16720-B Computer Vision: Homework 5

Nithin Kumar Sathish
AndrewID: nsathish

November 23, 2019

Problem Q1.1.

We start deriving from the right hand side,

$$\begin{aligned}
\text{RHS} &= softmax(x_i + c) && (1)\\
&= \frac{e^{(x_i+c)}}{\sum_j e^{(x_j+c)}} && (2)\\
&= \frac{e^{(x_i)}e^{c}}{\sum_j e^{(x_j)}e^{c}} && (3)\\
&= \frac{e^{(x_i)}}{\sum_j e^{(x_j)}} && (4)\\
&= softmax(x_i) && (5)\\
&= \text{LHS} && (6)
\end{aligned}$$

This shows that the softmax function is invariant to translation.

Using the value of $c = -\max x_i$ is important for preventing very high values of the exponential term. Sometimes it may happen that the values cannot be represented by the computer on which we are implementing the function. This is ensures stabiliy and at the same time precision as the we have proven above that the ratio remains unchanged.

Problem Q1.2.

1. If we consider the range of of each input element to be $-\infty < x_i < \infty$ then, the value of each output elements is $0 < softmax(x_i) < 1$.

   The sum over all elements is given by,

   $$\sum_i softmax(x_i) = 1 \tag{7}$$

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

3. The multi-step process for computing the softmax is as follows:

   (a) First step is to scale a single element of interest of the input value using exponent.

   (b) Then, we do the same thing for all elements and then sum them.

   (c) The fraction of the numerator and the denominator gives us the probability of x_i.

Problem Q1.3.

Let us consider the simplest form of linear regression,

$$y = Wx + b \tag{8}$$

In a multilayer neural network, the pre-activation can to the hidden layer is,

$$u^{(1)} = W^{(1)}x + b^{(1)} \tag{9}$$

since there is no non-linear activation function,

$$h^{(1)} = u^{(1)} \tag{10}$$
$$u^{(2)} = W^{(2)}h^{(1)} + b^{(2)} \tag{11}$$

From equation (9),(10) and (11) we get,

$$h^{(2)} = u^{(2)} \tag{12}$$
$$= W^{(2)}[W^{(1)}x + b^{(1)}] + b^{(2)} \tag{13}$$
$$= W^{(2)}W^{(1)}x + W^{(2)}b^{(1)} + b^{(2)} \tag{14}$$
$$= [W^{(2)}W^{(1)}]x + [W^{(2)}b^{(1)} + b^{(2)}] \tag{15}$$

The above equation is in the same form as that of a linear regression shown above.

Thus, that shows that without non-linear activation function, a multilayer neural network is just like linear regression.

Problem Q1.4.

Starting with the definition of sigmoid functions,

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{16}$$

$$\frac{d\sigma(x)}{dx} = \frac{(1+e^{-x}).\frac{d(1)}{dx} - \frac{d(1+e^{-x}).1}{dx}}{(1+e^{-x})^2} \tag{17}$$

$$= \frac{(1+e^{-x}).0 - (-e^{-x})}{(1+e^{-x})^2} \tag{18}$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} \tag{19}$$

rearranging the terms,

$$= \frac{1}{(1+e^{-x})}\Big[\frac{1+e^{-x}-1}{1+e^{-x}}\Big] \tag{20}$$

$$= \frac{1}{(1+e^{-x})}\Big[1 - \frac{1}{(1+e^{-x})}\Big] \tag{21}$$

$$= \sigma(x)[1-\sigma(x)] \tag{22}$$

Problem Q1.5.

We have been given,

$$y_j = \sum_{i=1}^{d} x_i W_{ij} + b_j \tag{23}$$

$$J = f(y_j) \tag{24}$$

Taking derivative with respect to scalars first and then reforming the matrix form. First w.r.t weights

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial y_j}\frac{\partial y}{\partial W_{ij}} \tag{25}$$

$$\tag{26}$$

$$= \delta x_i \tag{27}$$

then w.r.t inputs,

$$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{28}$$

$$\tag{29}$$

$$= \delta . W_{ij} \tag{30}$$

finally w.r.t the bias term,

$$\frac{\partial J}{\partial b_j} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b_j} \tag{31}$$

$$\tag{32}$$

$$= \delta \tag{33}$$

Taking derivatives for matrices,

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} \tag{34}$$

$$\tag{35}$$

$$= x.\delta^T \tag{36}$$

also,

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} \tag{37}$$

$$\tag{38}$$

$$= W.\delta \tag{39}$$

and,

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial b} \tag{40}$$

$$\tag{41}$$

$$= \delta \tag{42}$$

Problem Q1.6.

1. In a deep neural network the gradient with respect to a weight matrix can be written like this,

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial f}\frac{\partial f}{\partial y}\frac{\partial y}{\partial h^{(t-1)}}\frac{\partial h^{(t-1)}}{\partial \sigma(a^{(t-1)})}\frac{\partial \sigma(a^{(t-1)})}{\partial h^{(t-2)}}\ldots\frac{\partial \sigma(a^{(1)})}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial W^{(1)}}$$

   Since we know that the derivative of most activation functions lie between 0 and 1. Hence, as the number of the layers increase, the more the number of multiplications with the derivative of the activation functions in each layer which will eventually make the gradients zero as we go towards the input layer.

2. The range of tanh is -1 to 1. Tanh is preferred over sigmoid function because it has more more "memory". In the sense, that sigmoid saturates after a few steps, but tanh can give useful information longer than sigmoid

3. The range of derivative of tanh is given by from 0 to 1. Since derivative of tanh is higher than sigmoid. And also because it can give useful information longer than sigmoid , it has a less of a vanishing gradient problem.

4. We know that the sigmoid function can be written as

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

   Hence,

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

Problem Q2.1.1.

