16720-A F20 Neural Networks for Recognition

Homework 5

Shayeree Sarkar

A) 1.1) We have to prove that softmax is invariant to translation

Now for each index 'i' in vector 'x',

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \text{--- (1)}$$

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

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

$$= \frac{e^{x_i} \cancel{e^c}}{\cancel{e^c} \sum_j e^{x_j}}$$

$$= \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \text{--- (2)}$$

Since Eqn (1) is equal to Eqn (2), hence

$$\text{softmax}(x) = \text{softmax}(x + c)$$

We often use : $c = -\max_i x_i \rightarrow$ since this prevents the explosion of exponential so that e^{x_i} & $\sum_j e^{x_j}$, don't overflow.

8) 1.2) → Since softmax $(x_i) = \frac{s_i}{S} = \left(\frac{s_i}{\Sigma s_i}\right)$ and $s_i = e^{x_i}$, each element of the softmax is in the range (0, 1) and the sum of all elements is 1.

→ One could say that "Softmax takes an arbitrary real valued vector 'x' & turns it into a probability distribution"

→ So, basically, the outcome frequency 'x_i' in the exponential form is $s_i = e^{x_i}$

The total outcome frequency is $S = \Sigma s_i$

$\left(\frac{s_i}{S}\right)$ → Normalized the frequency of each 'x_i' & hence o/ps the probability

Q1.3) In order to show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression,

We know, $x_{i+1} = w_i x_i + b_i$

Now for Multilayer networks:

$$y = w_n x_n + b_n$$

$$= w_n(w_{n-1} x_{n-1} + b_{n-1}) + b_n$$

$$= w_n w_{n-1} x_{n-1} + w_n b_{n-1} + b_n$$

$$= w' x_{n-1} + b'$$

$$= w'(w_{n-2} x_{n-2} + b_{n-2}) + b'$$

$$\cdots\cdots$$

$$= wx + b$$

Hence this is same as solving a linear regression problem

Q) 1.4)

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

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

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

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

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

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

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

Hence Shown.

Q) 1.5) Given, $y = Wx + b$, or we can write:

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

$$\frac{\partial y_j}{\partial W_{ij}} = x_i, \quad \frac{\partial y_j}{\partial x_i} = W_{ij}, \quad \frac{\partial y_j}{\partial b_j} = 1$$

Hence for all y:

$$\frac{\partial y}{\partial W} = \begin{bmatrix} x_1 & \cdots & x_1 \\ \vdots & \ddots & \vdots \\ x_d & \cdots & x_d \end{bmatrix} \in \mathbb{R}^{d \times K}$$

$$\frac{\partial y}{\partial x} = W_j \in \mathbb{R}^{d \times 1}$$

$$\frac{\partial y}{\partial b} = 1 \in \mathbb{R}^{K \times 1}$$

Now applying Chain Rule:

$$\frac{dJ}{dW} = \sum_{j=1}^{K} \left(\frac{\partial J}{\partial y_j}\right) \cdot \left(\frac{\partial y_j}{\partial W}\right) = x\delta^T \in \mathbb{R}^{d \times K}$$

$$\frac{\partial J}{\partial x} = \sum_{j=1}^{K} \left(\frac{\partial J}{\partial y_j}\right) \cdot \left(\frac{\partial y_i}{\partial x}\right) = W\delta \in \mathbb{R}^{d \times 1}$$

$$\frac{\partial J}{\partial b} = \sum_{j=1}^{K} \left(\frac{\partial J}{\partial y_j}\right)\left(\frac{\partial y_i}{\partial b}\right) = \delta \in \mathbb{R}^{K \times 1}$$

Q) 1.6 (1) The maximum derivative of a sigmoid function is 0.25.. When the activation is applied to the layers & the derivatives are taken, then entries in 'x' will drop quickly. If it is used for many layers, it might lead to the "vanishing gradient" problem.

