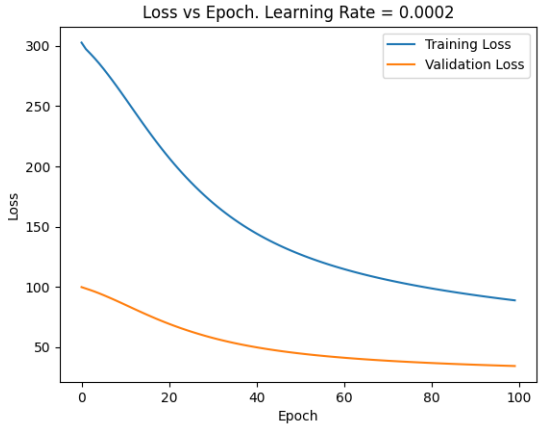
Question 3

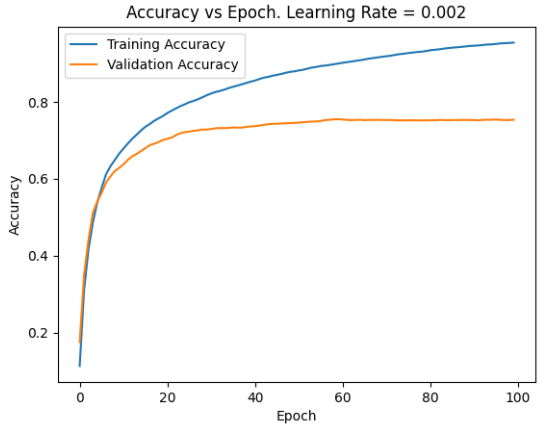
3.2

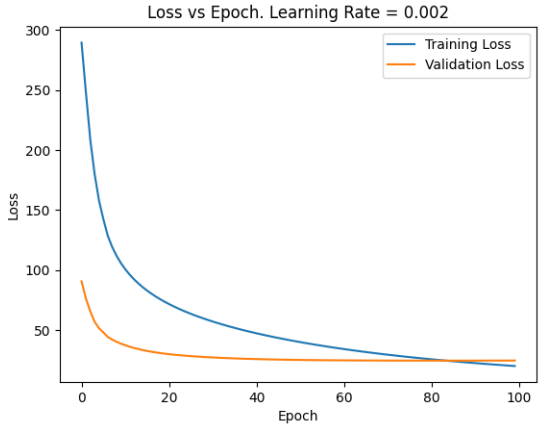
Loss and accuracy curves are 'smoother' with lower learning rate. When learning rate gets too high, the accuracy and loss curves become less well behaved and jump up and down.

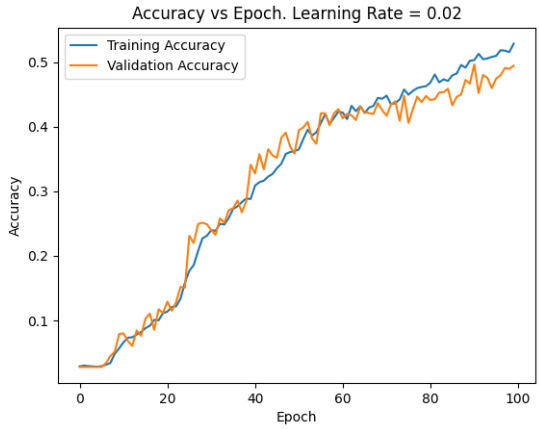
Best Validation Accuracy (learning rate = 0.002): 0.7505555555555555

