Q1.1

Evaluating softmax (x+c) we get:

$$\text{softmax}(x_i + c) = \frac{e^{x_i + c}}{\sum_{0}^{j} e^{x_j + c}}$$

$$= \frac{e^{c} . e^{x_i}}{e^{c} . \sum_{0}^{j} e^{x_j}}$$

$$= \frac{e^{x_i}}{\sum_{0}^{j} e^{x_j}}$$

Therefore, softmax(x) = softmax(x+c). Hence we can say that softmax is invariant to translation.
To avoid overflow or underflow during calculation, we want the values to stay between 0 and 1. For this, the values given to the exponential should be between -infinity and zero. We can achieve this by using $c = - \max x_i$

Q 1.2

- Range: (0,1]
  Sum: 1
- One could say that "*softmax takes an arbitrary real valued vector x and turns it into a* probability distribution".
- Step 1: Takes values and maps it in the positive space, for each element in x
  Step 2: Normalization step, where the total value of all elements is calculated
  Step 3: Output the probability of each element in x

Q 1.3

In a neural network without activation functions, each layer can be represented by the equation for a linear transformation:

$$Output = Weights \times Inputs + Bias$$

For a single-layer network without activation, the output can be expressed as:

$$Output = Weights_1 \times Inputs_1 + Bias_1$$

If we have multiple layers, the output of the previous layer becomes the input to the next layer:

$$Output_{layer2} = Weights_2 \times Inputs_{layer1} + Bias_2$$

This process continues through subsequent layers. Therefore, the entire network without non-linear activations behaves as a series of linear transformations stacked upon one another, resembling a linear regression model.

Q1.4

The gradient of the sigmoid function will be:

$$\frac{d}{\mathrm{dx}}\left(\frac{1}{1+e^{-x}}\right)=\frac{d}{\mathrm{dx}}\left(\frac{e^{x}}{1+e^{x}}\right)$$

$$=\frac{(1+e^{x}).e^{x}-e^{x^{2}}}{(1+e^{x})^{2}}$$

$$=\frac{e^{x}}{(1+e^{x})^{2}}$$

$$=\frac{e^{x}}{(1+e^{x})}\cdot\left(\frac{1+e^{x}-e^{x}}{1+e^{x}}\right)$$

$$=\frac{e^{x}}{(1+e^{x})}\cdot\left(1-\frac{e^{x}}{1+e^{x}}\right)$$

$$=\sigma(x)\,.\,(1-\sigma(x)\ )$$

Q1.5

$$y_j = \sum_{1}^{d} x_i . W_{\text{ij}} + b_j$$

$$\frac{d\, y_j}{d\, x_i} = W_{\text{ij}} \qquad \frac{d\, y_j}{d\, W_{\text{ij}}} = x_i \qquad \frac{d\, y_j}{d\, b_j} = 1$$

$$\frac{d\, J}{d\, y} = \delta$$

$$\frac{d\, J}{d\, W_{\text{ij}}} = \frac{d\, J}{d\, y_j} \cdot \frac{d\, y_j}{d\, W_{\text{ij}}} = \delta_j . x_i$$

$$\frac{d\, J}{d\, x_i} = \frac{d\, J}{d\, y_j} \cdot \frac{d\, y_j}{d\, x_i} = W_{\text{ij}} .\ \delta_j$$

$$\frac{d\, J}{d\, b_j} = \frac{d\, J}{d\, y_j} \cdot \frac{d\, y_j}{d\, b_j} = \delta_j$$

Therefore,

$$\frac{d\, J}{d\, W} = \delta^T . x$$

$$\frac{d\, J}{d\, x} = W . \delta$$

$$\frac{d\, J}{d\, b} = \delta$$

Q1.6

1. The sigmoid function maps the output of a perceptron between 0 and 1. Hence, a large range of inputs are fit into a very small space, reducing the gradient for changes in the input. Hence, with many sigmoid activation layers, the gradient will tend to zero when multiplying smaller values.
2. Sigmoid output range: $0 \rightarrow 1$
   Tanh output range: $-1 \rightarrow 1$
   Since the range of tanh is larger, the gradient of it is larger as well. Further, it is symmetric which would allow for faster convergence.
3. The gradient of tanh at 0 is 1, whereas that of the sigmoid function is about 0.25, which diminishes as it goes away from zero. Thus, due to its larger gradient, tanh has less of a vanishing gradient problem.
4. $tanh(x) = 2\sigma *(2x) - 1$