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

Hence o/p range for tanh is $(-1, 1)$

For sigmoid, the o/p range is $(0, 1)$

We might prefer tanh because 'x' is +ve, tanh is +ve & when 'x' is -ve, tanh is -ve

(3) The derivative of tanh varies b/w $(0, 1)$ unlike sigmoid where its $(0, 0.25)$.

Hence when we take derivatives in multiple layers, it is less likely that tanh will lead us to face the "vanishing gradient" problem.

(4) For tanh:,

We know for sigmoid:

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

$$\therefore \quad 2\sigma(x) - 1 = \left(\frac{2}{1+e^{-x}}\right) - \left(\frac{1+e^{-x}}{1+e^{-x}}\right)$$

$$\Rightarrow 2\sigma(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}} \quad \text{——} \quad (1)$$

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

Comparing eqns (1) & (2), we can write that:

$$\tanh(x) = 2\sigma(2x) - 1$$

Hence shown.

3) 2.1.1) If a network is initialized with all zeros, all the outputs generated from the network will be zero & the o/p probability vector will have the same entries. This will mislead back propagation & gradient descent, since the weights will be updated similarly, causing the

layers to do the same computations.

Q) 2.1.3) Initialization of the network with random numbers prevents the symmetry of the matrix so that same computation across the layers can be avoided

Scaling the initialization based on layer size can help keep the variance around the desired values when performing forward & backward propagation

Q)3.2)

Best predictions are obtained with a learning rate of 3e-3, hidden size=128. We observe that in the for Fig 1, where learning rate is 10 times the best learning rate, the training and validation accuracy fluctuate greatly and the loss curve doesn’t converge.

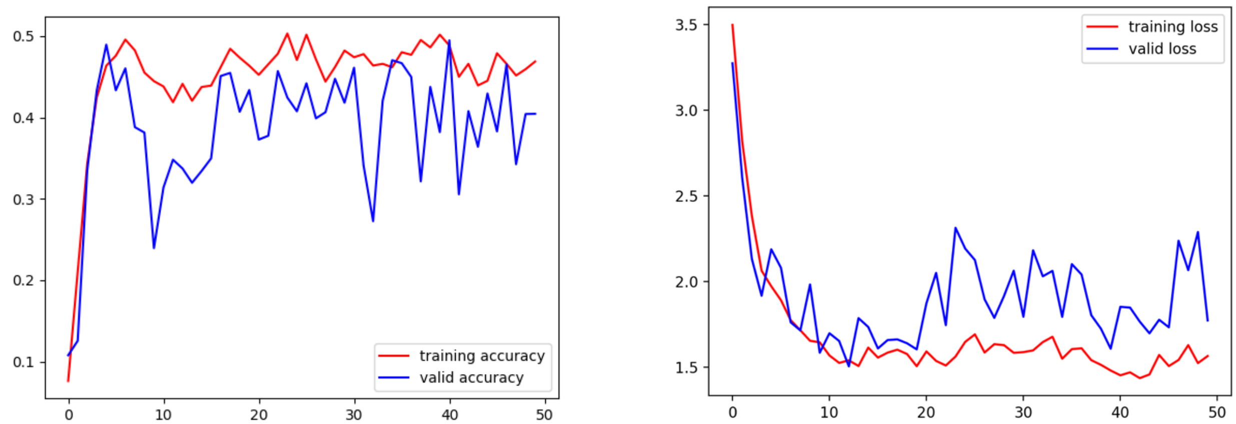


Fig 1. Learning Rate = 3e-2

For Fig 2, where learning rate is the best learning rate =3e-3 , the training and validation accuracy increase smoothly and the both the loss curves converges smoothly as well .