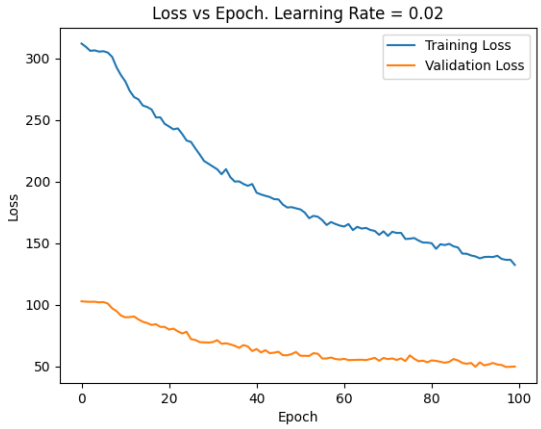






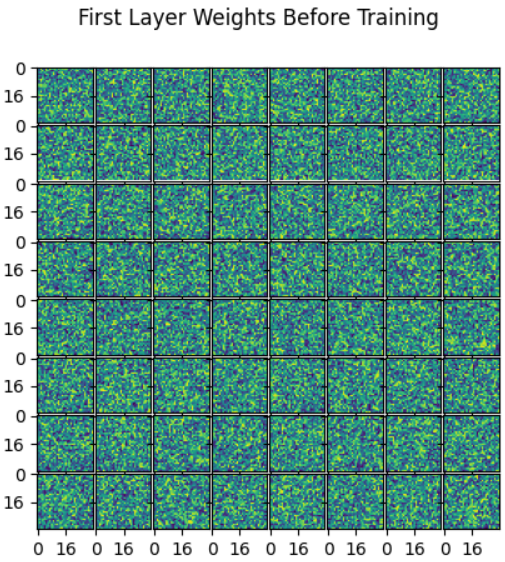


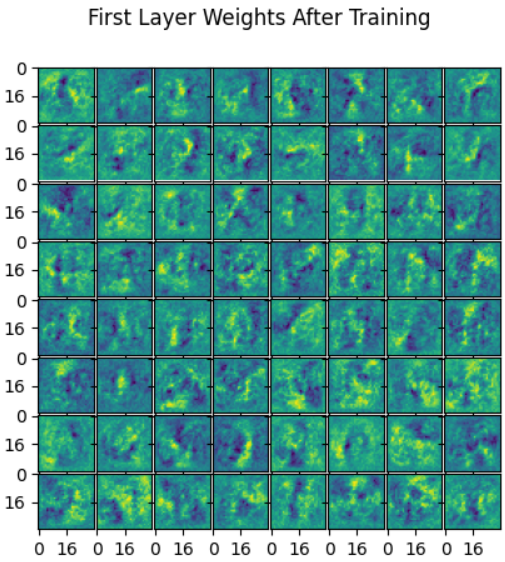




3.3

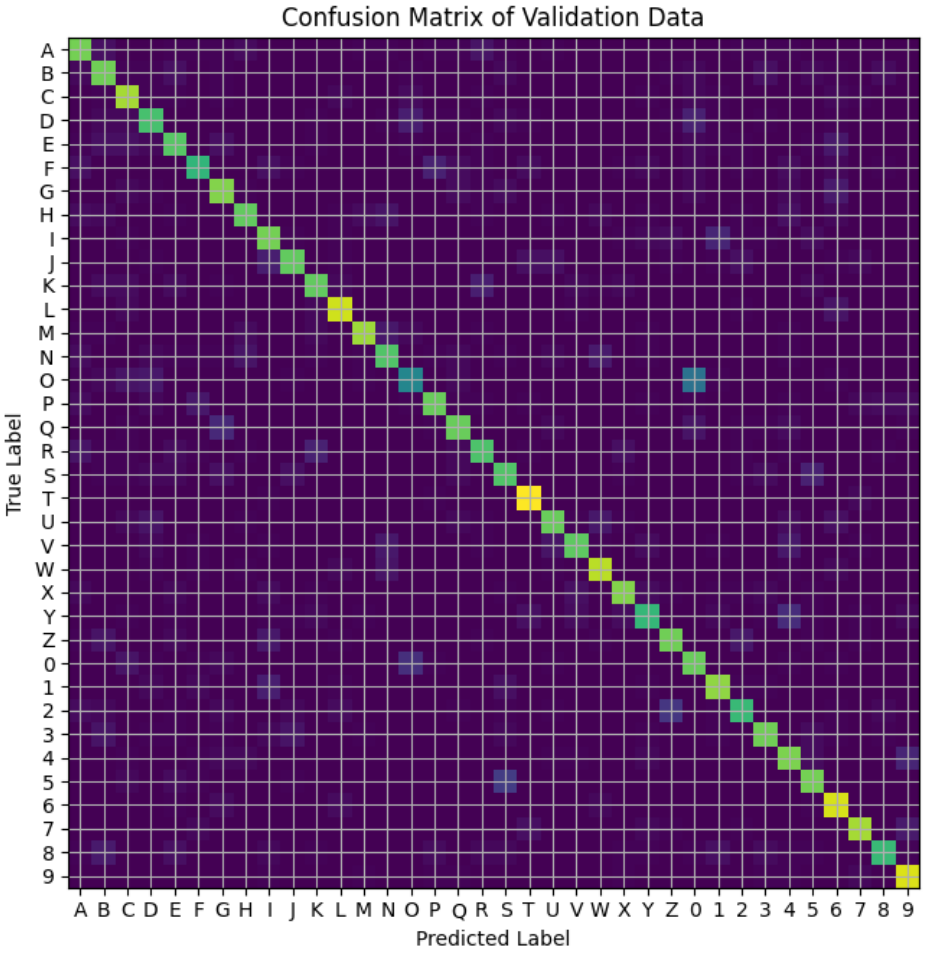
The learned weights have more of a pattern to them compared to the initial weights. The weights right after initialization look like static/noise. The weights after training have some kind of pattern to them, though it is not clear what features these patterns represent.