Q 2.1.1

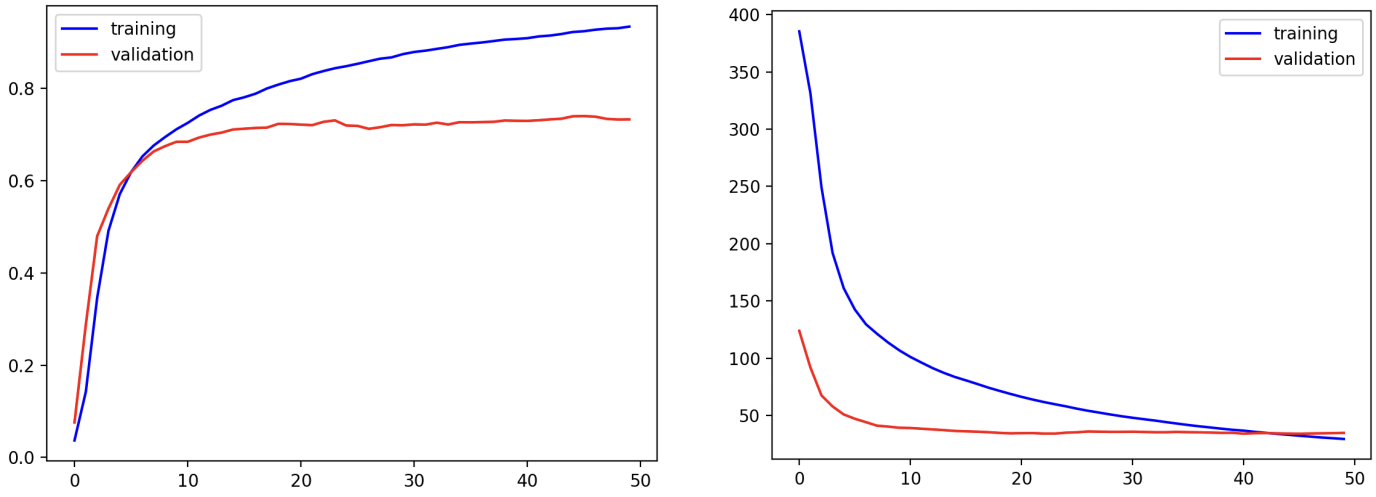
Initializing a network with all zero weights will introduce problems in training. It will cause neurons in each layer to learn the same features, reducing the network's ability to capture the complexity of the data. Additionally, this initialization will result in zero gradients during backpropagation, causing the network to fail to update its weights, a phenomenon known as the vanishing gradient problem. Hence, a network initialized with all zeros, even after training, will likely output the same values for every input and will fail to learn meaningful representations from the data.

Q 2.1.3

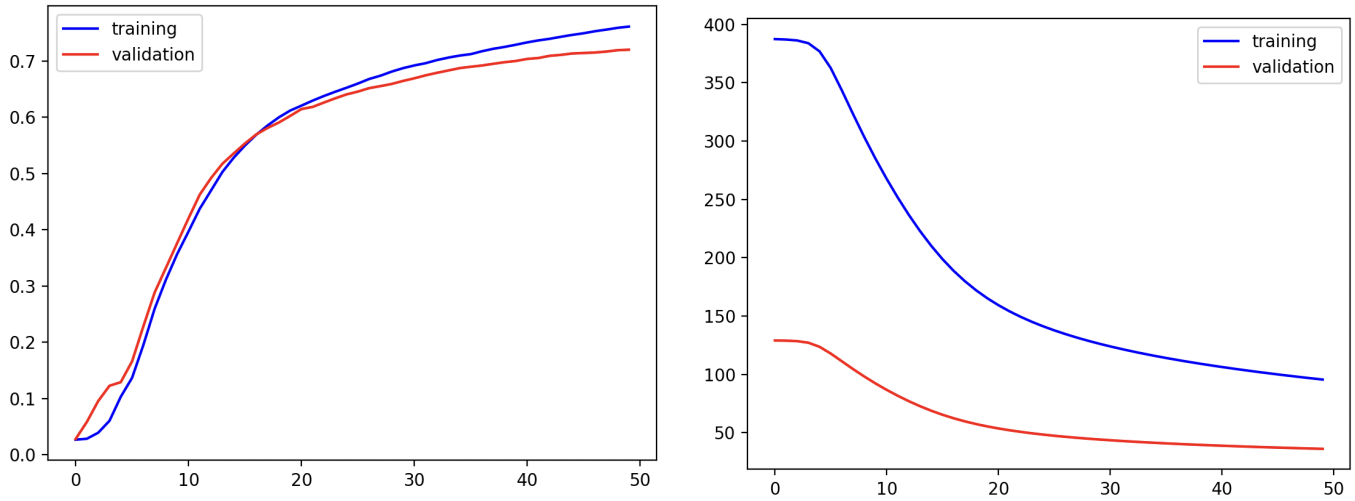
Random initialization breaks symmetry amongst neurons and lets each neuron learn unique features within a layer. It also makes it easier to find an optimal set of weights by preventing the network from getting trapped in a local minima.

Scaling the initialization depending on layer size ensures that the variance of outputs of each neuron is roughly the same. Without that, the output will depend on the number of inputs to the neuron, which can lead to saturation of the activation function and slow learning in case of neurons with very few or very large inputs.

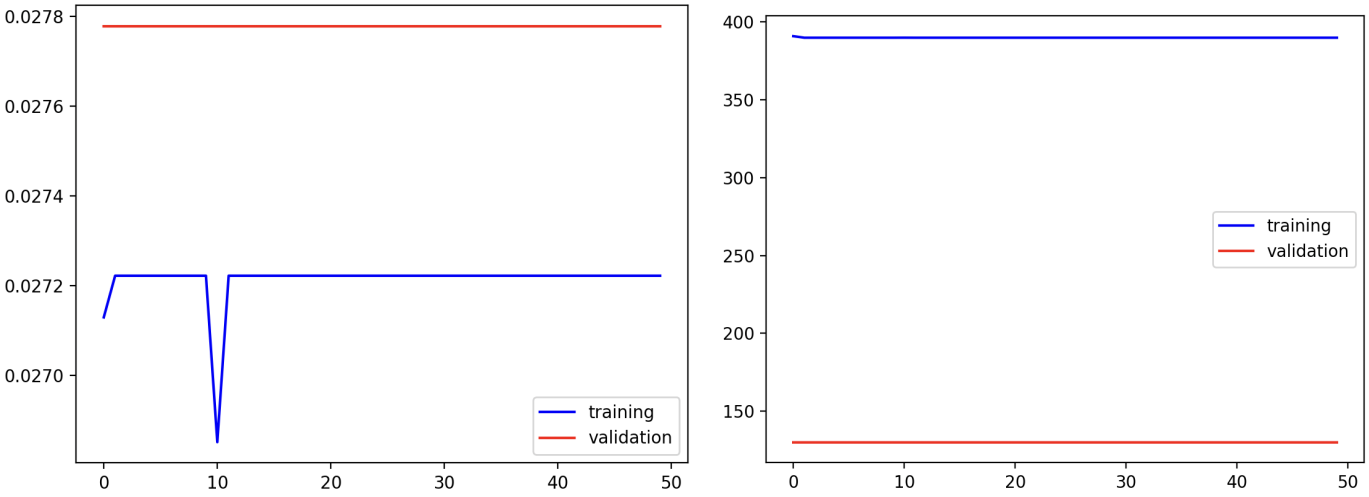
Q 3.2



Figures: (left) Accuracy, (right) Loss on training and validation data, with learning rate = 0.01



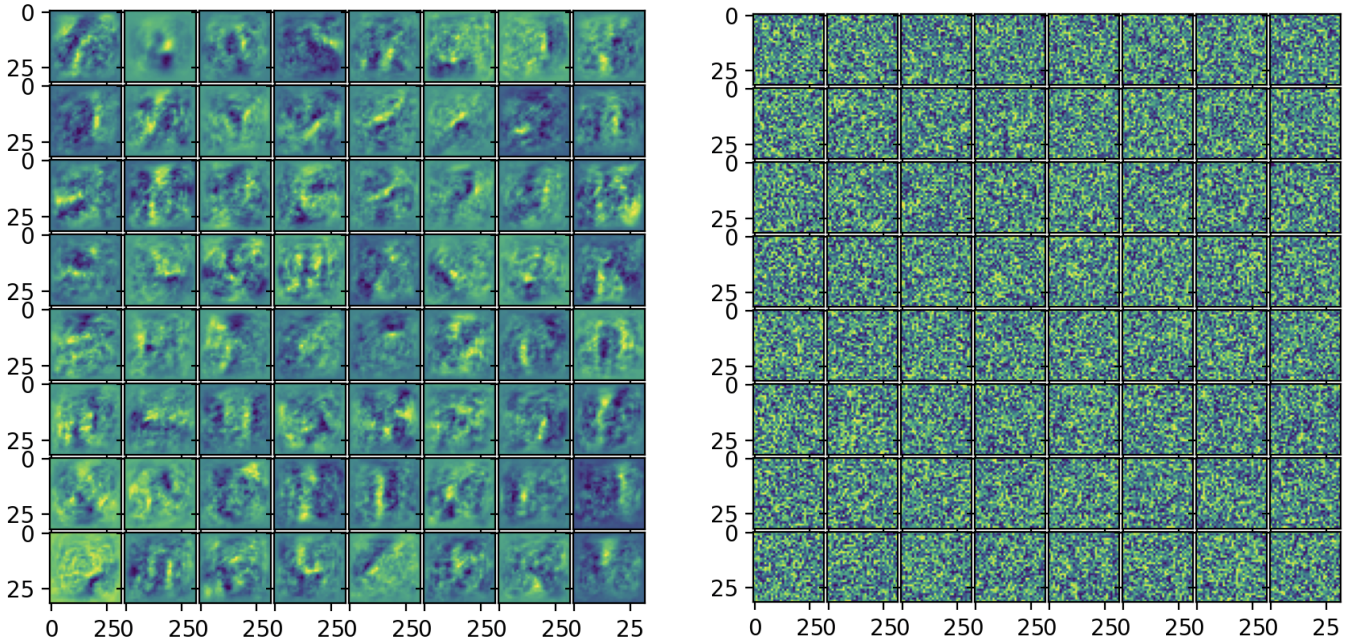
Figures: (left) Accuracy, (right) Loss on training and validation data, with learning rate = 0.001



Figures: (left) Accuracy, (right) Loss on training and validation data, with learning rate = 0.1

As we can see from the plots, with the best learning rate, the model can gradually converge to the appropriate parameters while learning the correct features. With the learning rate one tenth of that, the rate becomes too small and is slow to converge to the ideal parameters. The learning rate becomes too large at 10*lr and it jumps to reduce the error. In this case the jump to minimise the loss was so large that all the neurons learned the same features, and hence the network has the same output for all inputs

Q 3.3



Figures: Visualization of the first layer weights as images (right) immediately after initialization, and (left) after training

As we can see, the weights after initialization are completely random, as we would expect after random initialization, and after training, we can see patterns starting to form, as the learned weights represent different features.

Q 3.4

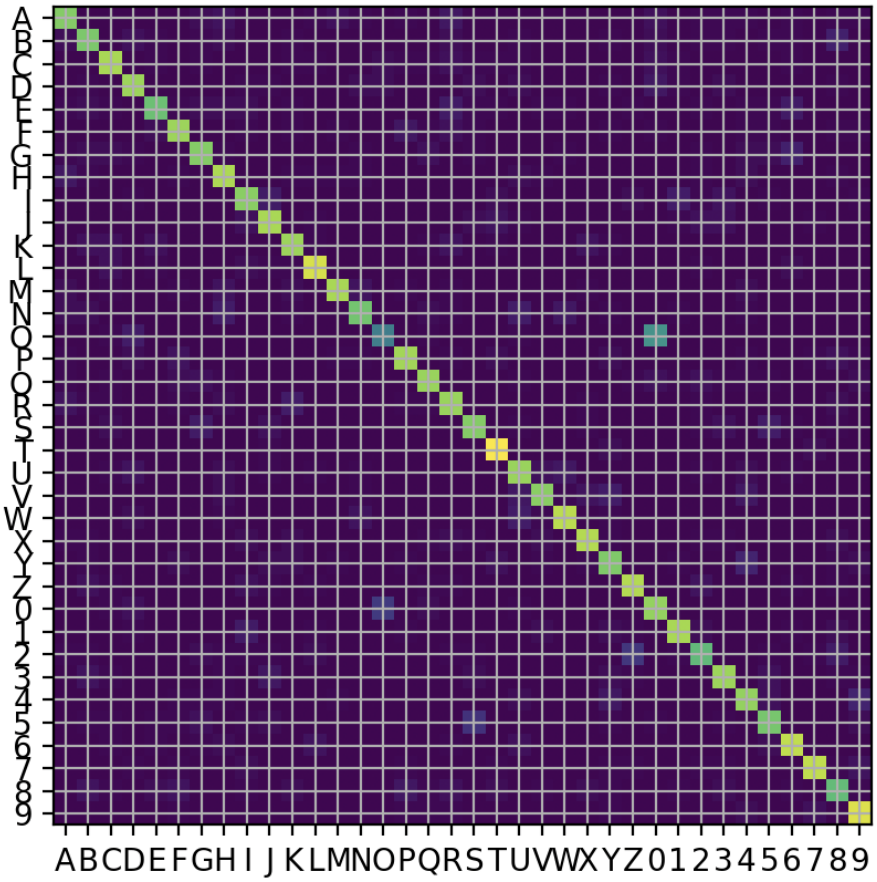
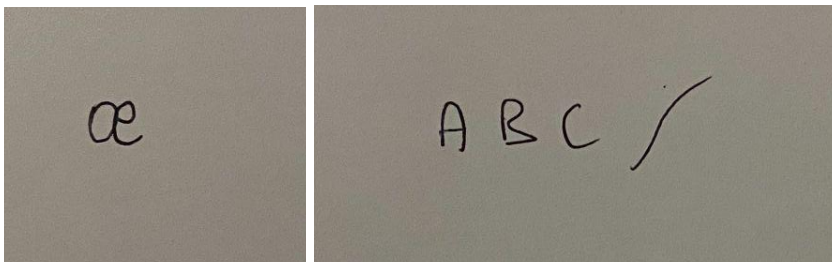


Figure: confusion matrix on the validation data using the best model.

As we can see, the model gets confused by characters that look the same. The most easily confused ones being 0 and O, 5 and S, 2 and Z.