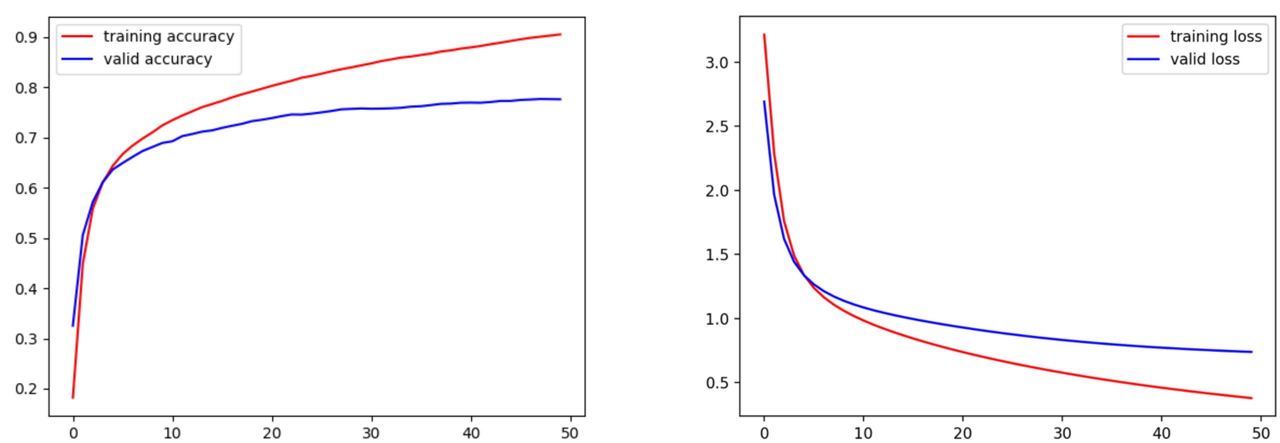


Fig 2. Learning Rate = 3e-3

For Fig 3, where learning rate is one tenth of the best learning rate , that is, 3e-4 , the training and validation accuracy increase smoothly and the both the loss curves doesn't converges to the optimum value.

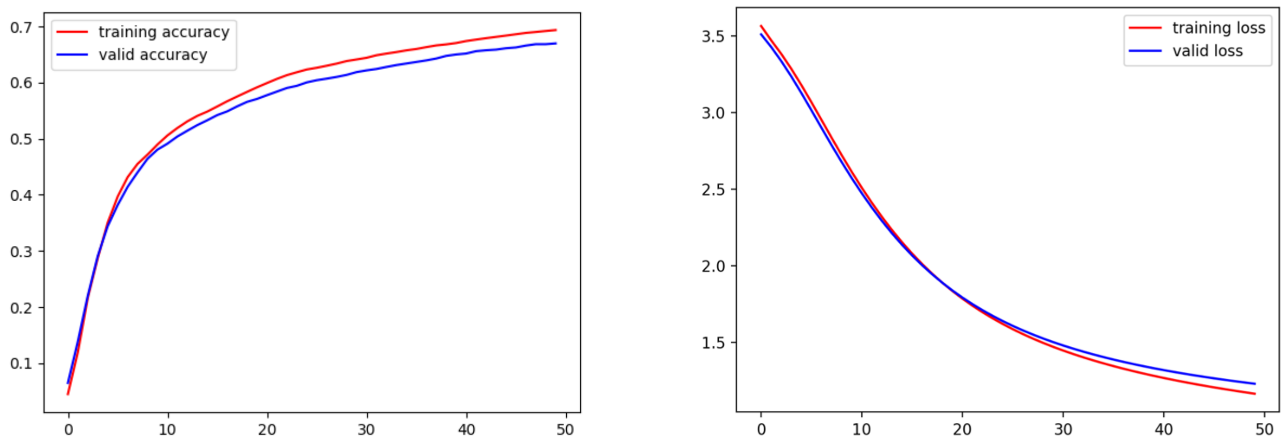


Fig 2. Learning Rate = 3e-4

Q)3.3). The first figure is that of the initialized weights and the second figure shows the learned weights. Since initially we had initialized the layers with random uniform distribution, the initial weights are random noisy points. However after 50 epochs, we can see the network has learned some clear patterns.