3.4

Two of the most commonly misclassified characters are: misclassifying 'o' as 0 or vice versa. and misclassfiying 's' as 5 or vice versa. This makes sense as in both cases, the two characters are very similar in appearance to each other.



Question 4

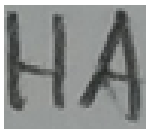
4.1

The first big assumption it makes is that each individual character will be separated by some amount of space. The second big assumption it makes is that a single character will be fully connected.

Potential fail cases:

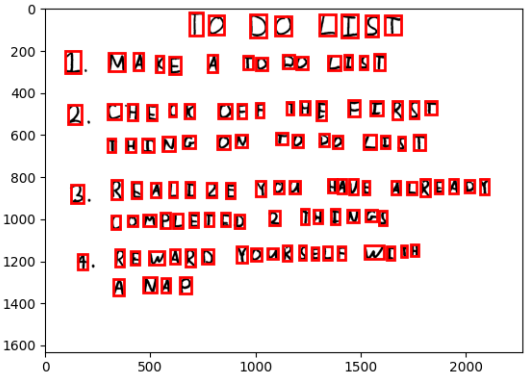
The E is a potential fail case because the middle line in the E is disconnected from the rest, so the method may recognize this as two separate letters.

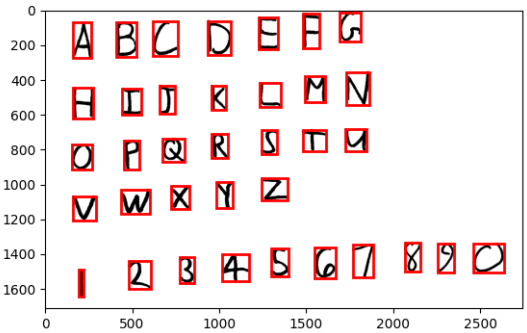
The HA is a potential fail case because they are very close, so during the blurring step these two characters may become connected and as a result be recognized as a single character rather than 2 separate ones.

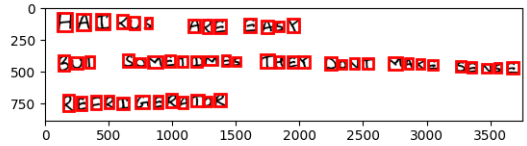


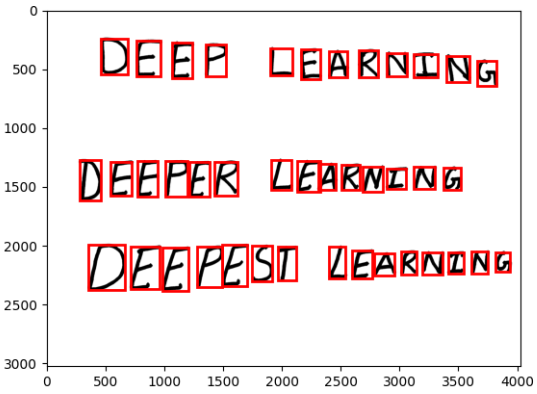
4.3

The find letters function is fully accurate on the given images.