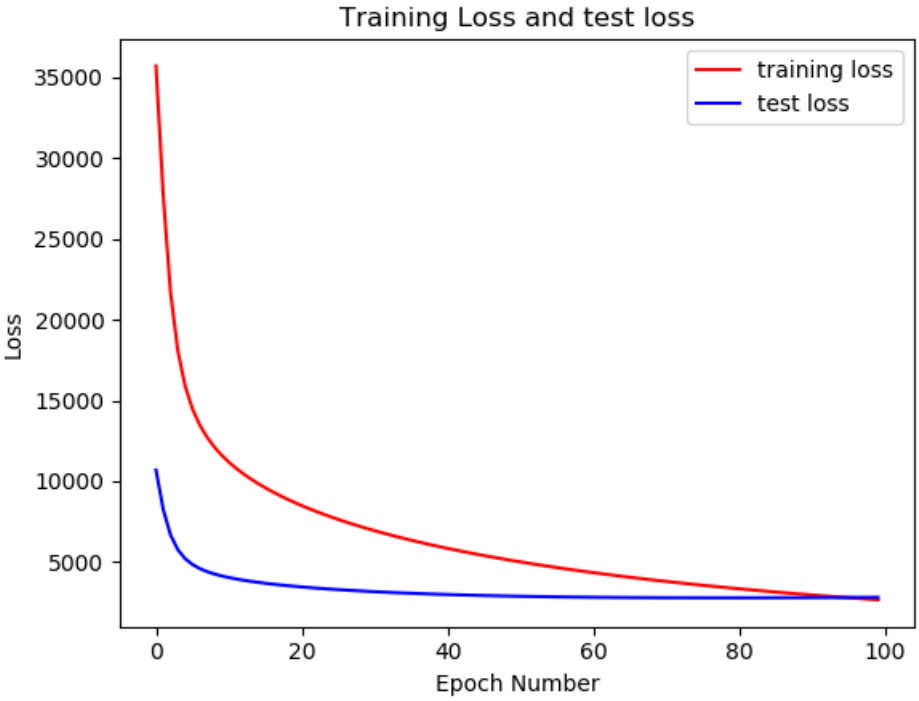
Say we have a neural network with sigmoid activation functions. If the weights are initialized to zero the output of the sigmoid function will be 0.5 for all the hidden units. Hence, all the hidden units in a layer will learn the samething and not something new. This will lead to gradient getting stuck in a local mininma.

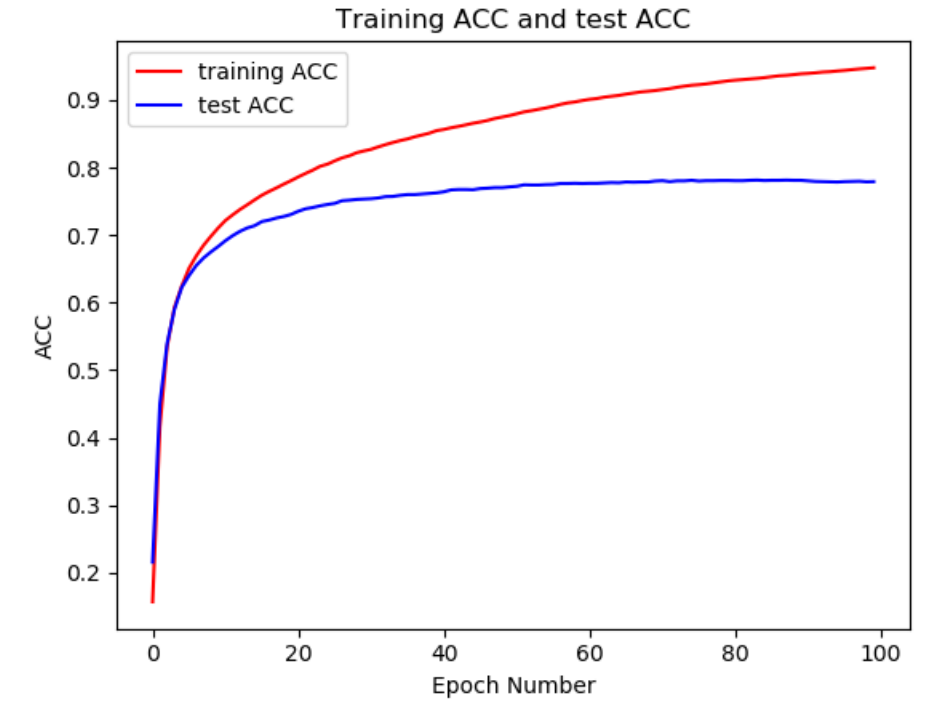
Problem Q2.1.3.

Initializing the weights randomly allows the hidden units in the neural network learn different things and gives a higher probability of avoiding the local minima problem as stated above.

The paper states that the variance of the gradients decreases as we move output layer toward the input layer. Since we want the variance to remain the same in every layer, we scale the initialization depending on the layer size

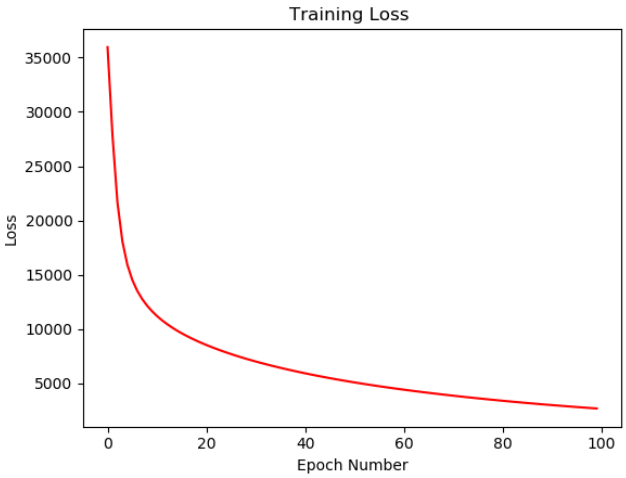
3.1.1

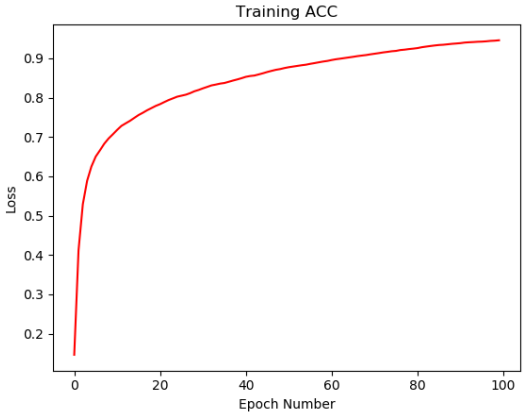

Training Loss and test loss
training loss
test loss
35000
30000
25000
20000
15000
10000
5000
Loss
0
20
40
60
80
100
Epoch Number