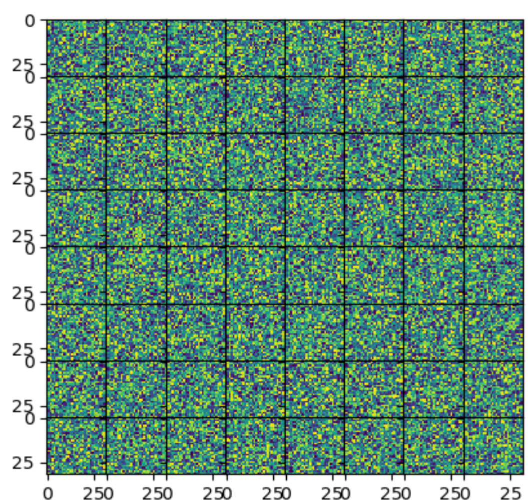


Fig 1. Initial weights

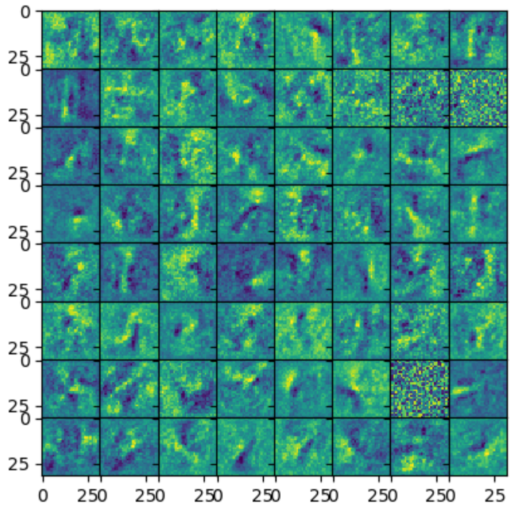


Fig 2. Learned weights

Q)3.4). Below is the obtained Confusion Matrix. The brighter the grid, the greater no of true positives it has. The most commonly confused classes included ‘O’ and ‘Q’, ‘5’ and ‘S’, ’2’ and ‘Z’

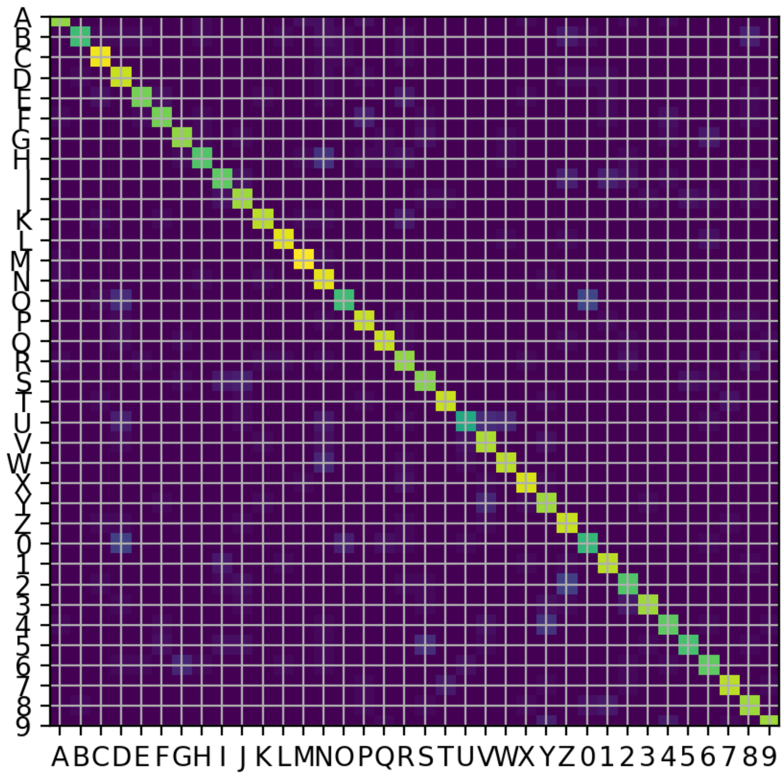
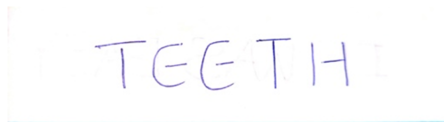