4.4

Output for image 1
T0 D0 LIST
I NAXE A TO DO LIST
Z CHFCE DF8 7H1 FIRST THING QN TO D0 LIST
3 R5ALIZE YOU HAVE RLREADT COMPLETID 2 YHINGS
9 REWARD YOURSELF WI9B A NAP

Output for Image 2
1 B C D E F G
H I Y K L M N
0 P Q R 5 T U
V W X Y Z
1 X 3 4 S 6 7 8 9 0

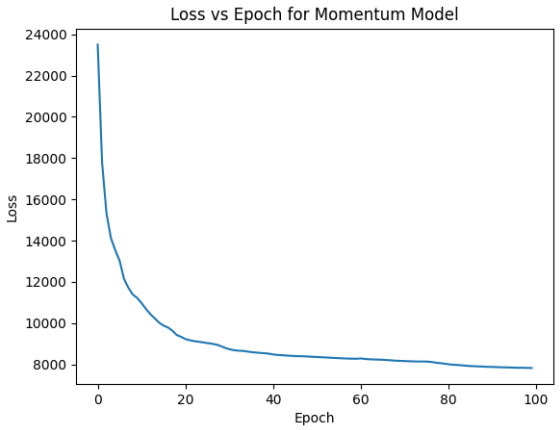
Output for Image 3
HAIKUS ARE EASY
BUT SQMETIMES T8EY DONT MAKE SENQE
REFRIGERATOR