Training ACC and test ACC
training ACC
test ACC
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
ACC
0
20
40
60
80
100
Epoch Number

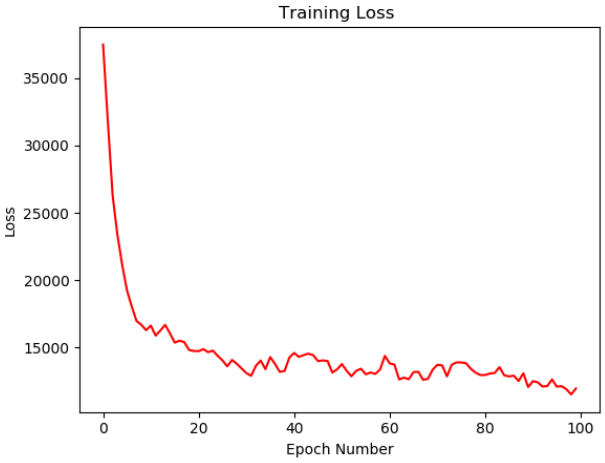
3.1.2

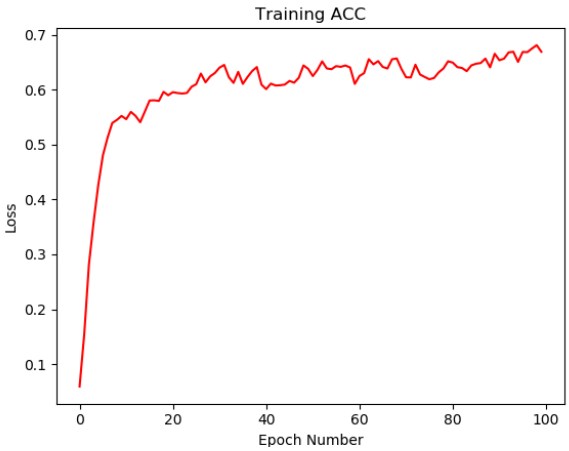
Lr = 3e-3



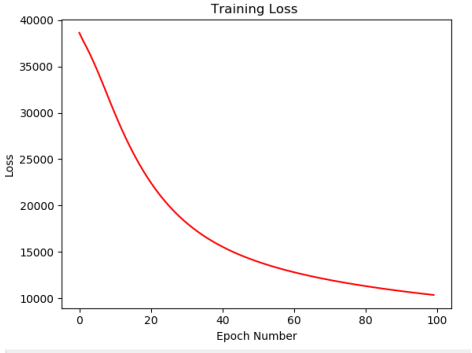


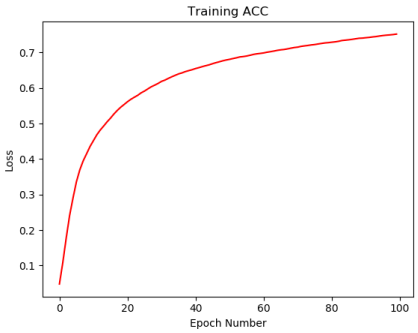
Lr=3e-2





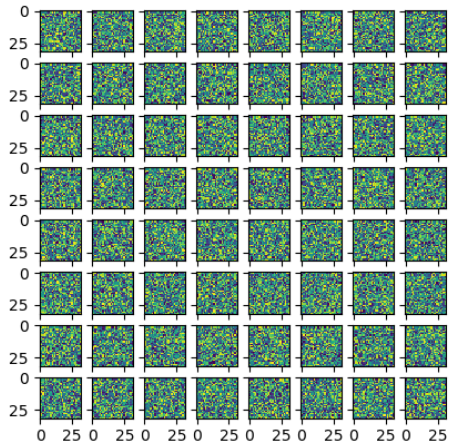
Lr =3e-4

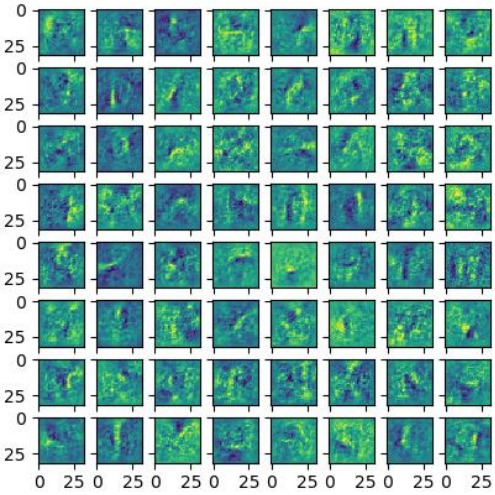




Final Accuracy = 0.7611

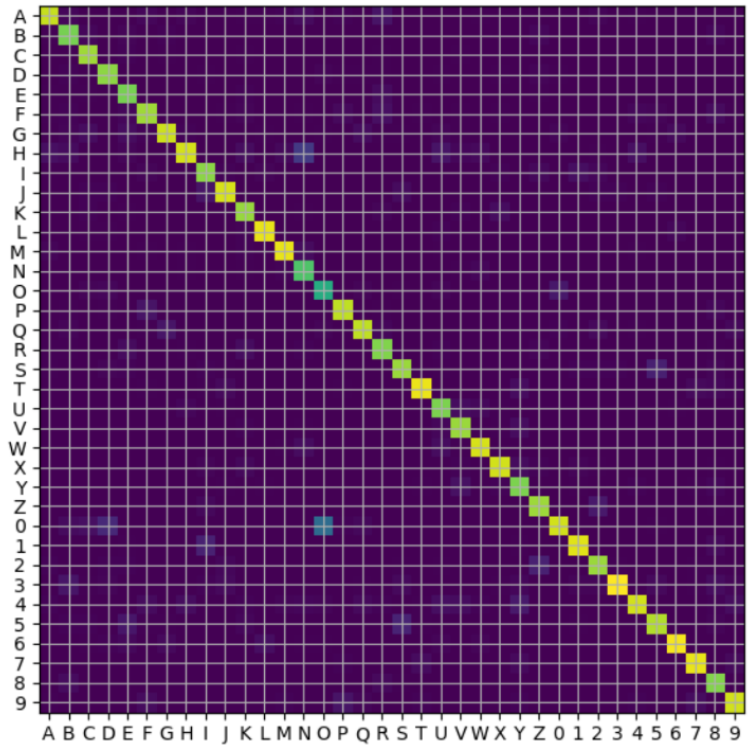
3.1.3 Weights Visualization





From the two images it is clear that the weights randomly initialized is just noise while after training the weights seem to learn some patterns