Fig 1. Confusion Matrix

Q)4.1) The two big assumptions that are made in are :

- Every letter is fully connected. However this might not be the case especially if the parts in a single letter are not fully connected.



- Two different letters cannot be connected. Here since we extract letters wrt the connections hence if two letters overlap, then the method cannot segregate the two.

Q).4.3) Results from q4.py

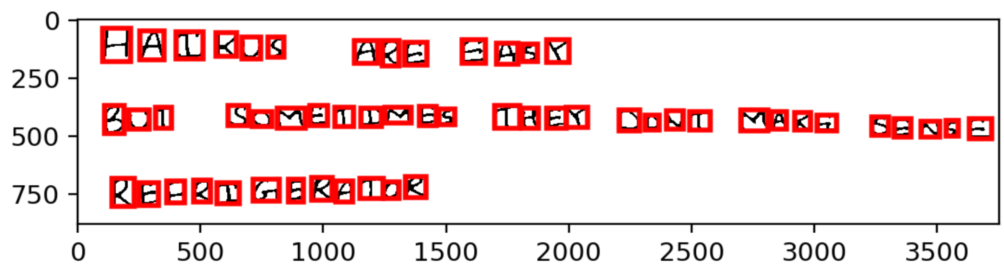


Fig 1. Corresponding to 03_haiku.jpg

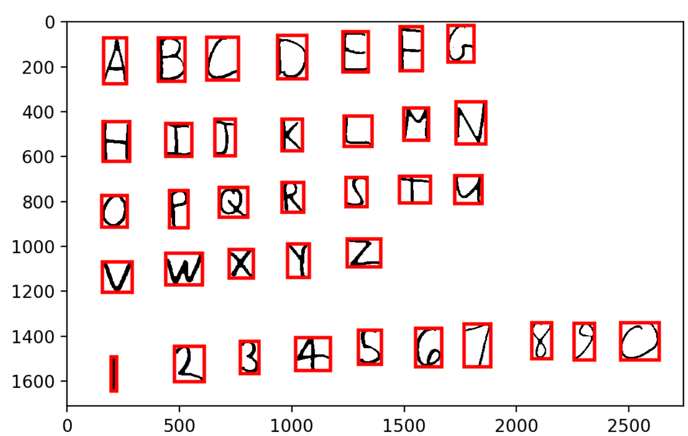


Fig 2. Corresponding to 02_letters.jpg

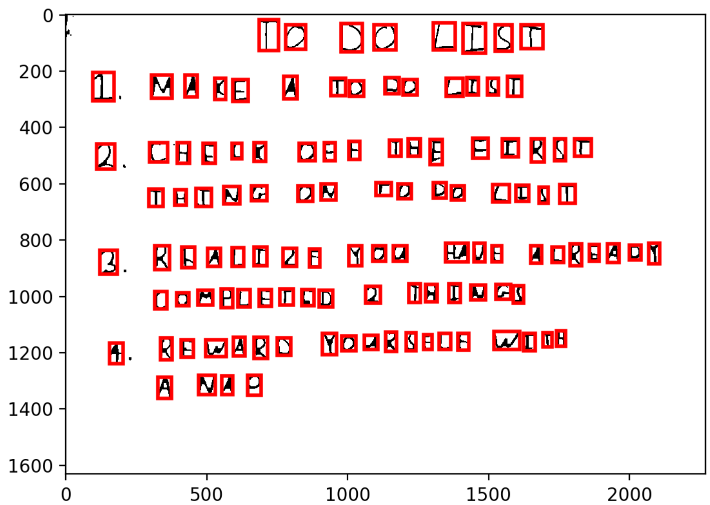


Fig 3. Corresponding to 01_list.jpg

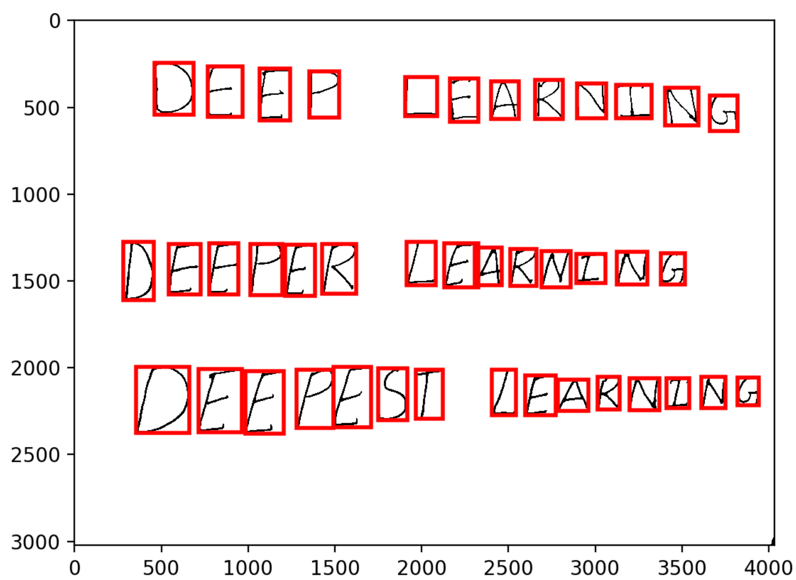


Fig 4. Corresponding to 04_deep.jpg

Q)4.4). The following corresponds to 04_deep.jpg

DEEF LEARMING
DEFPER LEARNING
DE8PEST LEARNING

The following corresponds to 01_list.jpg

TQ DQ LIST
I NAKE A TQ QQ LIST
2 CHLCK DFF 3HE FIRST
THING QN TQ DQ LIST
3 RLALIZE YQU HAVE ALR6ADT
GQMPLGILD 2 IHINGS
4 RFWARD YQWRSFLF WIIB
A NAP

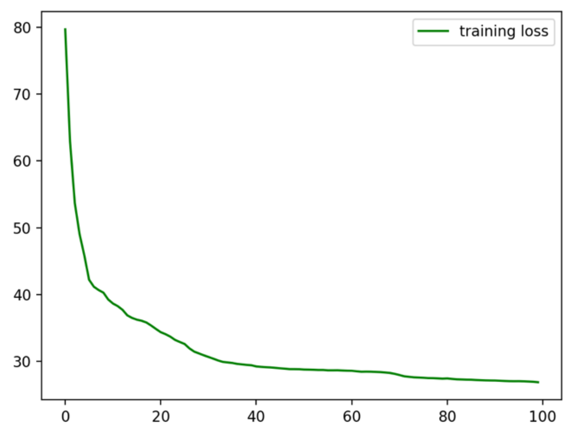
The following corresponds to 02_letters.jpg

A B L D E F G
H I I K L M N
Q P Q R S C T W
V W X Y Z
1 Z 3 G S G 7 8 9 D

The following corresponds to 03_haiku.jpg

HAIKUS ARG EAGY
BWT SQMETIMBS TREY DDWT MAKG SGMGE
REGRIGBRATQR

Q)5.2) The Y-axis represents average loss whereas the X-axis represents number of epochs for training. We see that with the vanilla parameters the networks, the loss curve is smooth and decreases over the epochs and actually almost saturates at the 25th epoch itself and slowly converges thereafter .



Q)5.3.1) Having selected 5 classes from the total 36 classes in the validation set ,for each selected class 2 validation images and their reconstruction has been shown in the following figures:

Fig 1. Class 'A'

Fig 2. Class 'H'

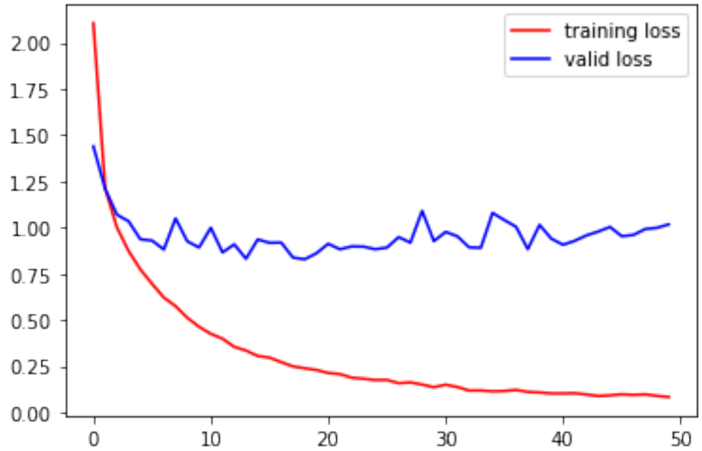
Fig 3. Class 'O'

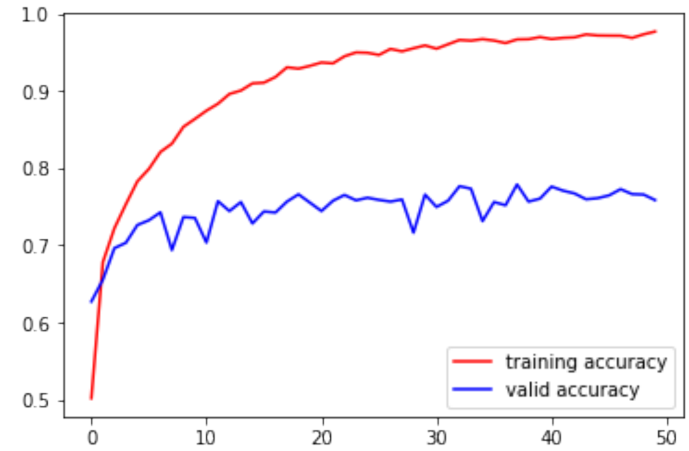
Fig 4. Class 'V'

Fig 5. Class '2'

Q)5.3.2) The PSNR score obtained from the autoencoder across all images in the validation set is 16.6334.

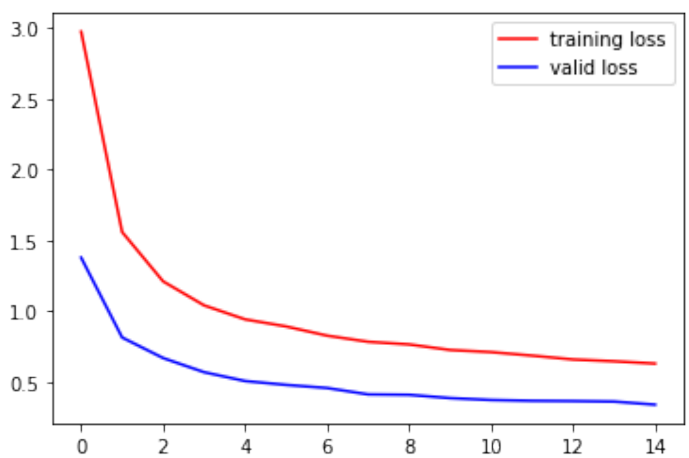
Q) 6.1.1) For our fully-connected network on the included NIST36 in PyTorch using the same vanilla hyperparameters as that in the question run_q3.py, the plots for loss and accuracy as found below:

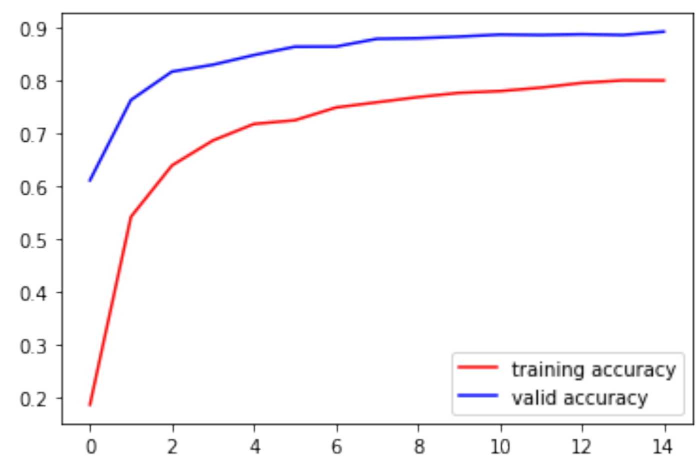




The Validation Accuracy is 77 percent while the training accuracy reaches almost a 100% which is because the model overfits.

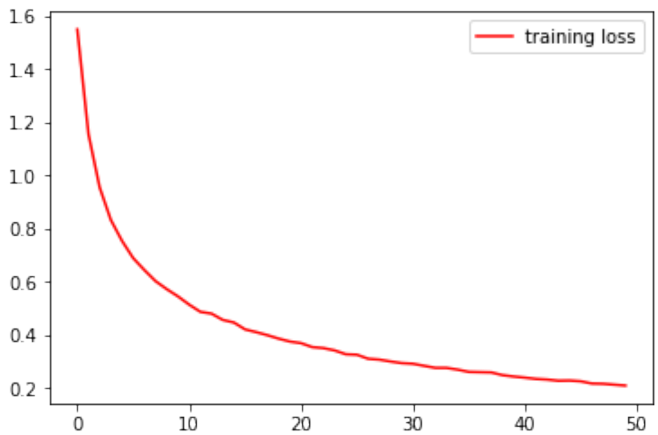
Q) 6.1.2) We apply CNN based architecture on the included NIST36 in PyTorch, the plots for loss and accuracy as found below:

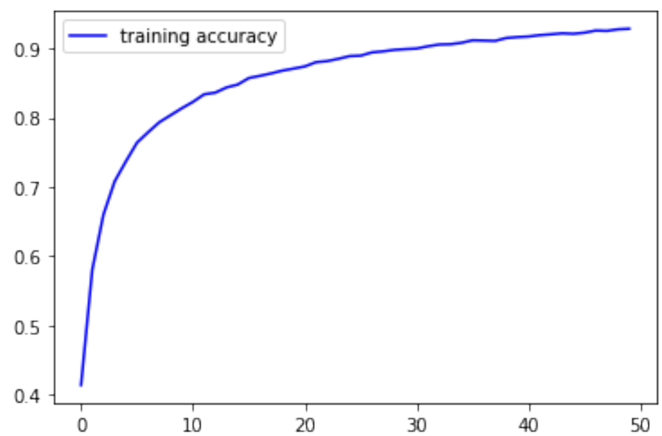




Now if we compare the performance of this CNN to the MLP architecture used in q6.1.1 we instantly notice that the CNN outperforms the MLP in terms of accuracy , which is about 90%, over only 77% with the MLP. We also notice that the network converges much faster as in about, one third the time it took in the MLP architecture used

Q) 6.1.3) We apply CNN based architecture on the CIFAR-10 dataset in PyTorch, the plots for loss and accuracy as found below:





Q) 6.1.4) When we apply ConvNet architecture to the Sun Database System for Scene classification then we see that using Resnet 50 architecture as a backbone architecture network we are able to achieve an accuracy of 65 % on the training dataset compared to the 56.5% using vanilla hyperparameters in homework 1.