Output for Image 4
DEEP LEARNING
DEEPER LEARNING
DE1PE5F LEARNIN6

Question 5

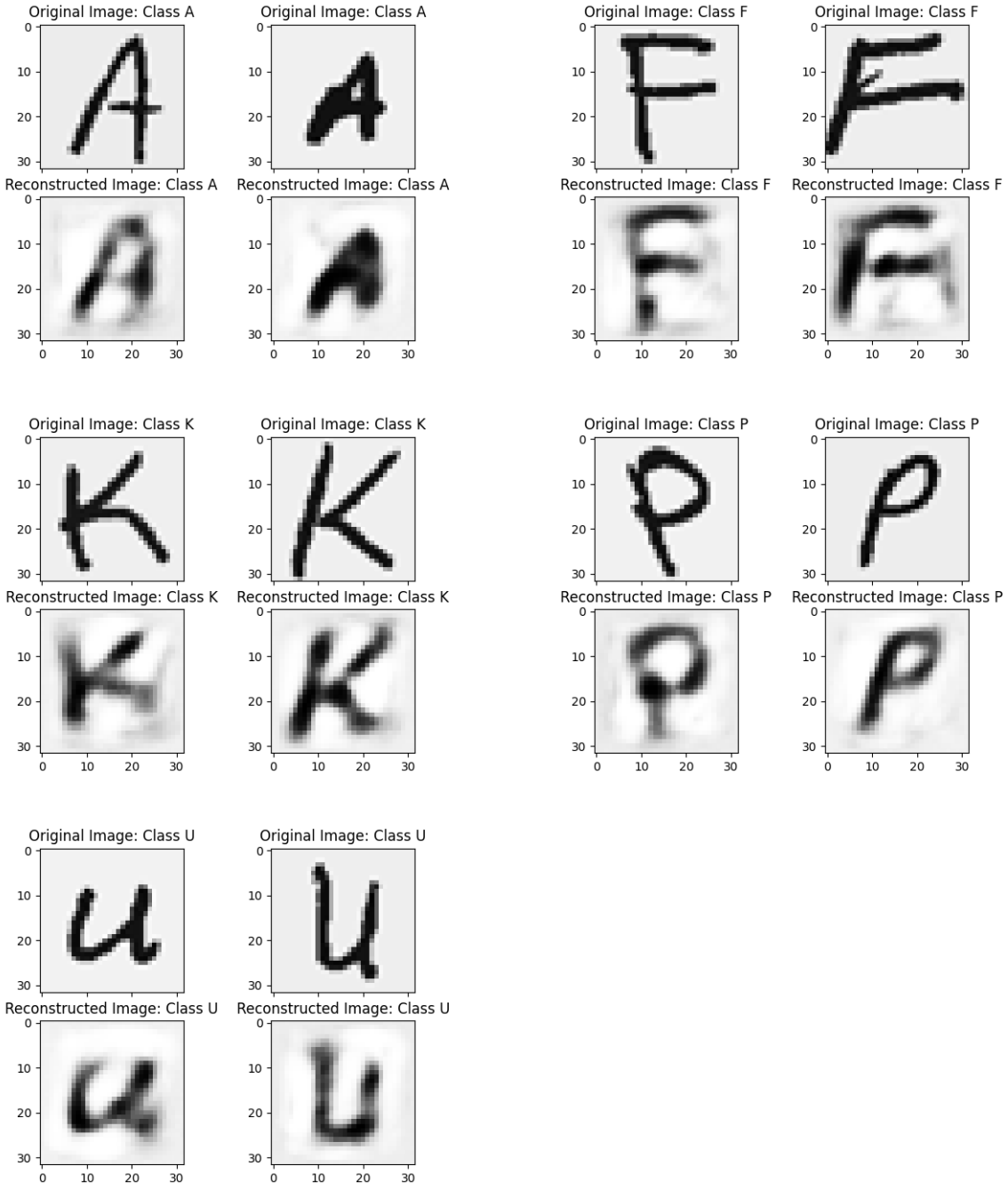
5.2

The rate at which loss decreases is very drastic at the start of training. However as the loss gets closer to a local minimum, the rate at which loss decreases slows down by alot.



5.3.1

Compared to the original images, the reconstructed images tend to be more blurry. Additionally, the reconstructed images seem to be slightly 'brighter' near the center of the image.