3.1.4 Confusion Matrix



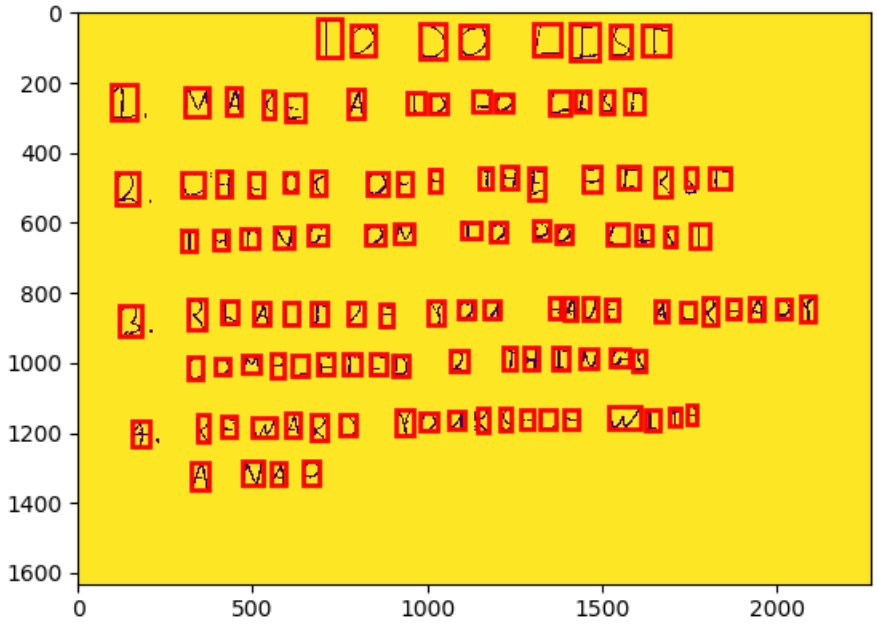
From the matrix we can say that the most commonly confused classes are ‘0’ and ‘O’ , ‘S’ and ‘5’, ‘1’ and ‘I’.

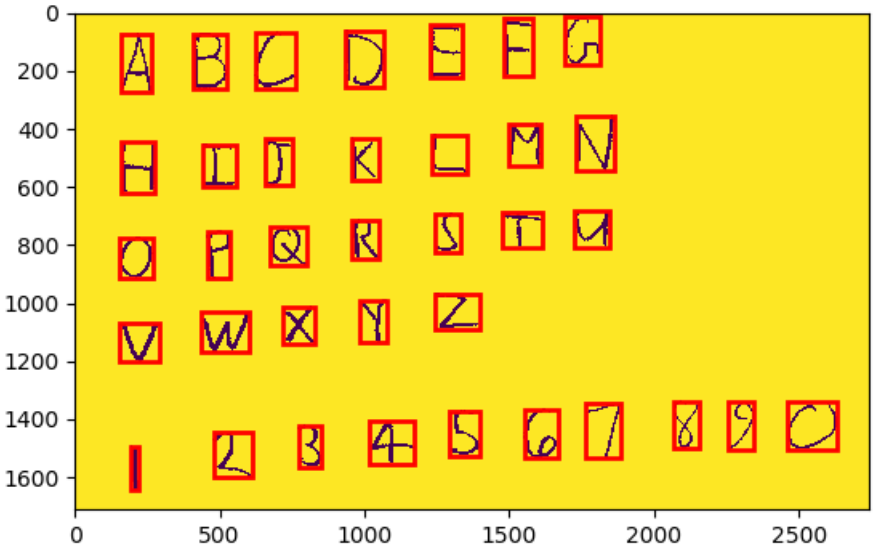
4.1

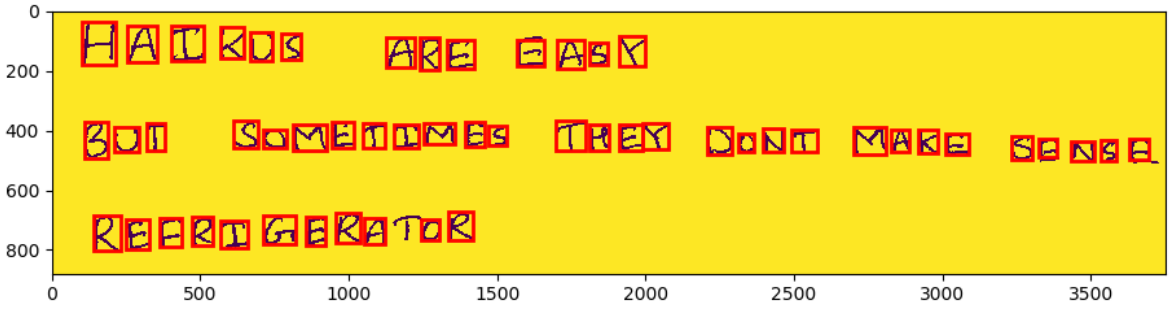
The three main assumptions are:

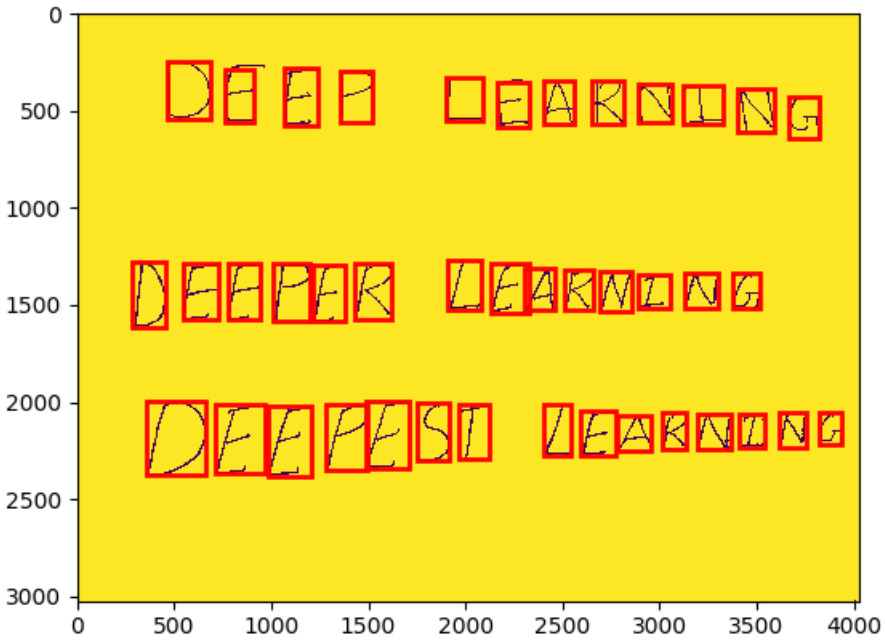
1. All the characters are of the same size
2. There are no two characters which are connected to each other

4.3





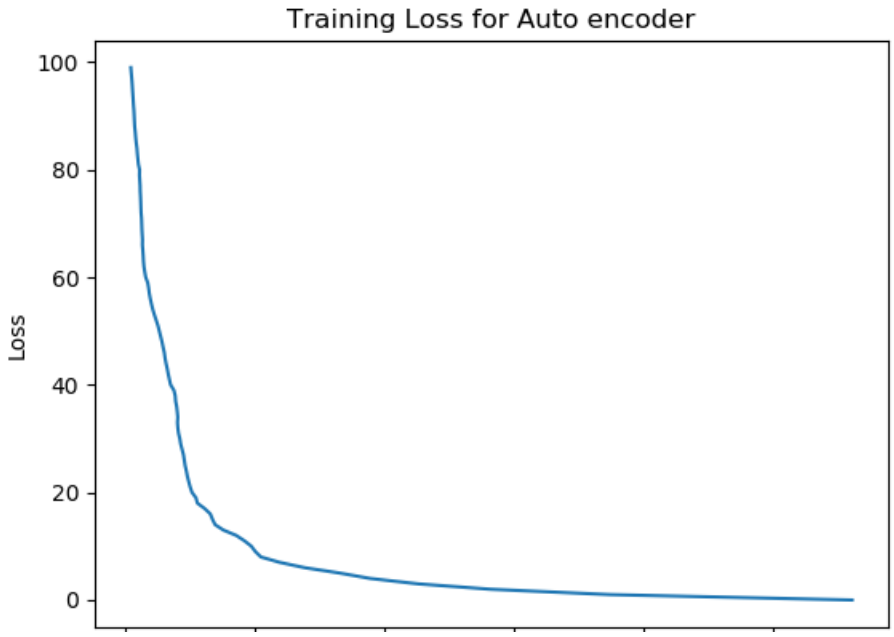

HAIKUS ARE EASY
BUT SOMETIMES THEY DONT MAKE SENSE
REFRIGERATOR


DEEP LEARNING
DEEPER LEARNING
DEEPEST LEARNING

4.4

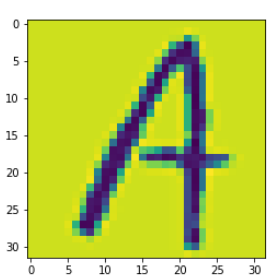
```
Current image: 01_list.jpg
Num letters in the image:  115
T0J0LI5T
IMAKEAT0D0LIST
2CHEEK0FFTHEFIR5T
THING0NT0D0LIST
3REALI2EY0UHQVEALKEADT
C0MPLETED2THINGS
4REW2RDX0UKSELFWITH
ANAP

Current image: 02_letters.jpg
Num letters in the image:  36
2BCJEFG
HIJKLMN
QPQR5TU
VWXYZ
IZ3GSG7X7C

Current image: 03_haiku.jpg
Num letters in the image:  53
HAIKUSAREBAGX
5UTSOMETIMESTHEXDONTMAKESGNGG
REFRIGERA0R

Current image: 04_deep.jpg
Num letters in the image:  41

CVHFLKARMING
DHFTFKLFAKNING
5FFFFSTLEARNING
```

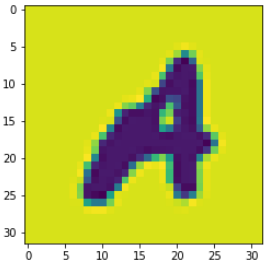
5.2 AutoEncoders

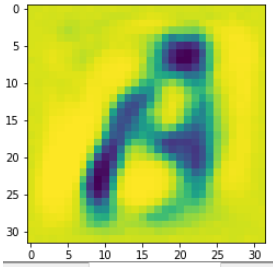


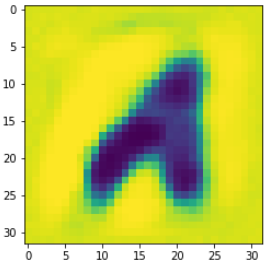
Overall as the number of epochs increased the training loss of the autoencoder kept decreasing.

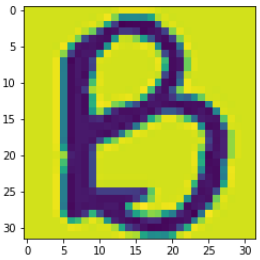
5.3.1 Images and Reconstructed Images for AutoEncoder

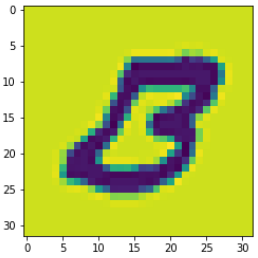


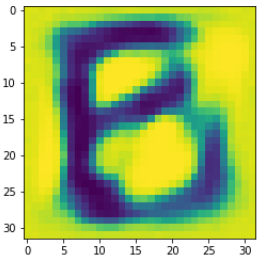


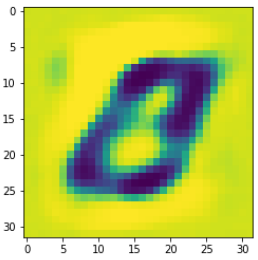


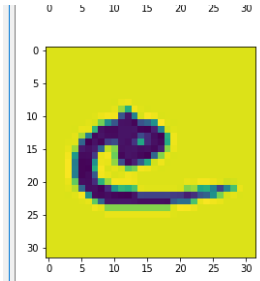


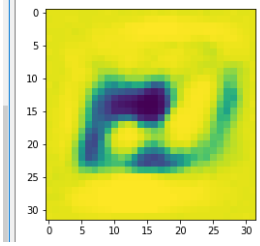


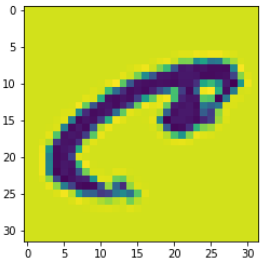


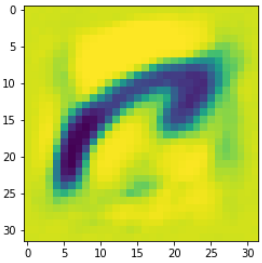


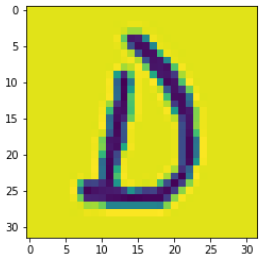


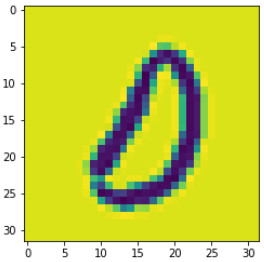


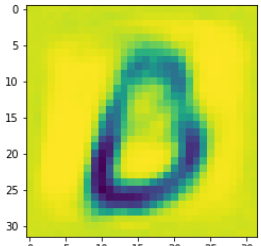


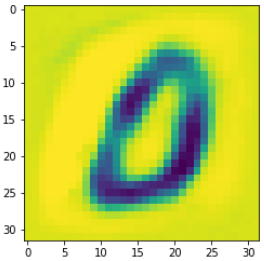


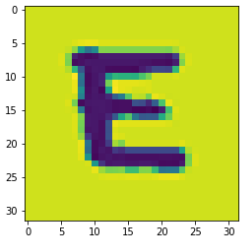


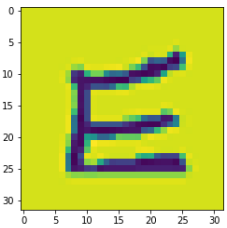


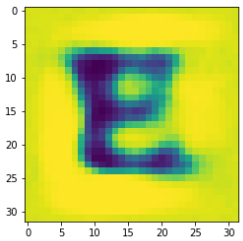


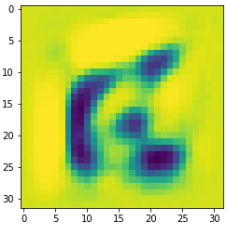












Compared to the original images the reconstructed images look blurred.

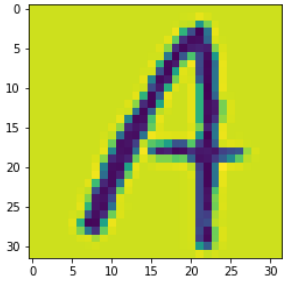
5.3.2 PSNR

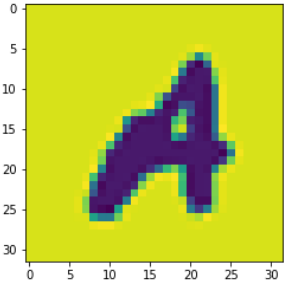
Average PSNR = 15.943

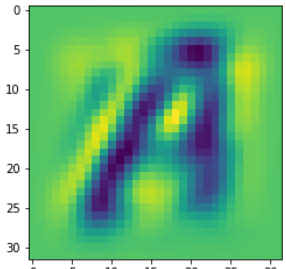
6.1 Projection Matrix

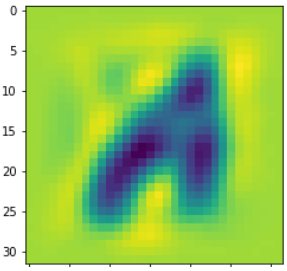
The size of projection matrix is 1024X32. The rank is 32

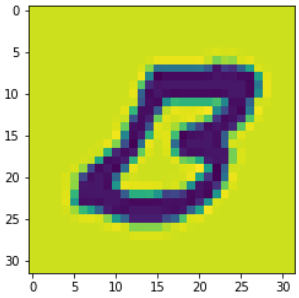
6.2 Images and Reconstructed Images for PCA

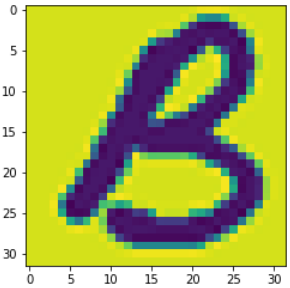


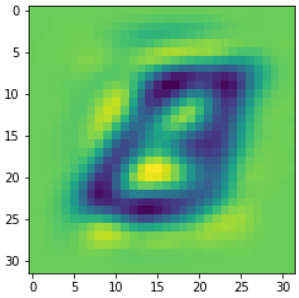


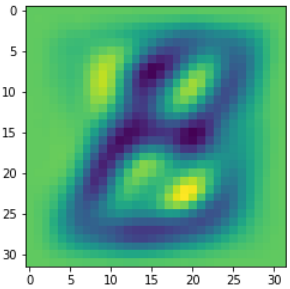


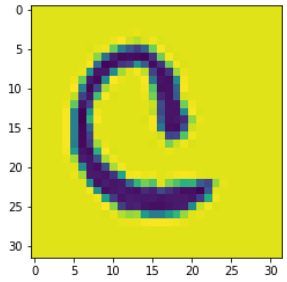
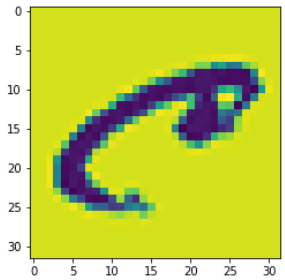
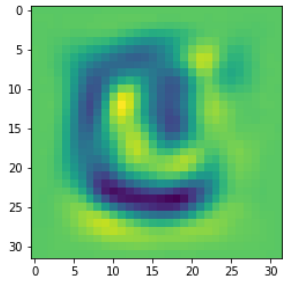
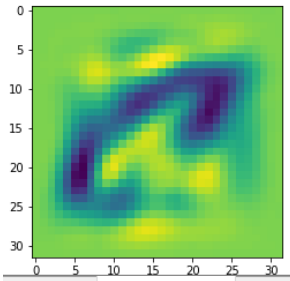
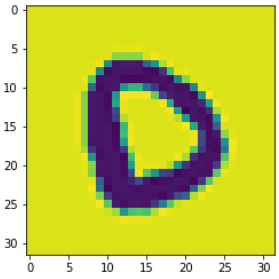
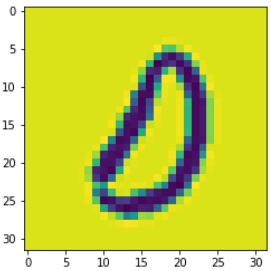
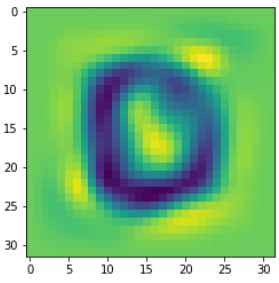
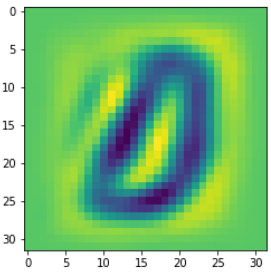


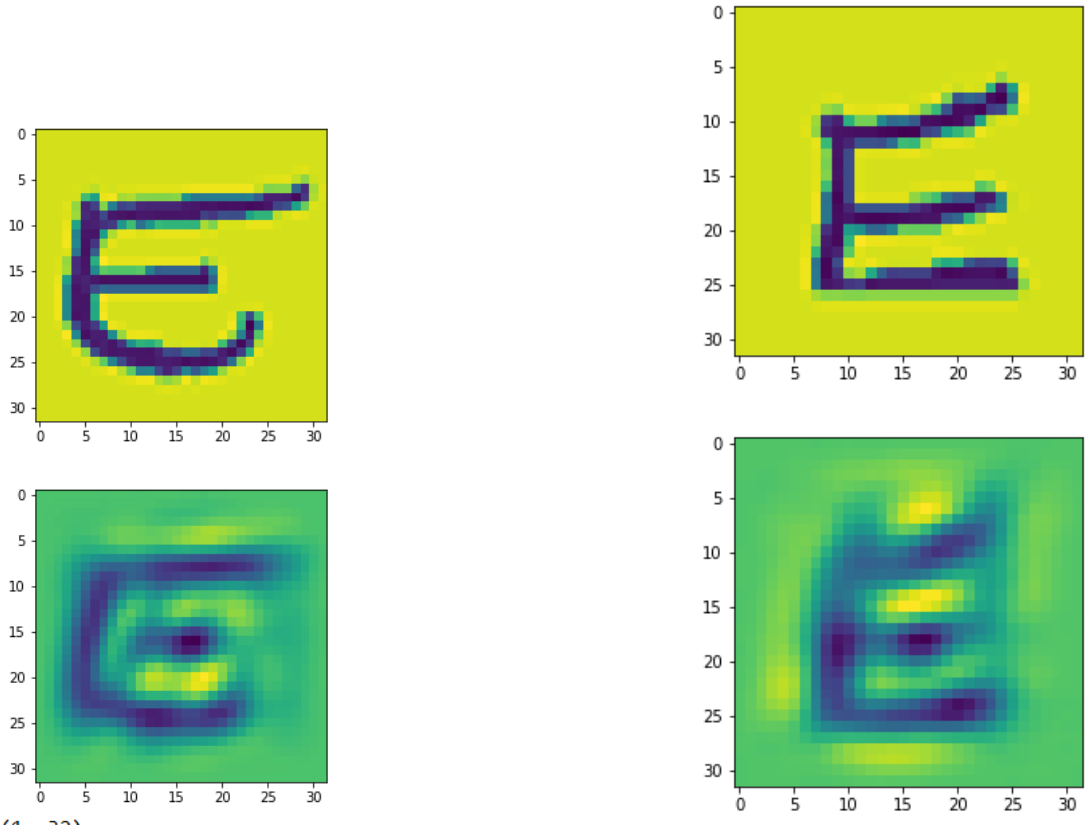








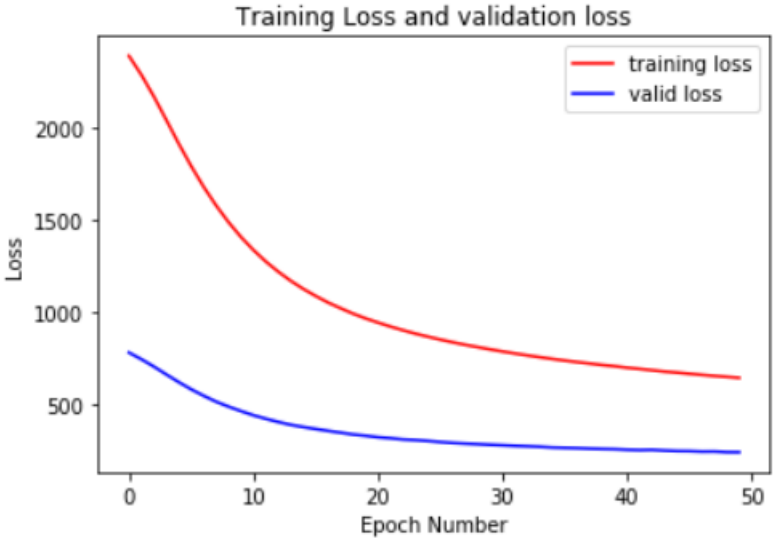









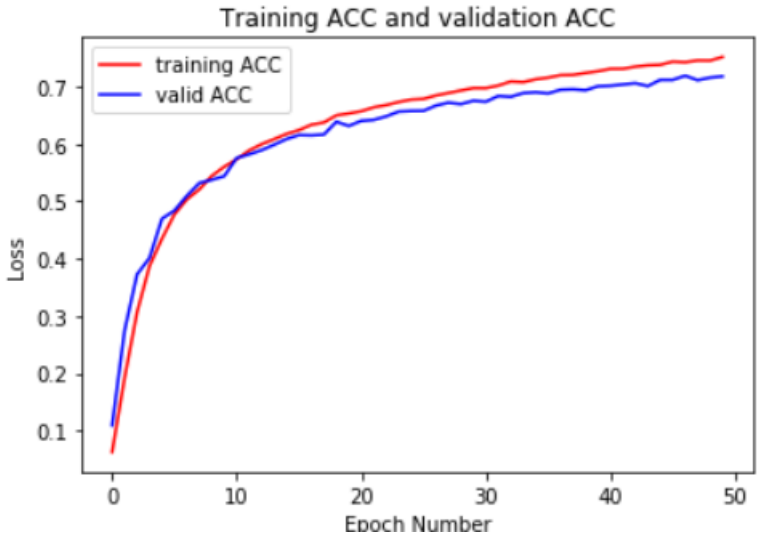


The reconstructed images are slightly blur but the reconstruction is much better compared to auto-encoder