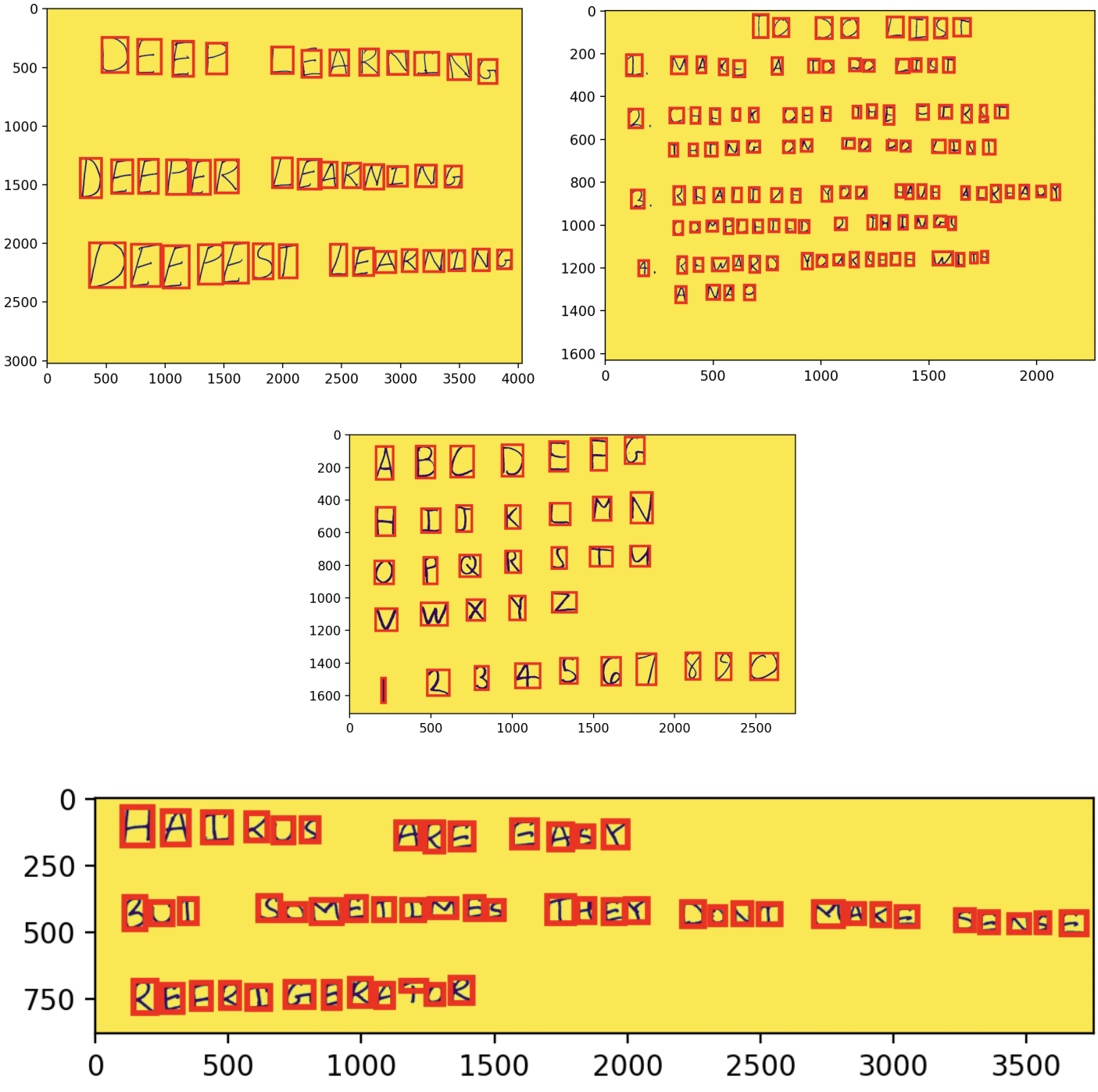
Q 4.1

Our method makes the assumption that the image of only one character is being passed to the model at a time, which will not be the case if two or more characters are written close to each other. The detection method fails to make this distinction.
We also assume that only characters of interest are written in the image, and it doesn't contain non-characters that could accidentally be responded to.



Figures: sample images where the character detection would fail: (Left) letters written close to each other, (right) non character markings in the image

Q 4.3



Figures: Result images of using findletters on the given images

Q 4.4

Ground Truth:	**Model Output**:
DEEP LEARNING DEEPER LEARNING DEEPEST LEARNING	ZEEP LEARBING DEBPER LEARNING GNIRNAL EISFREDE
Ground Truth: TO DO LIST 1 MAKE A TO DO LIST 2 CHECK OFF THE FIRST THING ON TO DO LIST 3 REALIZE YOU HAVE ALREADY COMPLETED 2 THINGS 4 REWARD YOURSELF WITH A NAP	**Model Output:** TO DO LIGT I MA8E A 7OD0 LIFT 2 CHR8K OFF 3HF F2R9KT 7HING ON T0 PO LIST 3 RKALIZE XOU HUVE ALREADT COMPLETID 2 1HINGG A REWARD YOU8FELF WITA A NPA
Ground Truth: ABCDEFG HIJKLMN OPQRSTU VWXYZ 1234567890	**Model Output:** ABCDBF HIYKLMN QPQRSTV VWXYZ 8Q9S76G3Z1
Ground Truth: HAIKUS ARE EASY BUT SOMETIMES THEY DONT MAKE SENSE REFRIGERATOR	**Model Output:** HAIKUG ARE EASX BUT SQMETIMEG FAEX D0NT MAKE SENGE MR0RGARBFIBR

Q 5.2

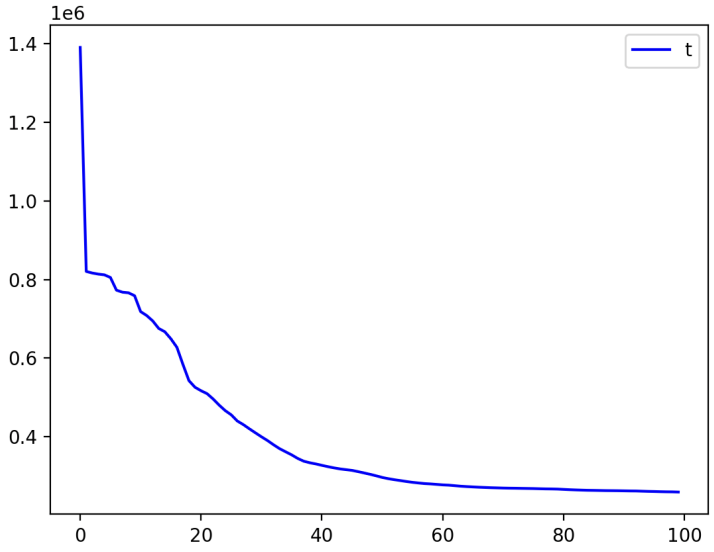
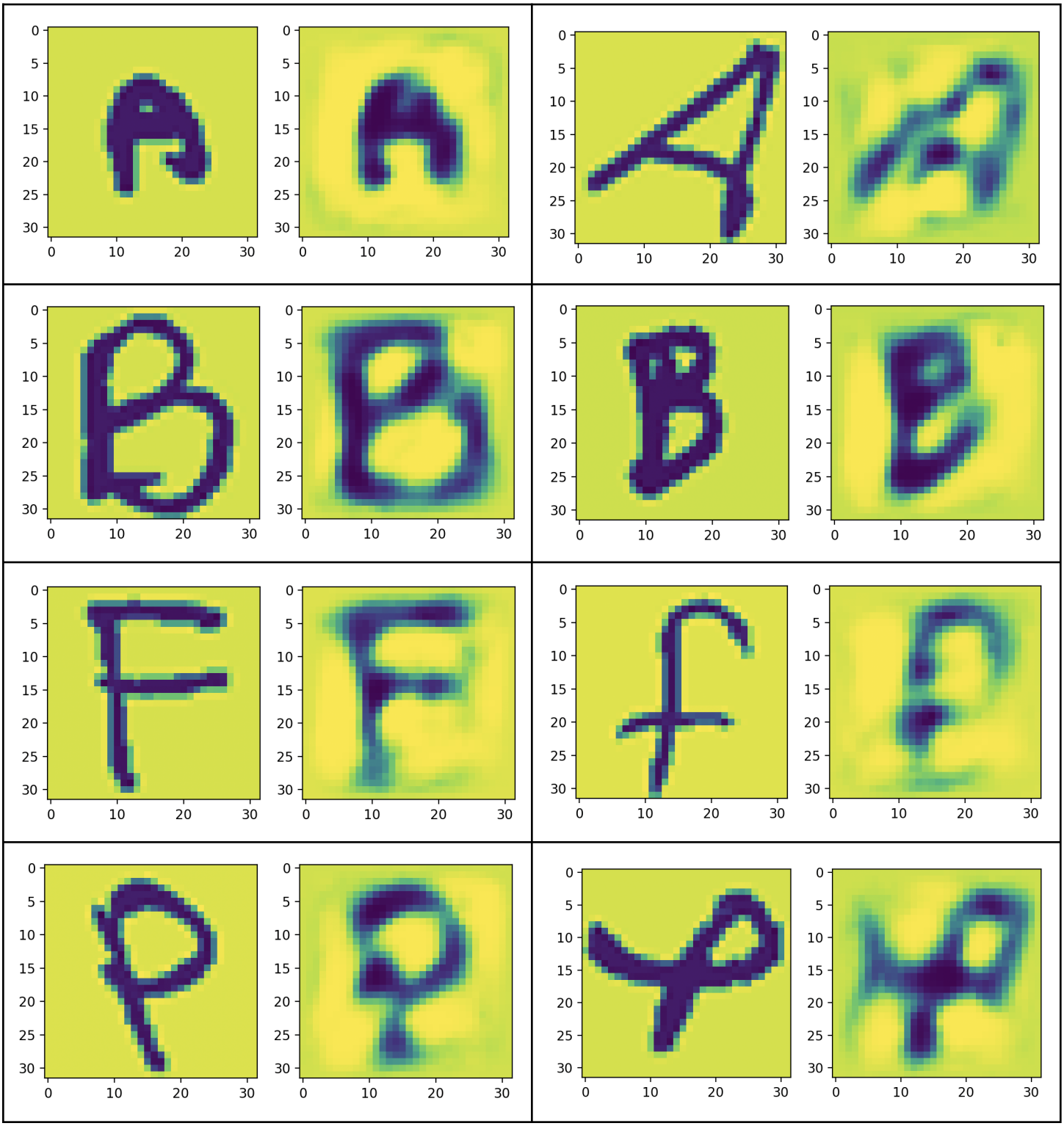
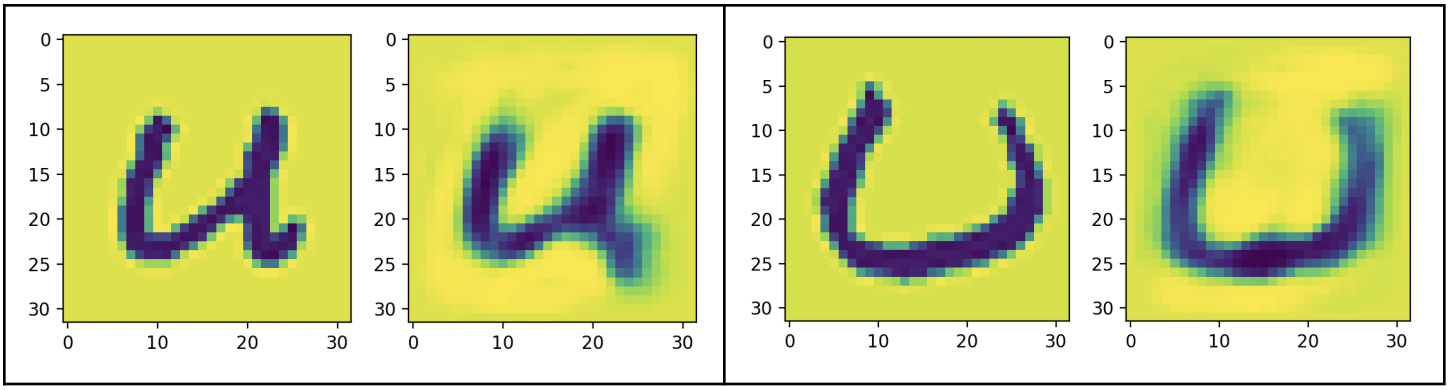


Figure: Plot of the training loss of the autoencoder.

The loss of the autoencoder suddenly drops over the first couple of iterations, and then reduces exponentially as it tends to converge at a certain loss.

Q 5.3.1

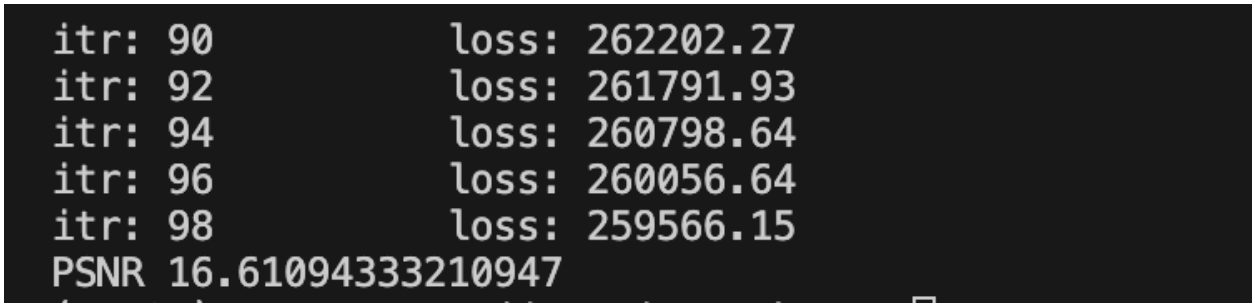




Figures: Reconstruction output samples from the autoencoder.

We can see that the reconstructed images are blurred as compared to the original image, and some reconstructed images have residual noise around the characters.

Q 5.3.2



The average PSNR from the autoencoder across all images in the validation set is 16.61