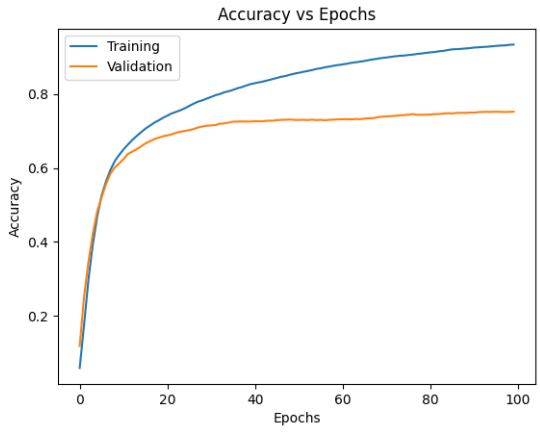


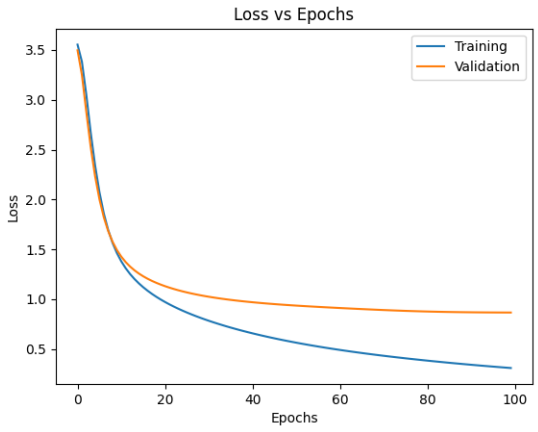
5.3.2

Mean PSNR accross all validation data: 15.92313048932615

Question 6

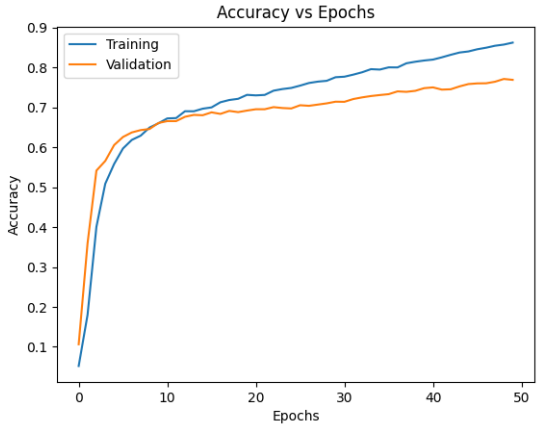
6.1.1

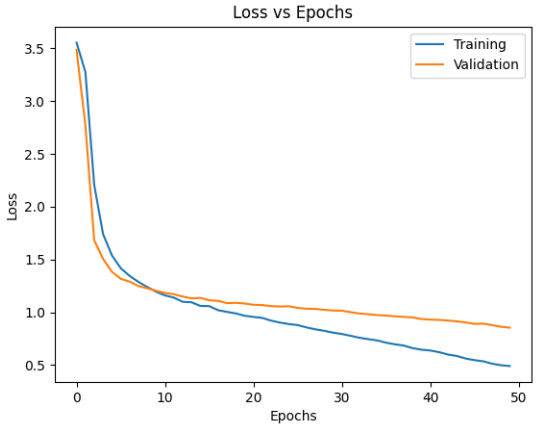




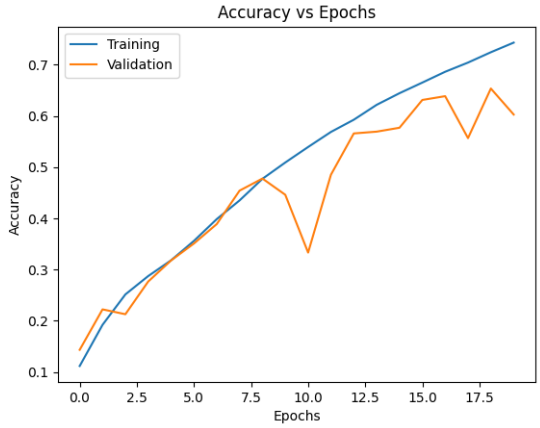
6.1.2

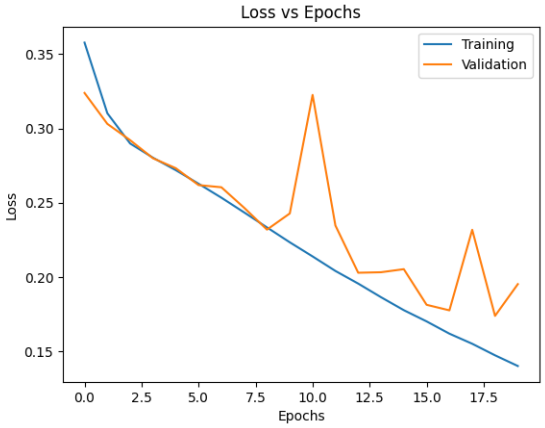
The pytorch CNN model seems to perform slightly better than the fully connected model. This can be seen in the loss and accuracy curves. The validation accuracy/loss split from the training accuracy/loss is less severe in the CNN model compared to the fully connected model. This means that the CNN model is over fitting less than the fully connected model.





6.1.3

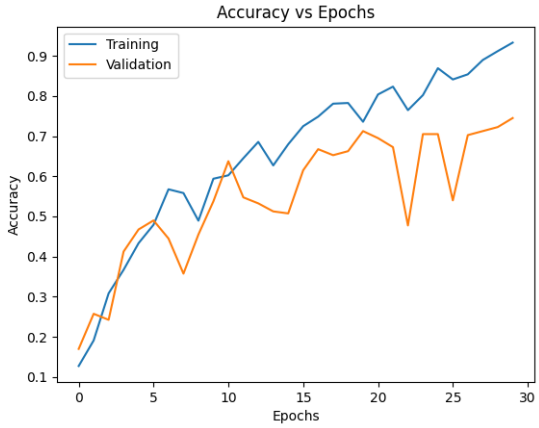


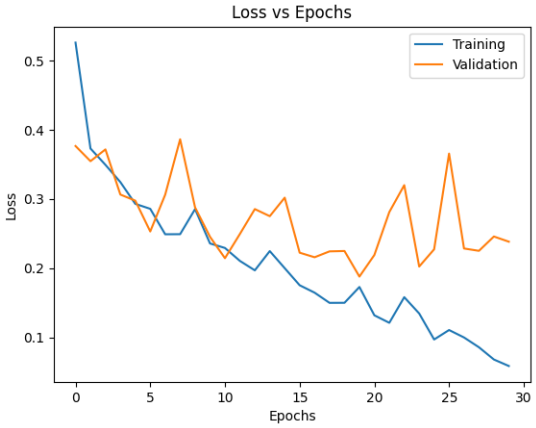


6.1.4

The final accuracy of the CNN model was 74.5%.

The CNN model was able to achieve higher accuracy on the validation data then the model we used in HW1. While the accuracy of the model did trend upwards during training, it was variable and would sometimes drop significantly despite training loss decreasing.





6.2

The custom CNN model architecture was difficult to properly tune for the dataset. The training set and test set accuracy began to split off around 35% and as a result, the model begins to overfit on the training data. Dropout was used to attempt to remedy this however, the overfitting was not resolved. Further tuning of the custom cnn model is necessary to create a model that can predict well.

The squeeze model was implemented however, I was unable to tune the parameters of the squeeze model to acquire more than 10% accuracy.

Custom Architecture Loss and Accuracy: