Q1.1)

$$The\ softmax\ function\ is\ given\ by:$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

$$softmax(x_i + c) = \frac{e^{x_i + c}}{\sum_{j=1}^{N} e^{x_j + c}} = \frac{e^{x_i}\ \mathbf{x}\ e^{c}}{\sum_{j=1}^{N} e^{x_j} \mathbf{x}\ e^{c}} = \frac{e^{c}\ \mathbf{x}\ e^{x_i}}{e^{c}\ \mathbf{x} \sum_{j=1}^{N} e^{x_j}}$$

$$softmax(x_i + c) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

$$\Longrightarrow softmax(x_i + c) = softmax(x_i)\ \forall\, c \in \Re \tag{1}$$

Since we computing the exponential of x_i, if x_i's are large positive values then the value of e^{x_i} will explode.

Since softmax is invariant to translation, we choose $c = -max(x_i)$

Thus, $x_i - max(x_i)$ is always $< 0 \Longrightarrow e^{x_i - max(x_i)}$ is always < 1

Hence the computation of softmax does not lead to numerical instability

Q1.2)

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

$$e^{x_i} > 0 \quad \forall\, x_i \in \Re$$

$$Hence,\ Softmax(x_i) > 0\ \forall\, x_i \in \Re \tag{1}$$

$$Since\ all\ the\ elements\ of\ numerator\ and\ denominator\ are\ positive$$

$$e^{x_i} < \sum_{j=1}^{N} e^{x_j}$$

$$\Longrightarrow Softmax(x_i) < 1 \tag{2}$$

$$Therefore,\ the\ range\ of\ each\ element\ of\ the\ softmax\ function\ is:$$

$$0 < Softmax(x_i) < 1 \tag{3}$$

$$\sum_{i=1}^{N} Softmax(x_i) = \sum_{i=1}^{N} \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = \frac{\sum_{i=1}^{N} e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = 1$$

$$\Longrightarrow \sum_{i=1}^{N} Softmax(x_i) = 1 \tag{4}$$

$$Therefore\ the\ properties\ of\ the\ softmax\ function\ are:$$

$$1.\ Range\ of\ softmax:\ 0 < Softmax(x_i) < 1$$

$$2.\ Summation\ of\ softmax:\ \sum_{i=1}^{N} Softmax(x_i) = 1$$

Q1.3)

Consider a multi – layer perceptron consisting of 1 *input layer, N hidden layers,* 1 *output layer*
Consider a standard notation of weights and input data.

Let the activation function be linear function.

The output from 1*st hidden layer* :

$$y_1 = XW_1 + b_1 \tag{1}$$

The output from 2*nd hidden layer* :

$$y_2 = y_1W_2 + b_2$$

$$y_2 = (XW_1 + b_1)W_2 + b_2$$
$$y_2 = X(W_1W_2) + (b_1W_2 + b_2) \tag{2}$$

Consider the out from nth hidden layer :

$$y_n = y_{n-1}W_n + b_n$$

$$y_n = (y_{n-2}W_{n-1} + b_{n-1})W_n + b_n$$

substitute all the y_i*'s; we obtain* :

$$y_n = X(W_nW_{n-1}\ldots W_1) + (b_n + b_{n-1}W_n + .. + b_1W_2)$$

$$\implies y_n = XW_{eff} + b_{eff} \tag{3}$$

$$\text{where, } W_{eff} = W_nW_{n-1}\ldots W_1 \;;\; b_{eff} = b_n + b_{n-1}W_n + .. + b_1W_2$$

eq (3) *is a linear regression in X. Therefore, a multi – layer perception*
with linear activation function simply results in a linear regression.

Q1.4)

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

$$\frac{\partial\, sigmoid(x)}{\partial\, x} = -\frac{1}{\left(1+e^{-x}\right)^2}\, e^{-x}\,(-1) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

$$\frac{\partial\, sigmoid(x)}{\partial\, x} = \frac{1}{1+e^{-x}} \mathbf{x} \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$\frac{\partial\, sigmoid(x)}{\partial\, x} = sigmoid(x)\, \mathbf{x}\, (1 - sigmoid(x))$$

$$\Longrightarrow \frac{\partial\, \sigma(x)}{\partial\, x} = \sigma(x)\,(1-\sigma(x))$$

Q1.5)

$$\textit{Given,}$$

$$y = Wx + b$$

$$\frac{\partial J}{\partial y} = \delta \in \Re^{k \times 1}$$

$$W \in \Re^{k \times d}; \; x \in \Re^{d \times 1}; \; b \in \Re^{k \times 1}$$

apply chain rule to get the gradients :

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial W} = \delta \frac{\partial y}{\partial W}$$

$$\frac{\partial y}{\partial W} = x^T$$

$$\Longrightarrow \frac{\partial J}{\partial W} = \delta x^T \tag{1}$$

Note that $x \in \Re^{d \times 1}$, *i.e. its a single training data point*

if $x \in \Re^{N \times d}$ *where N is the sample size*

then $\frac{\partial J}{\partial W} = x^T \delta$

similarly,

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial x} = \delta \frac{\partial y}{\partial x}$$

Let y be the outputs for N sample sizes, then :

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & .. & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & .. & .. \\ .. & & .. & \\ & & & \frac{\partial y_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} W_{11} & W_{21} & .. & W_{n1} \\ .. & & & .. \\ .. & & & \\ & & & W_{nn} \end{bmatrix} = W^T$$

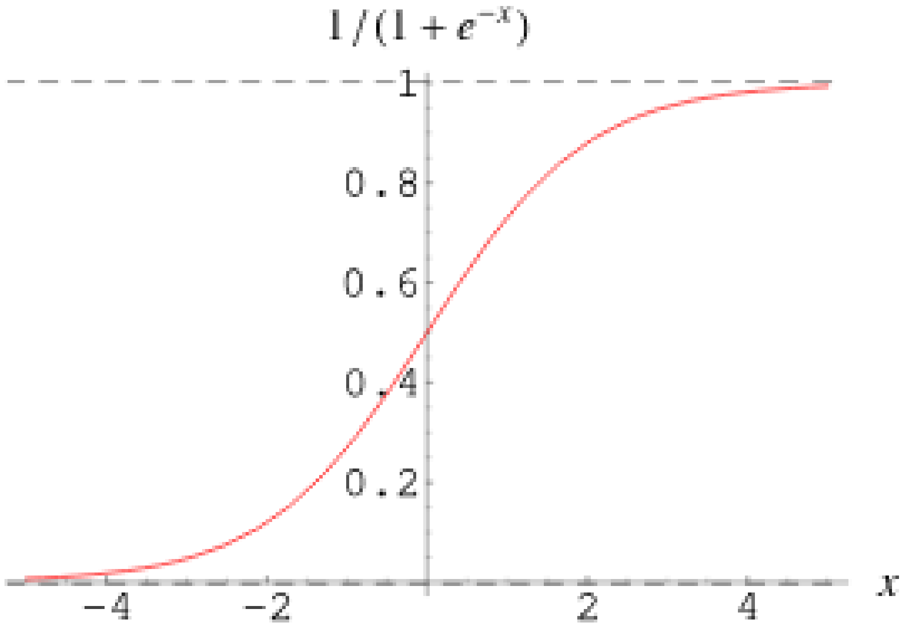
$$\frac{\partial J}{\partial x} = \delta W^T$$

```
    delta = delta*activation_deriv(post_act)

    grad_W = np.matmul(np.transpose(X), delta)
    grad_b = delta.sum(0)
    grad_X = np.matmul(delta, np.transpose(W))
```

Q1.6)

1)

The following is the plot for sigmoid function.



Observe that for values of x < -4, the value of sigmoid(x) is essentially equal to zero.

$$\frac{\partial f(g(x))}{\partial x} = f'(g(x))\, g'(x)$$

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

$$g'(x) = g(x)\,(1 - g(x))$$

$$\begin{gathered} For\ x < -4 : \\ g(x) \rightarrow 0 \\ \implies g'(x) = g(x)\,(1 - g(x)) \rightarrow 0 \end{gathered}$$

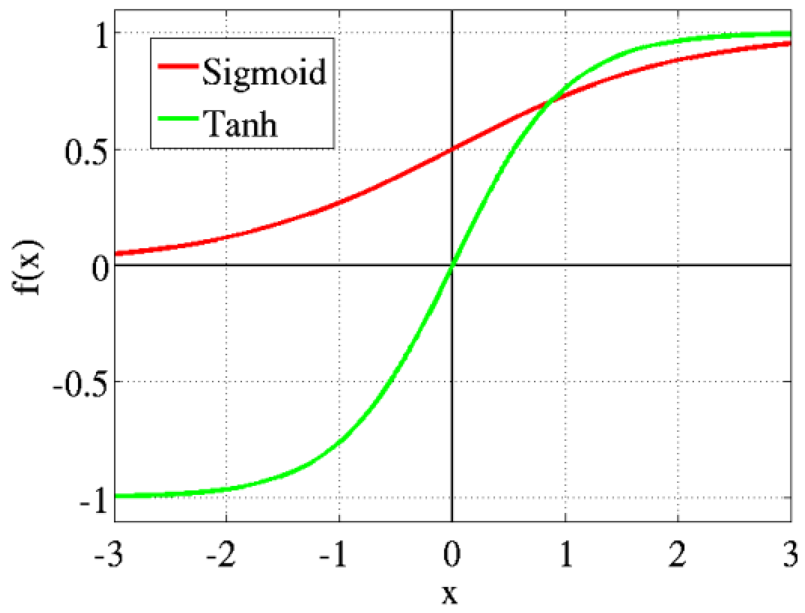
$$\text{For } x > 4:$$
$$g(x) \rightarrow 1$$
$$\implies g'(x) = g(x)\,(1 - g(x)) \rightarrow 0$$

$$\textit{Therefore, for very small and large values of } x\textit{, the derivative of sigmoid tends to zero}$$

$$\implies \frac{\partial f(g(x))}{\partial x} = f'(g(x))\,g'(x) \rightarrow 0$$

$$\textit{Hence, the gradients tend to become zero for sigmoid activation functions}$$
$$\textit{when the value of magnitude of } x \textit{ is large.}$$

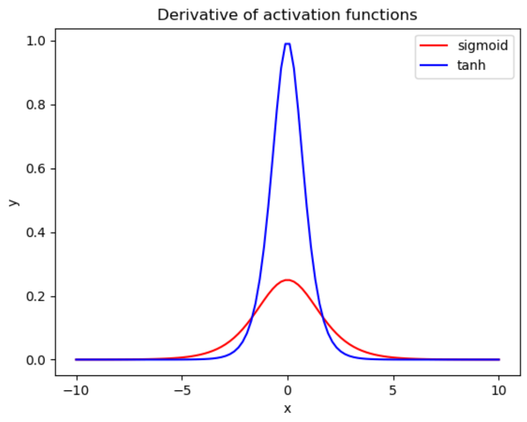
2)



source

We can observe that the range tanh is from [-1,1] whereas that of sigmoid is [0,1]. Therefore, having a broader range can help with certain trainsets where we want to capture the output on both positive and negative sides. Therefore, the internal values being centered around zero can lead to better training.

3)



source

Therefore, we can observe that the values of the derivatives for tanh are significantly higher than that of sigmoid. Therefore, during backprop and differentiation, there is a lesser possibility for the gradients to vanish to zero for the tanh case

4)

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

$$tanh(x) = 2sigmoid(-2x) - 1$$

Q2.1.1) Zero Initialised Weights

zero initialised weights is not a good idea for the following reasons:

1. If all the weights and biases are initialised to zero, the initial output of the network is zero. If we initialise all the weights to zeros, then the training can get stuck in a local minima.

2. The activation function has effect on the training. If the activation function is either ReLU or Tanh where the post activation value is zero for zero input, the gradients become zero always and the model never learns anything.

$$\begin{aligned} delta &= delta * activation\_deriv(post\_act) \\ grad\_W &= np.matmul(np.transpose(X),\ delta) \end{aligned}$$

For zero initialisation, X = 0 and therfore grad_W = 0 always

3. Since all the weights are initialised to be the same, there is a great possibilty for them to descent in the same direction. Therefore, the model performs poorly due hindered learning.

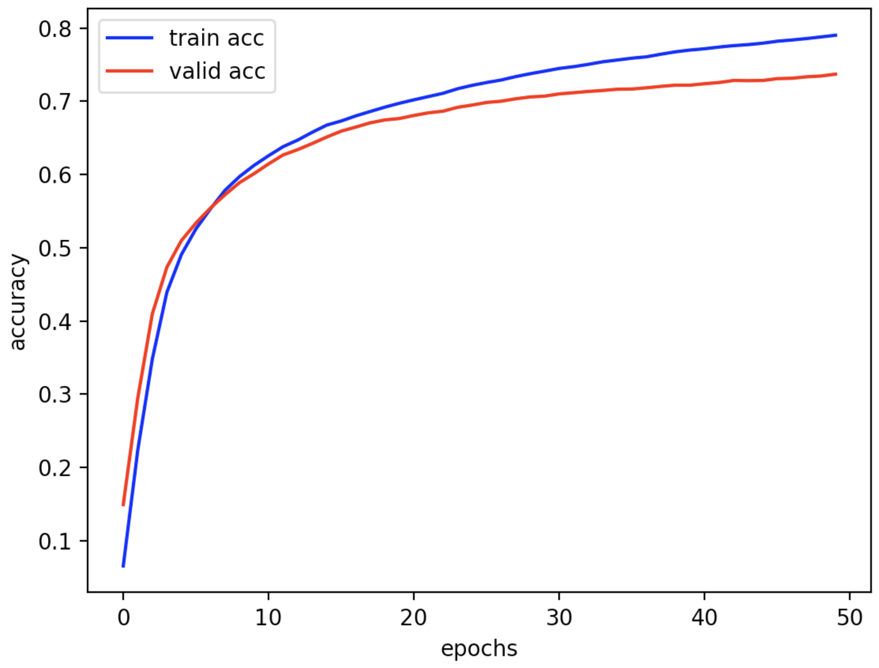
Q2.1.3)
We initialise the weights with random numbers in order break any underlying prior distribution or symmetry. Having a prior distribution of weights can bias the training leading to poor learning.

Initialising with random numbers makes the weights to learn in different trajectories leading to diversity in the learned features.

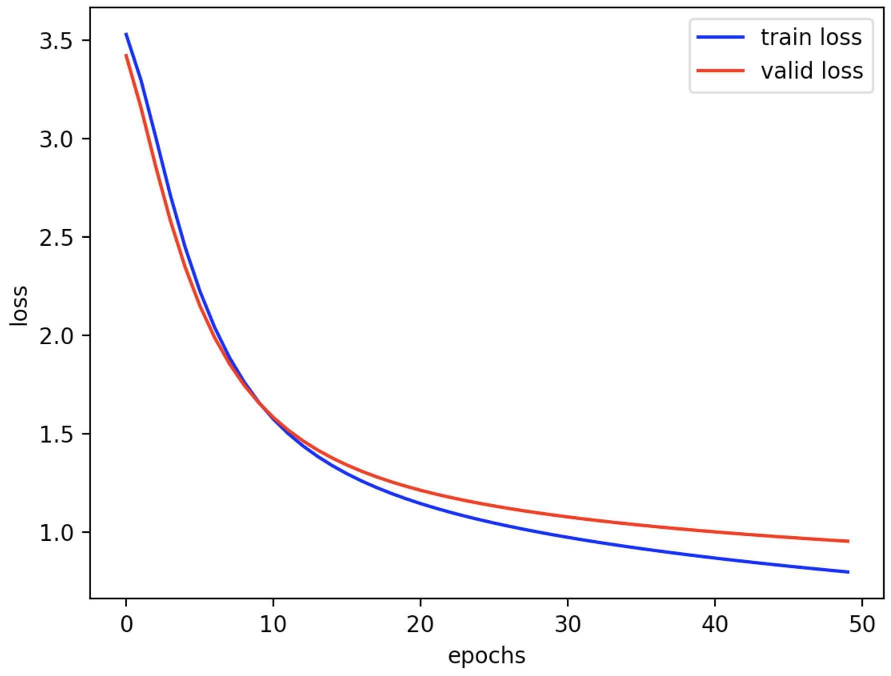
The number of weights in each layer varies. As we go deeper into the layers, if the variance is not accounted for, then during feedforward the value propagated either keeps incresing or decreasing. This leads to either exploding or vanishing gradients which are undesirable.

Having variance = 2/(n_in + n_out) ensures a balanced distribution of pre activation outputs.

Q3.1)

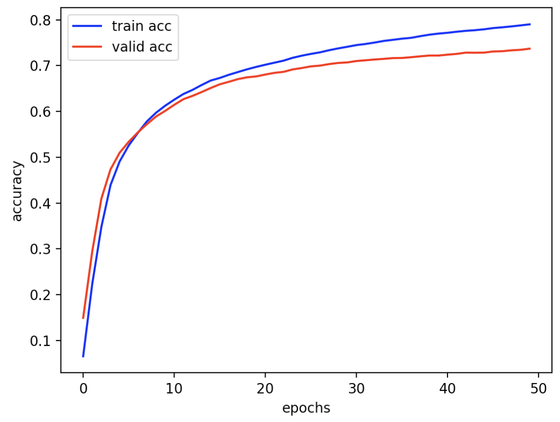


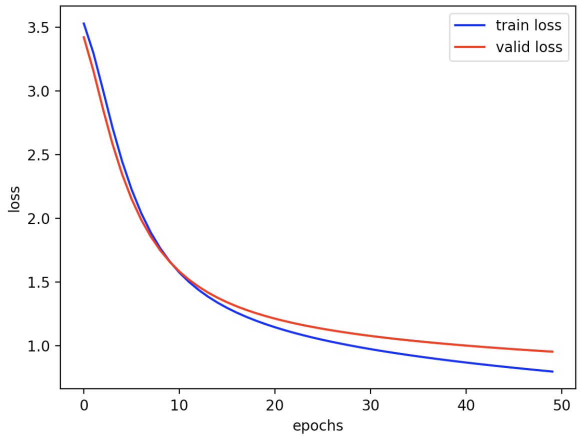
The achieved accuracy on the validation set = 73.9



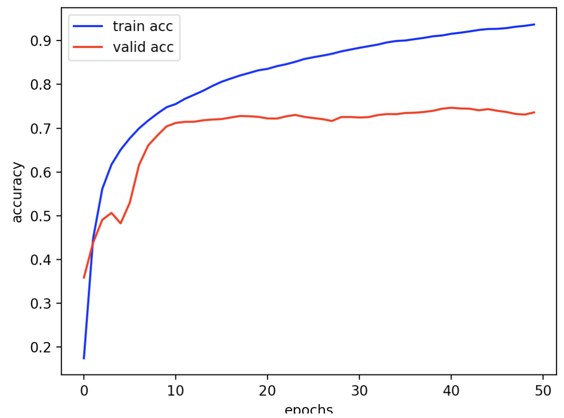
Q3.2)

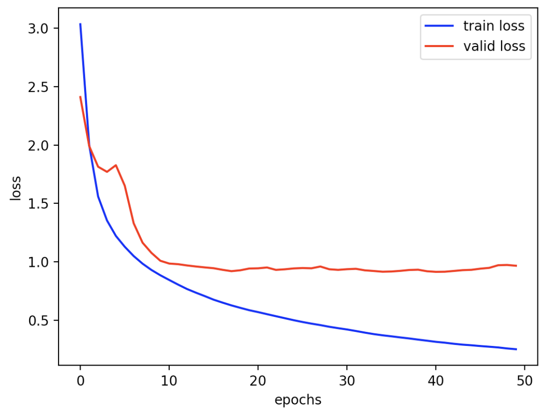
LR = 0.001



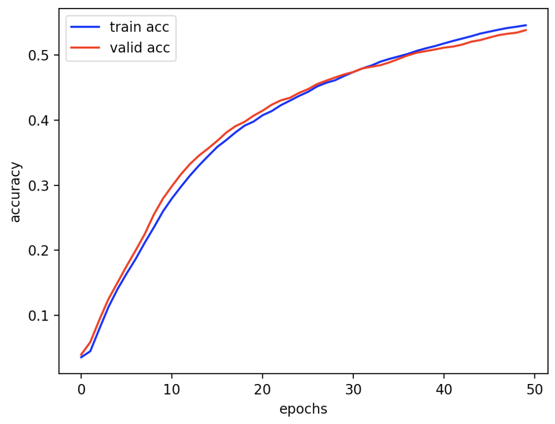


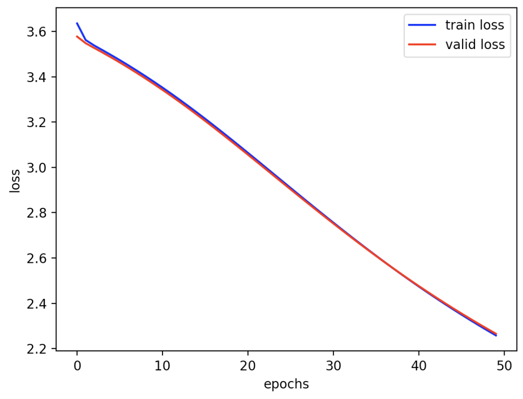
Learning rate = 0.01 = 10*LR





Learning rate = 0.0001 = 0.1 LR





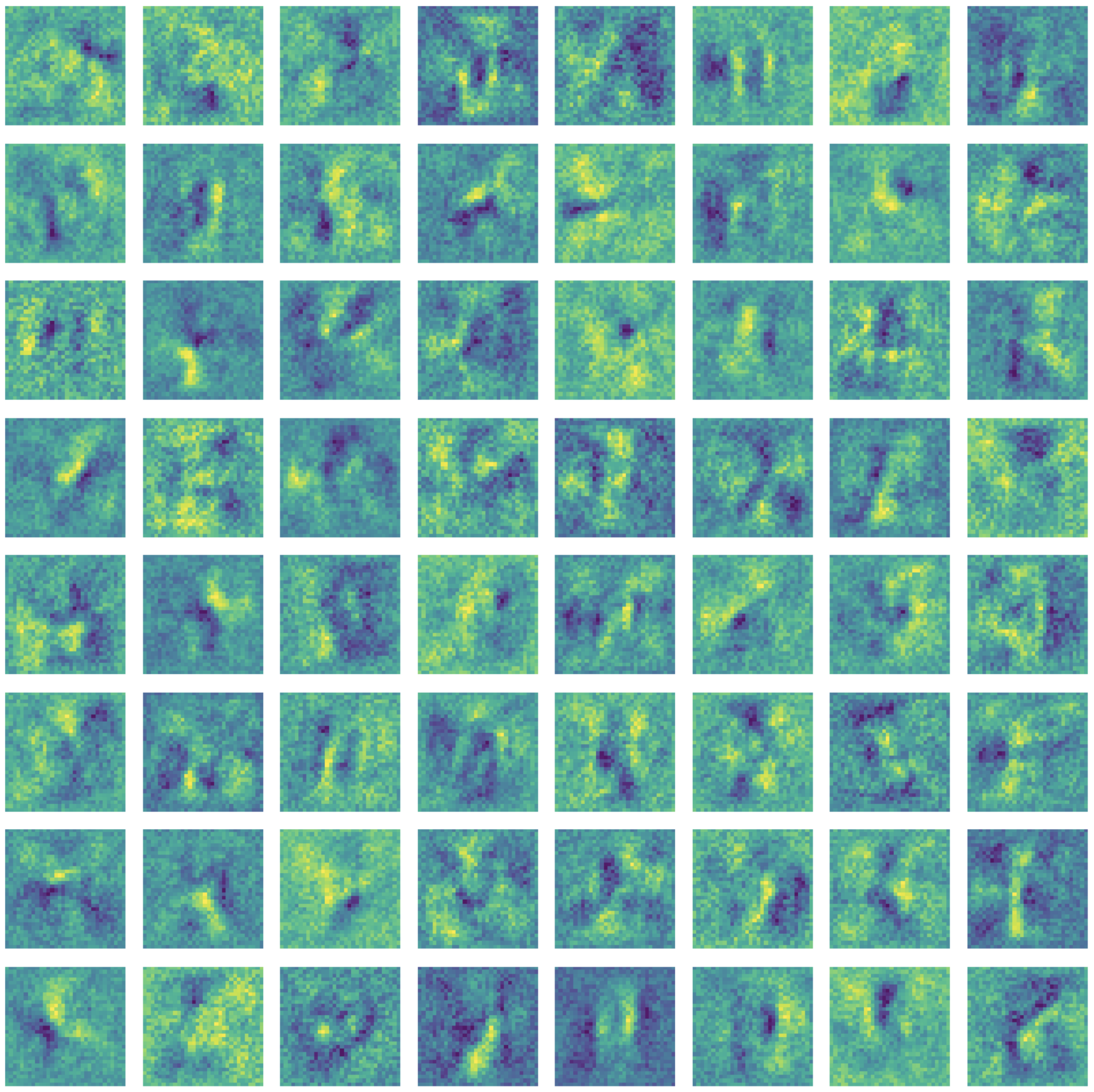
Observations:

1. When the learning rate is high, i.e. 0.01, the training loss converges faster and infact the loss reduces to great extent. However the validation loss doesn’t reduce much in comparison to the initial training loss. There is a lot of overfitting in this case.

2. When the learning rate is small, there is less overfitting. The training converges slowly, and the accuracy obtained is less compared to the initial training accuracy

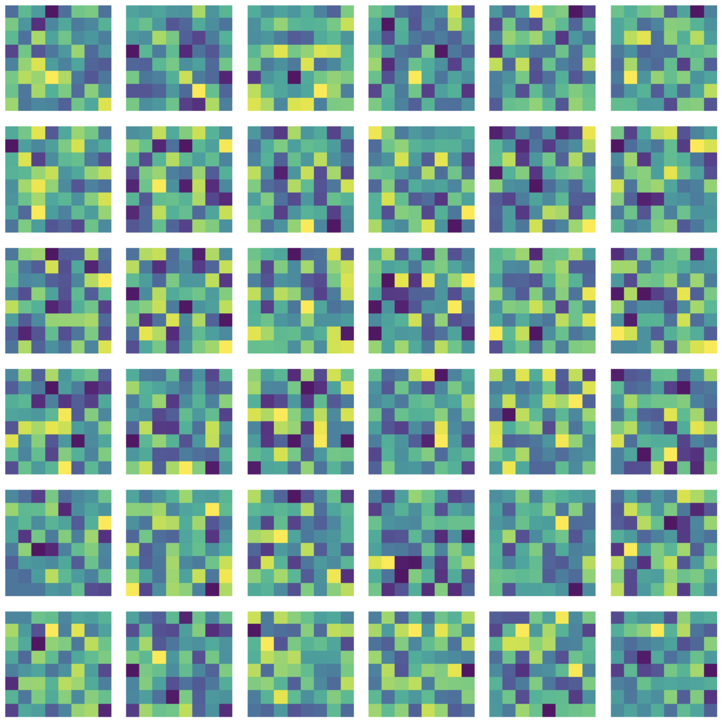
Q3.3)

Layer 1 weights

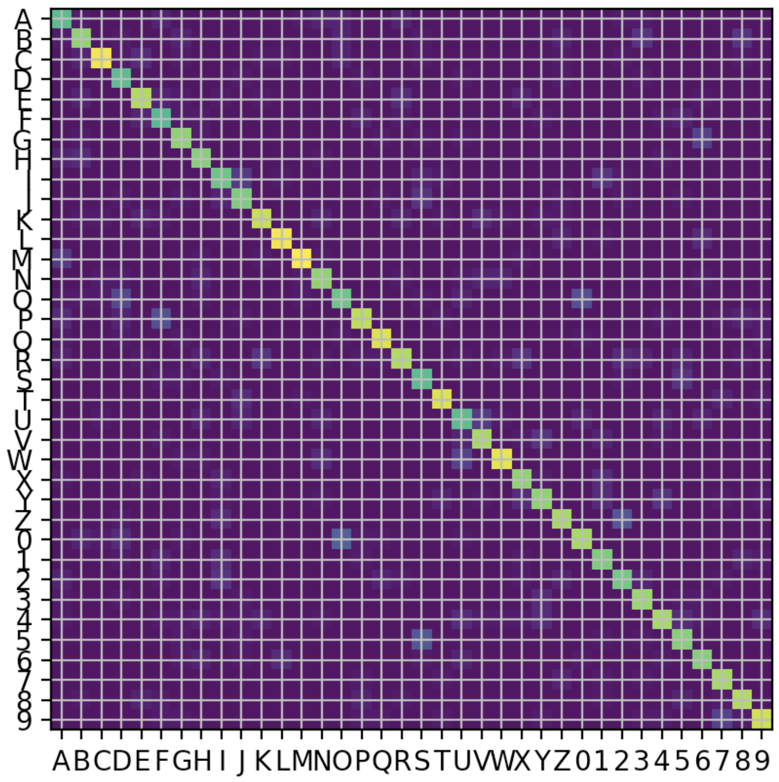


We can see some fundamental patterns in the weights images. There are some vertical lines, some horizontal, some closed shapes similar to the letter “O”. All these fundamental structures are learned from the characters in the training data. These weights matrices consists of the fundamental parts that construct a character.

Output layer weights:



Q3.4) Confusion Matrix



From the confusion matrix, the characters corresponding to high blue and off-diagonal are most commonly confused.

The common mistaken pairs are:

1. O and 0
2. O and D
3. P and F
4. 5 and S

Observe that these characters are being confused because they share a common shape or a common part. For example, the shape of 5 and S, and O and 0 are very similar. O and D are closed shapes and a little similar and therefore being confused.

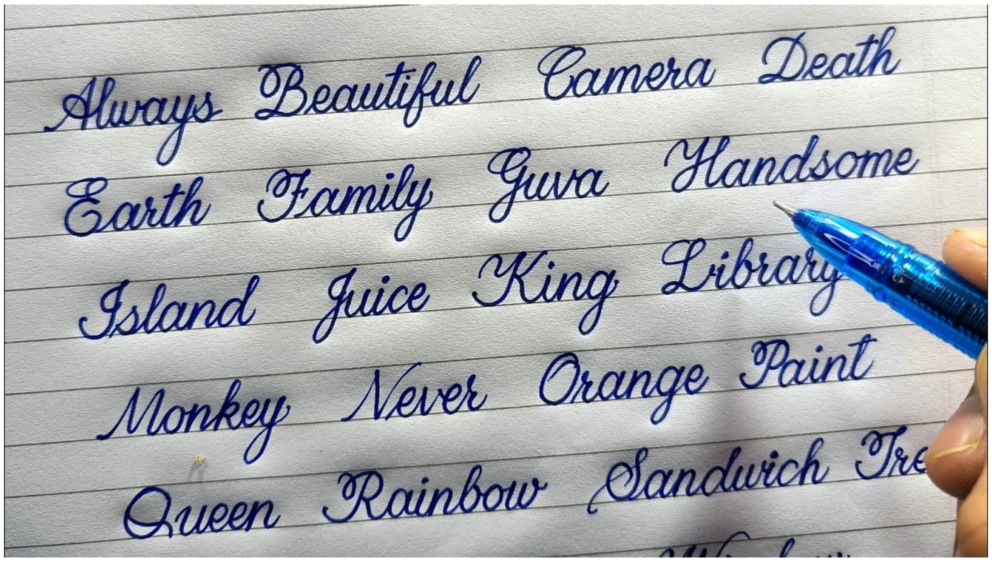
Q4.1)

The assumptions in the method are:

1. The method assumes clear formatting and spacing between the text
2. It works for structured text. For texts in unstructured environments, where the text is not necessarily written horizontally, this algorithm fails
3. The letters can be separated from the background by thresholding. This assumption may fail in complex images

Cases where the character detection fails:

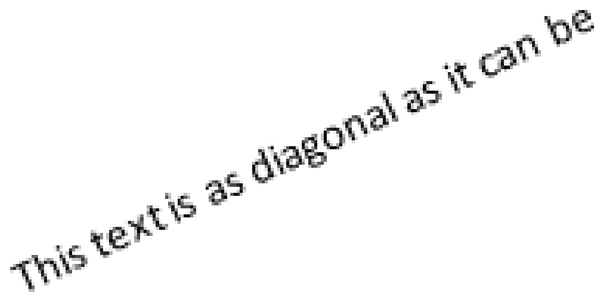
1. Cursive handwriting

   
   Always Beautiful Camera Death
Earth Family Guva Handsome
Island Juice King Library
Monkey Never Orange Paint
Queen Rainbow Sandwich

   Source image [Google]: https://i.ytimg.com/vi/_JDUwfKlsd0/maxresdefault.jpg

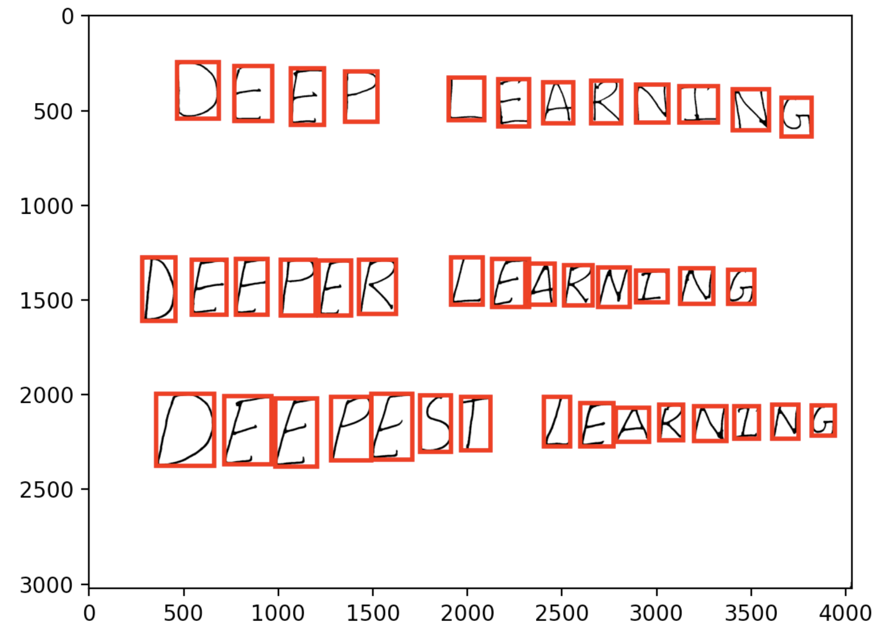
   Here due to the cursive handwriting, the characters are not separated.

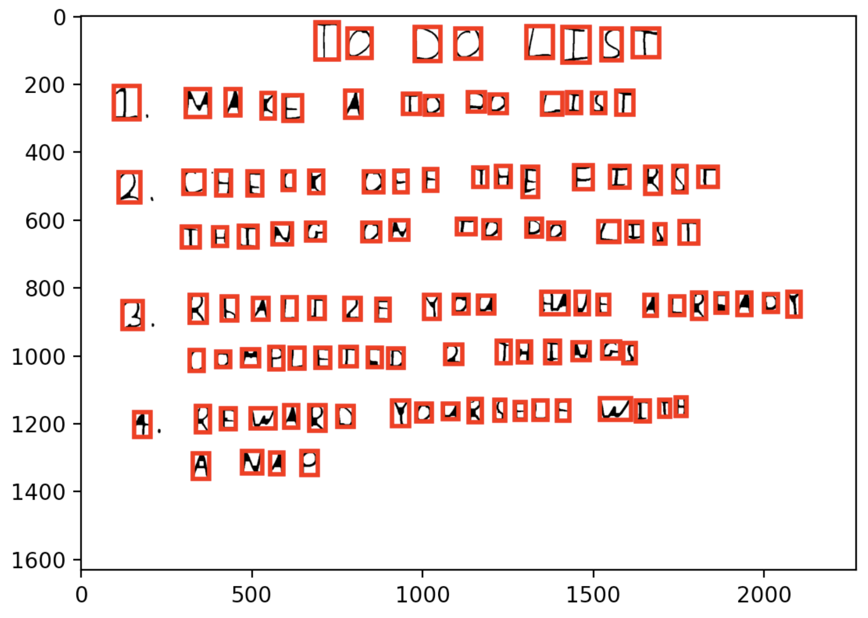
2. The text in the image is placed diagonally:

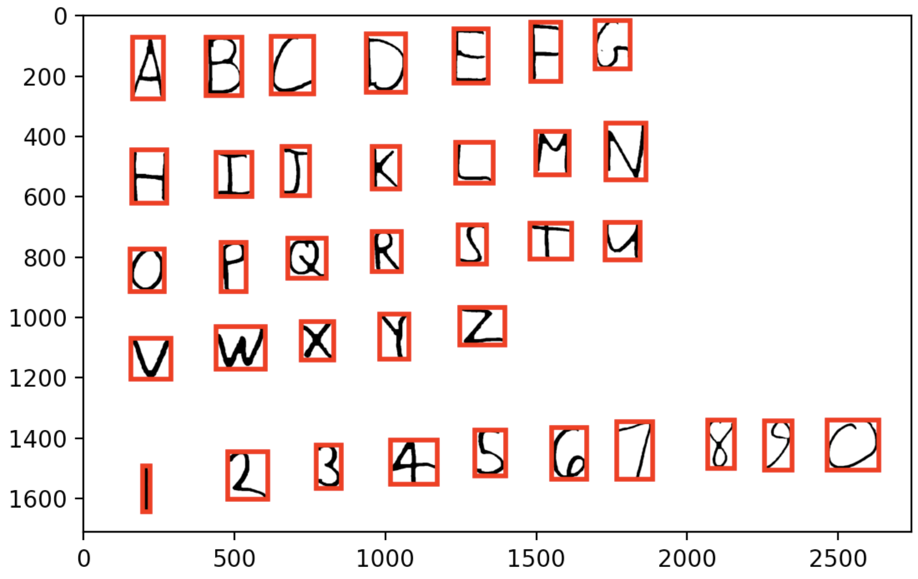
   
   This text is as diagonal as it can be

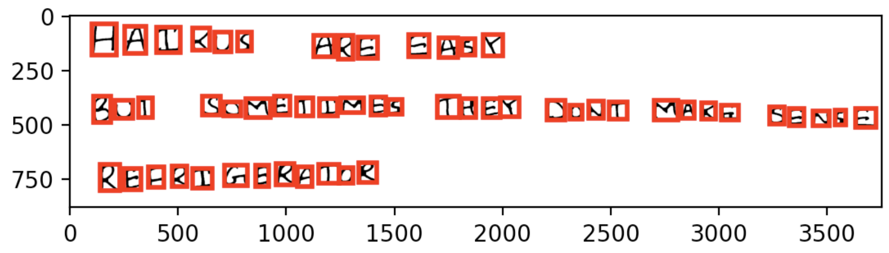
   In this case, the characters will be detected but they wont be in sequence since they are not placed horizontally

Q4.3)


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
3500
4000
DEEP LEARNING
DEEPER LEARNING
DEEPEST LEARNING

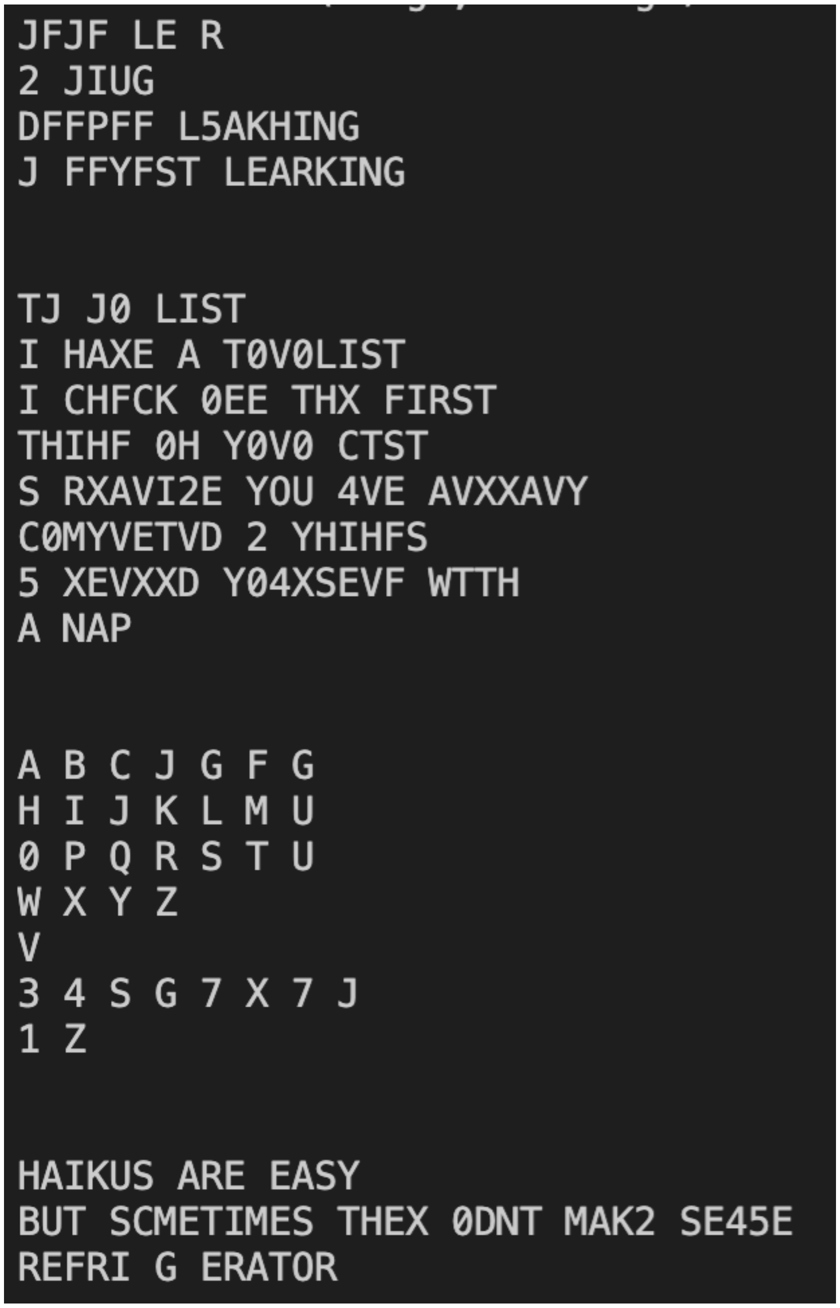

0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000
TO DO LIST


A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
1 2 3 4 5 6 7 8 9 0
0 200 400 600 800 1000 1200 1400 1600
0 500 1000 1500 2000 2500


HAIKUS ARE EASY
BUT SOMETIMES THEY DONT MAKE SENSE
REFRIGERATOR
0 250 500 750
0 500 1000 1500 2000 2500 3000 3500

Q4.4)

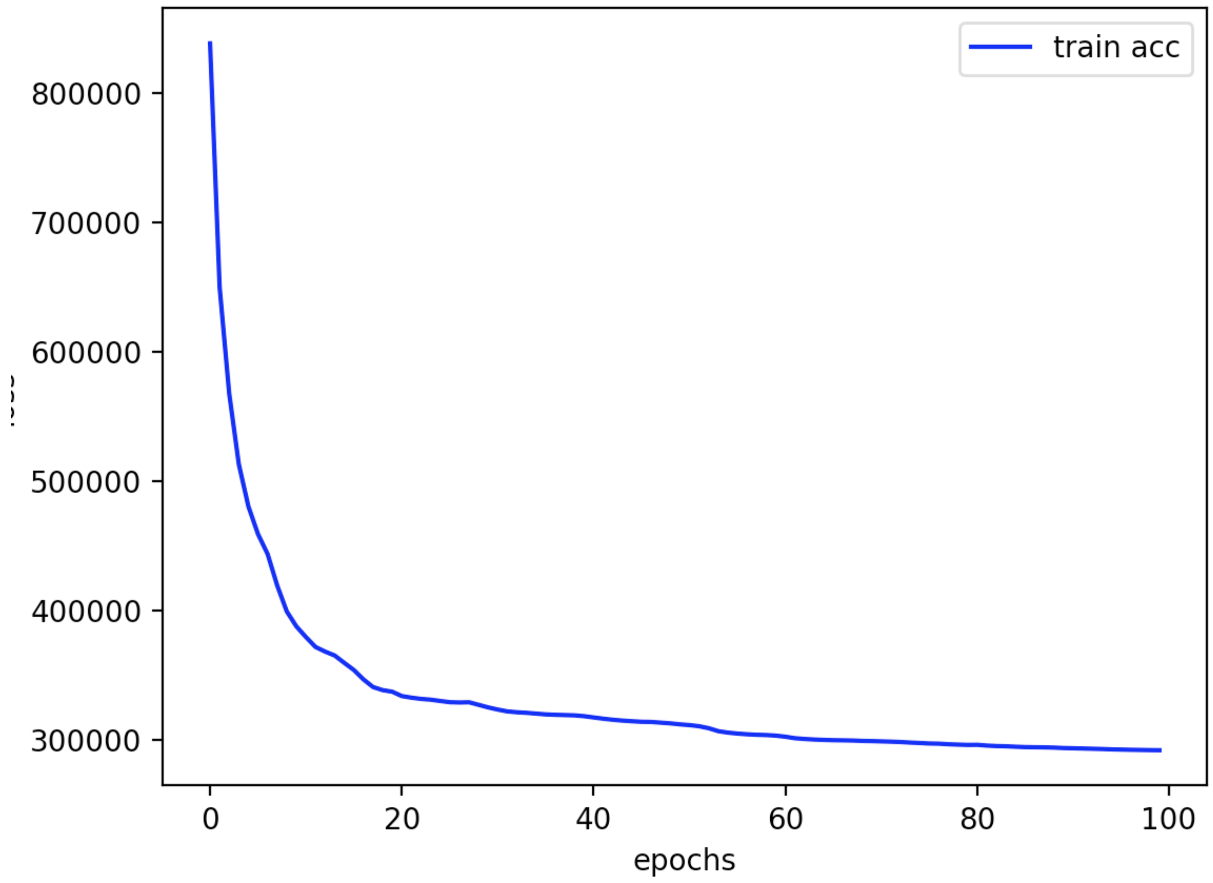
Obtained detections and classifications:



```
JFJF LE R
2 JIUG
DFFPFF L5AKHING
J FFYFST LEARKING


TJ J0 LIST
I HAXE A T0V0LIST
I CHFCK 0EE THX FIRST
THIHF 0H Y0V0 CTST
S RXAVI2E YOU 4VE AVXXAVY
C0MYVETVD 2 YHIHFS
5 XEVXXD Y04XSEVF WTTH
A NAP


A B C J G F G
H I J K L M U
0 P Q R S T U
W X Y Z
V
3 4 S G 7 X 7 J
1 Z


HAIKUS ARE EASY
BUT SCMETIMES THEX 0DNT MAK2 SE45E
REFRI G ERATOR
```

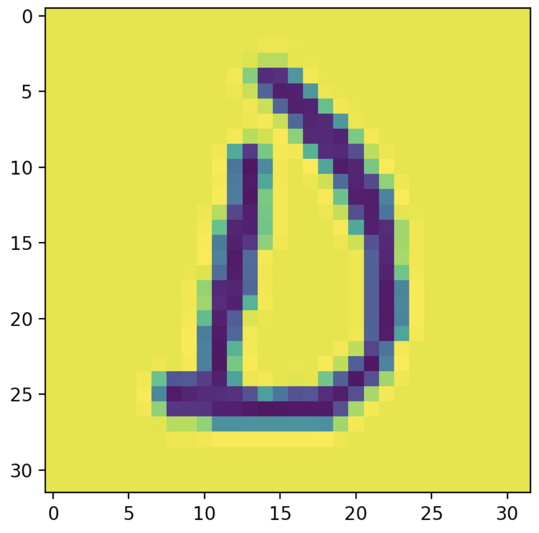
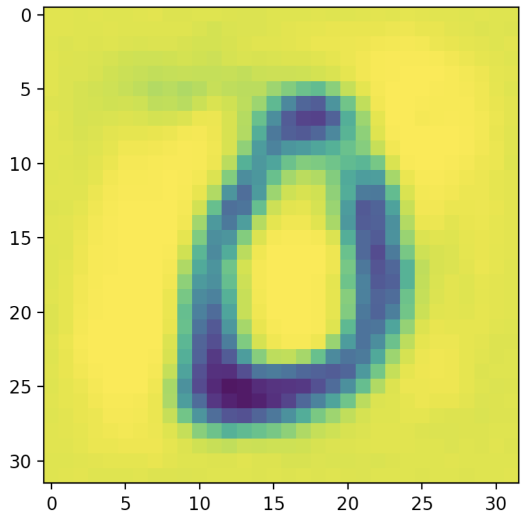
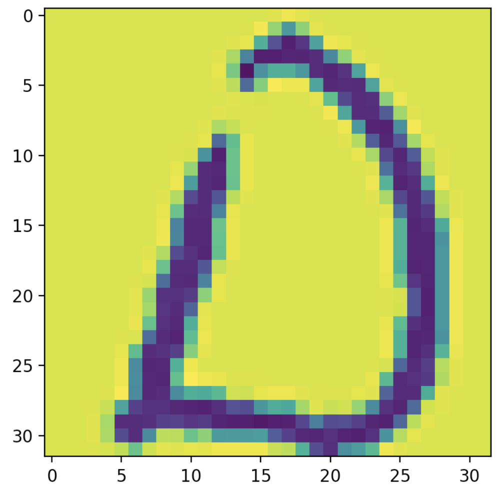
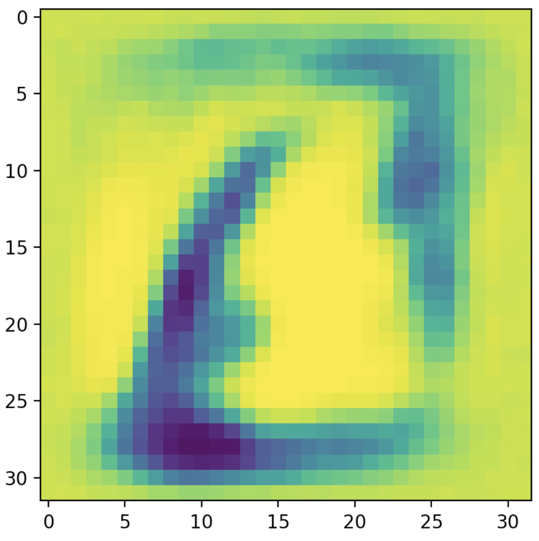
Q5.2)

max_iters = 100
batch_size = 36
learning_rate = 3e-5
hidden_size = 32
lr_rate = 20

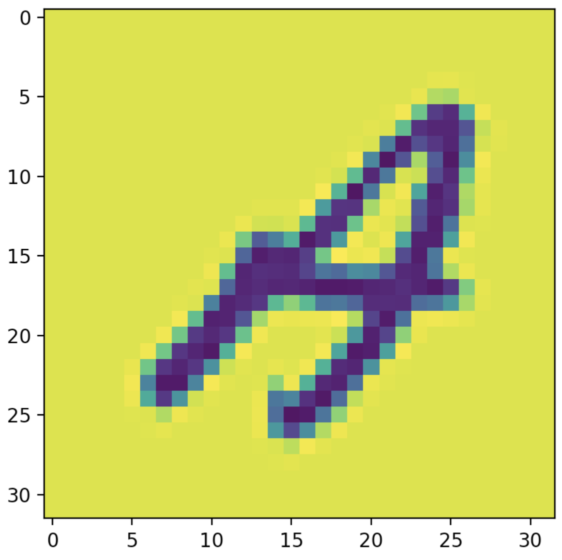


The training loss curve is significantly steep. Beyond around 40 epochs, the loss is practically the same implying that the model is not learning much any further.

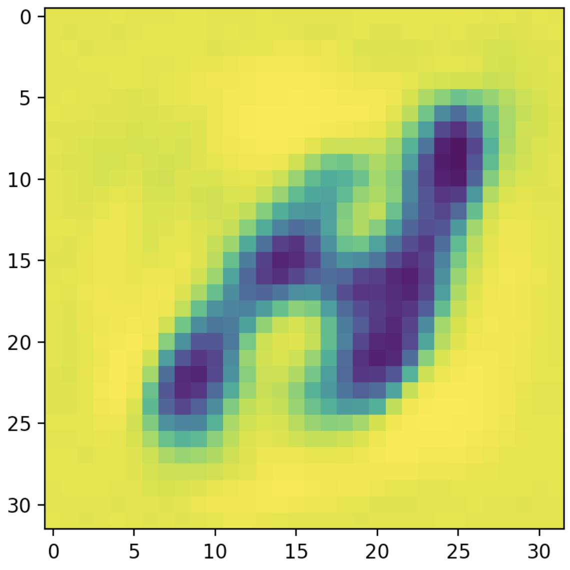
Q5.3)
Class D

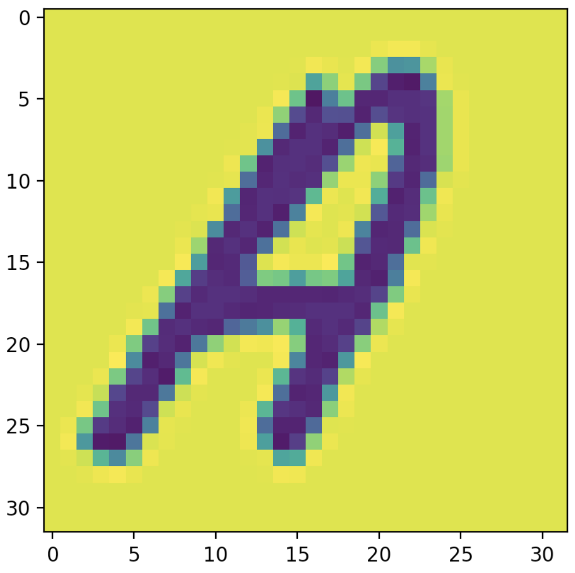
Class A



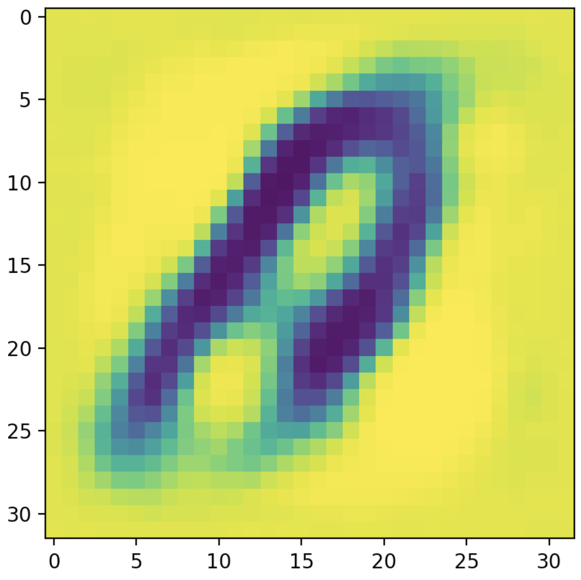
0
5
10
15
20
25
30
0
5
10
15
20
25
30



0
5
10
15
20
25
30
0
5
10
15
20
25
30

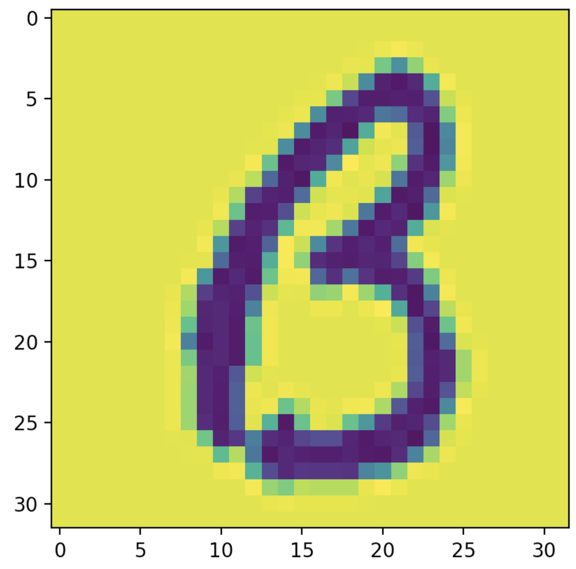


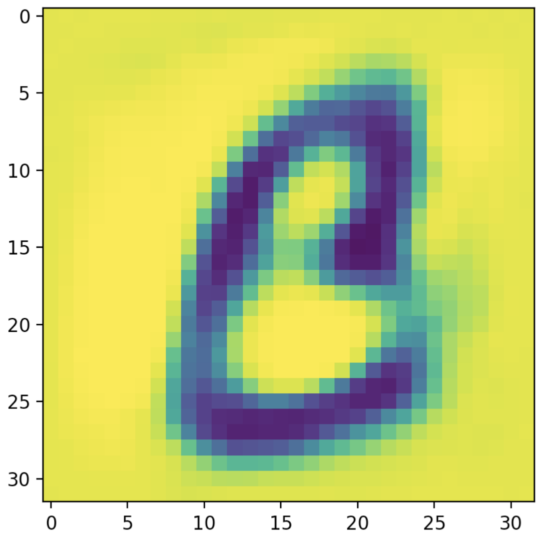
0
5
10
15
20
25
30
0
5
10
15
20
25
30

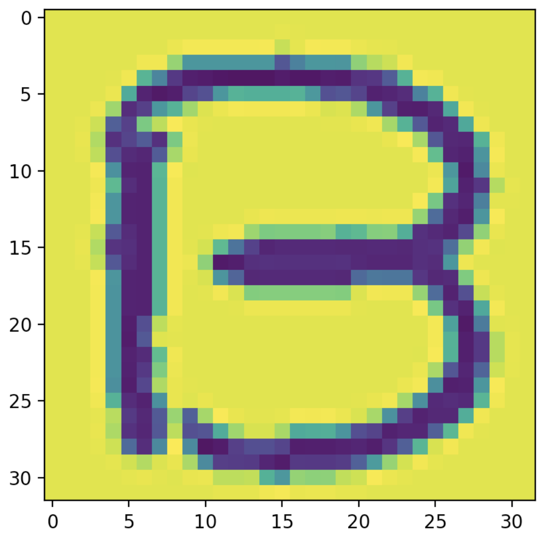


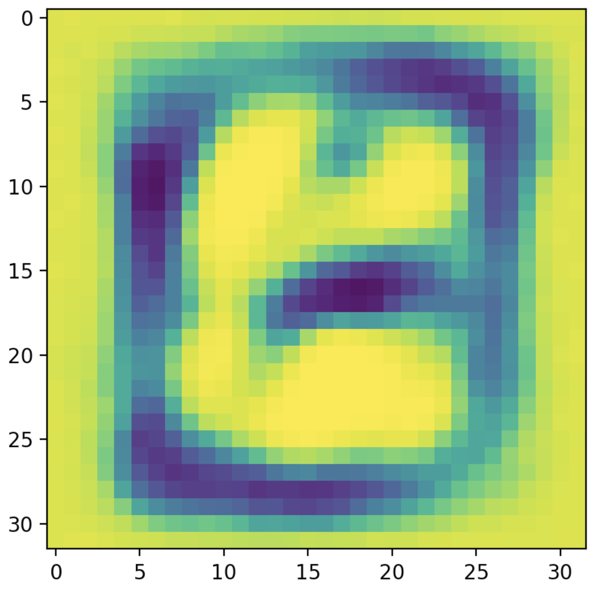
0
5
10
15
20
25
30
0
5
10
15
20
25
30

Class B

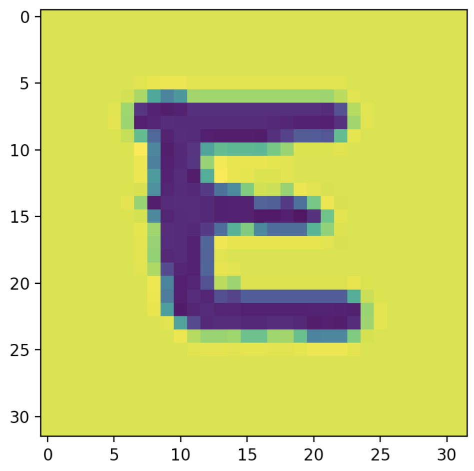
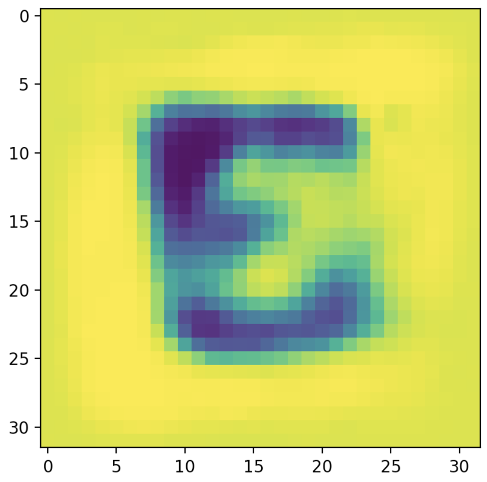
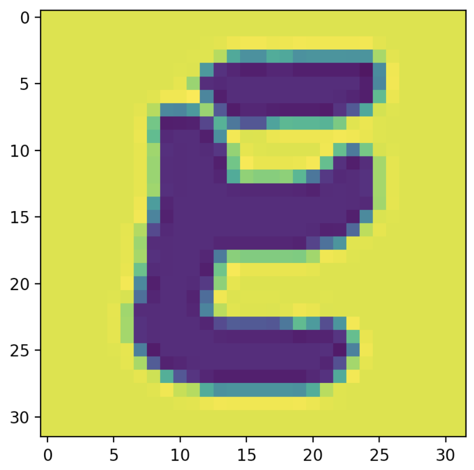
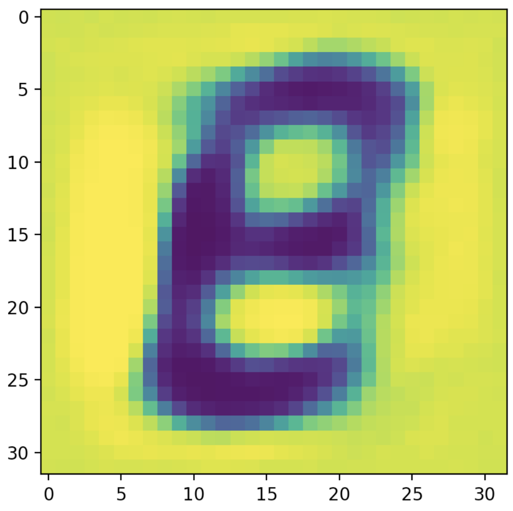

0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30

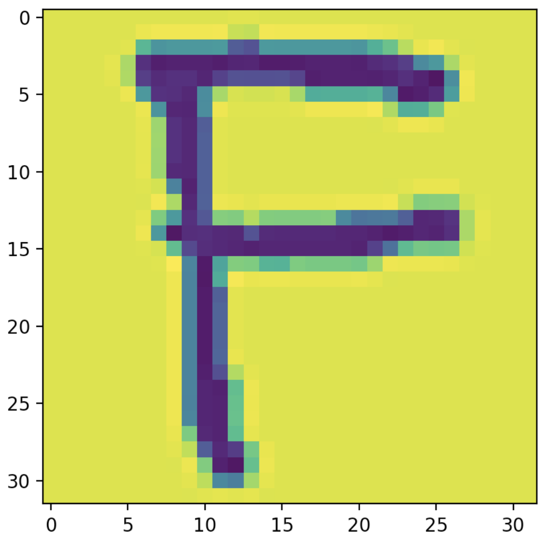
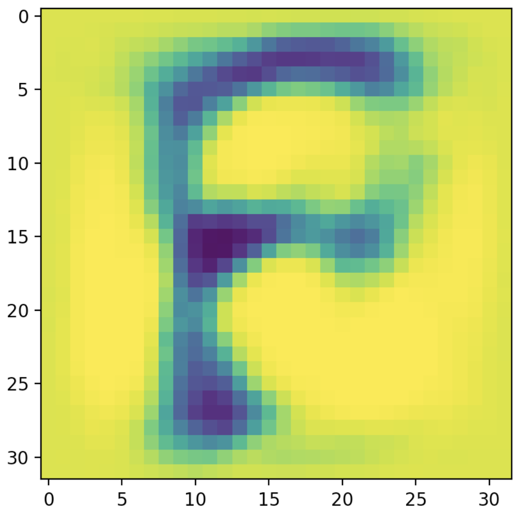
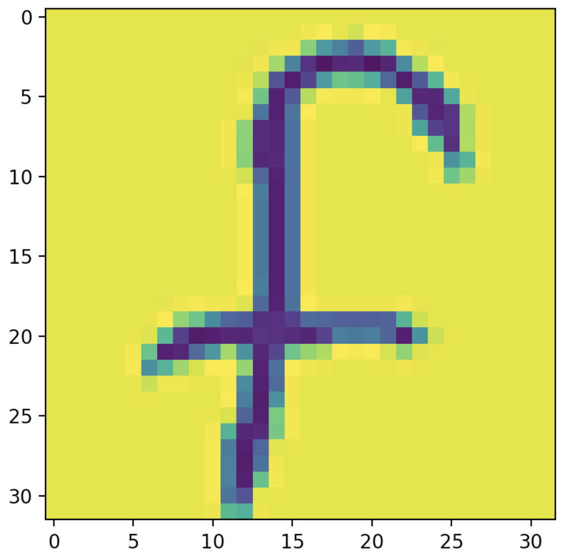
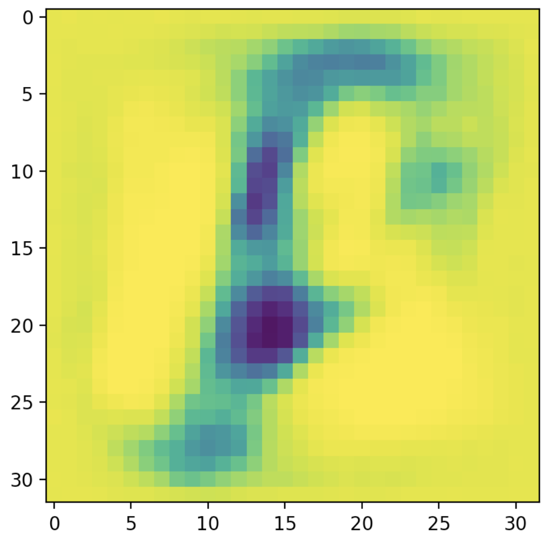

0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30

Class E

Class F

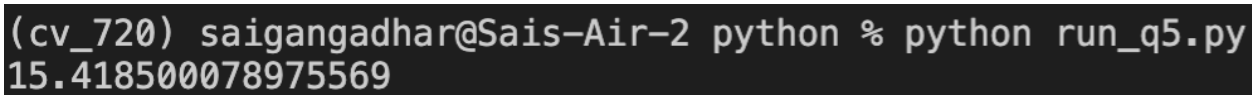





Observations on the reconstruction:

1. Since the model is made to learn all the classes there are some overlaps between the classes in the reconstructions
2. For example, the reconstruction of F has small faint vertical line making it appear as P. This is because there is quite some similarity between F and P. The same argument holds for E, it appears a little similar to B.
3. Therefore, the reconstructed image looks mostly similar to the original image, however with some smooth blend with other classes which appear similar

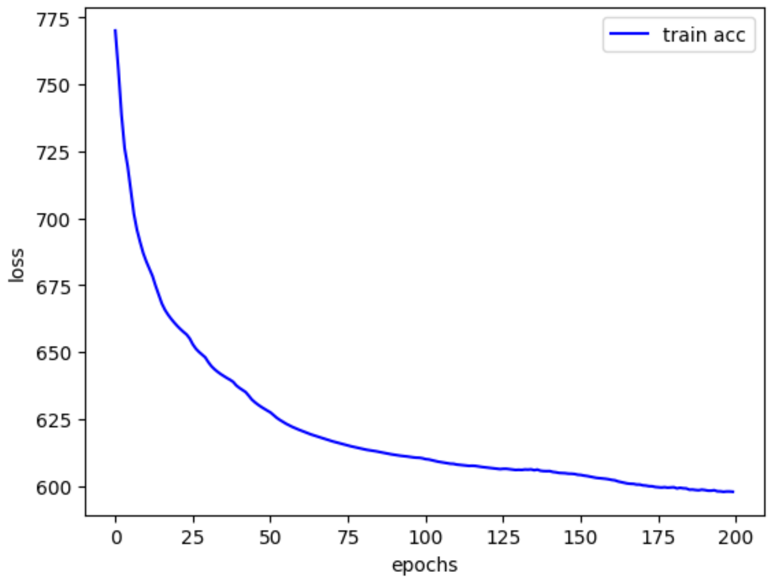
Q5.3.2)

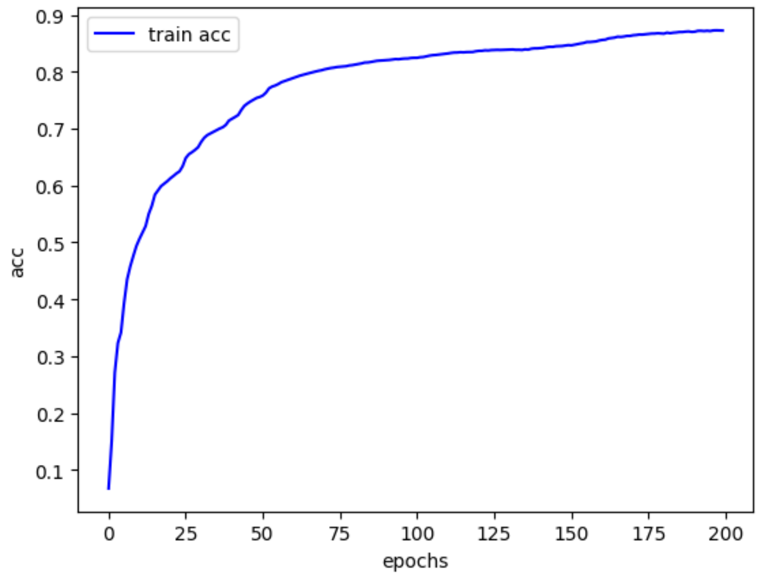
The achieved PSNR value for the reconstruction is:



```
(cv_720) saigangadhar@Sais-Air-2 python % python run_q5.py
15.418500078975569
```

6. Extra Credit
Q6.1.1)



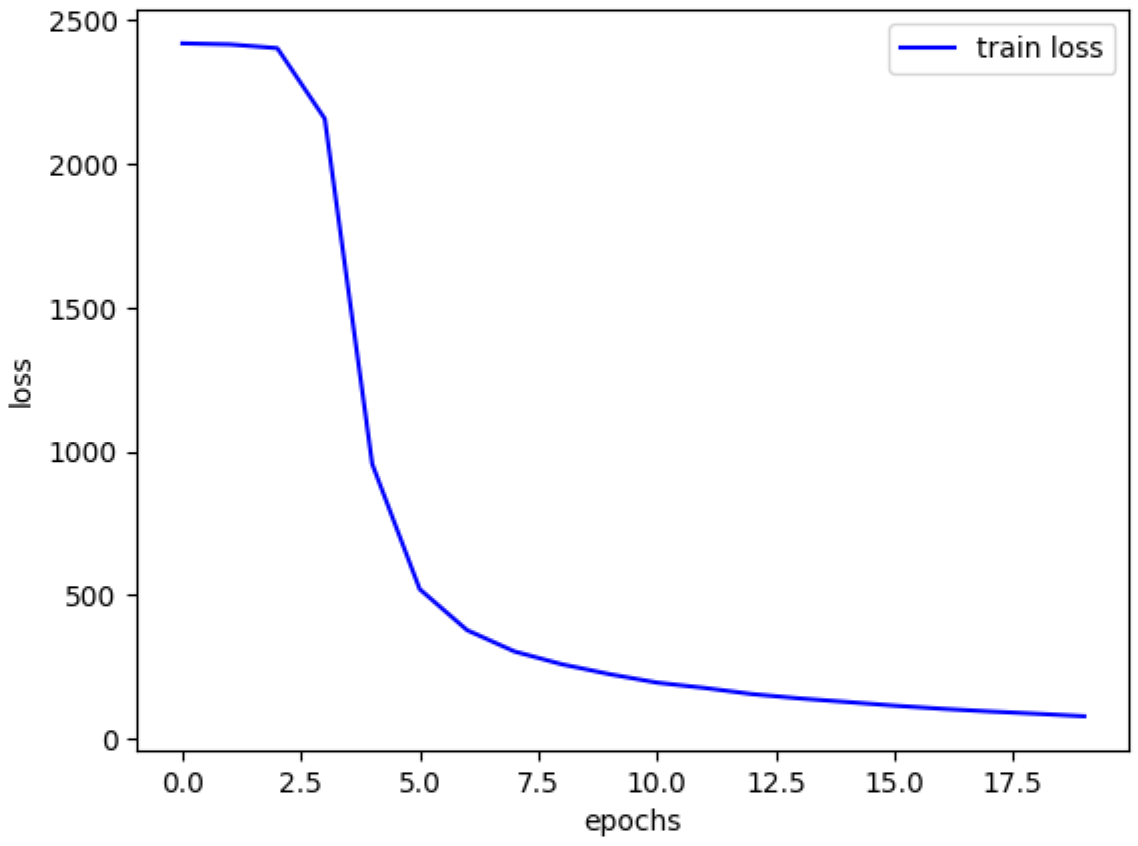


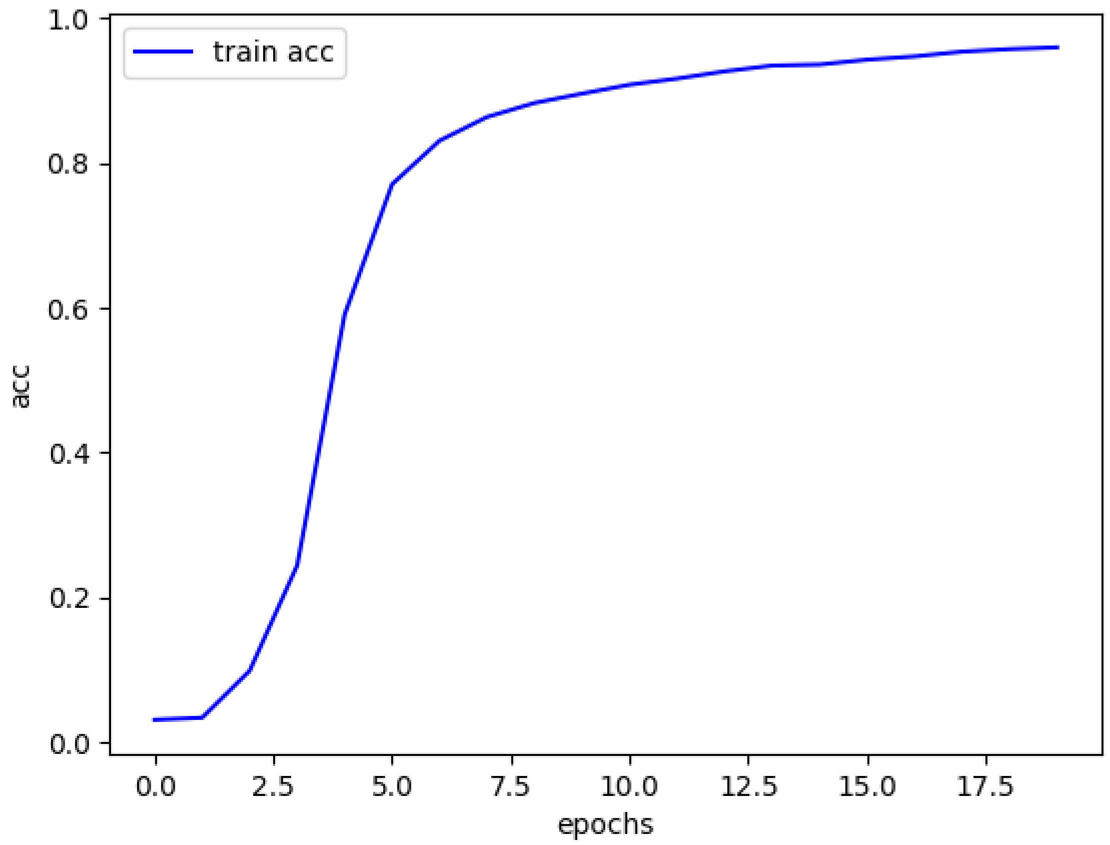
valid accuracy: 0.69

Q6.2)

Using CNN Architecture:

```
CNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1024, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=36, bias=True)
)
```





Achieved valid accuracy is 0.897

This is much better than what was achieved using MLP (0.69).