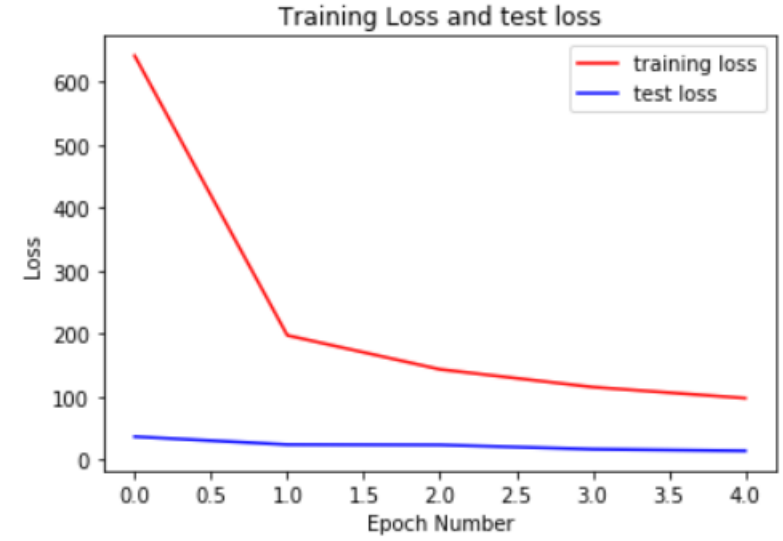
PSNR: 16.277549

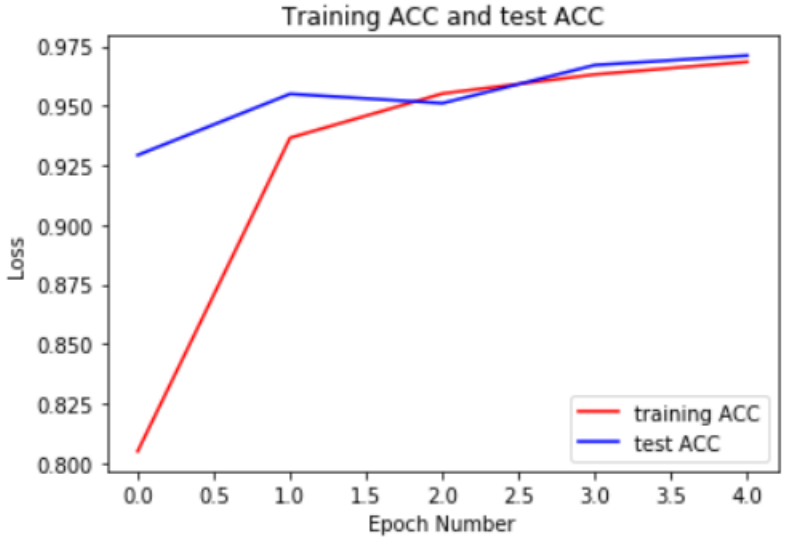
7.1.1 Fully connected Network on NIST36



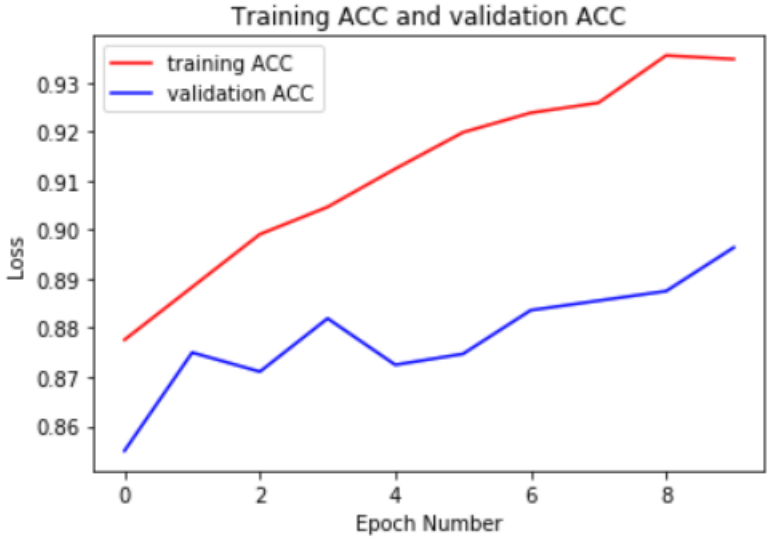


7.1.2 CNN on MNIST

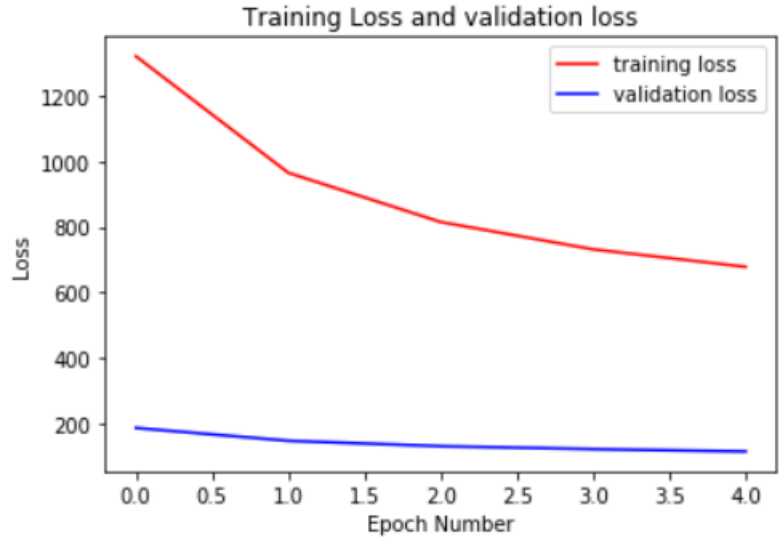


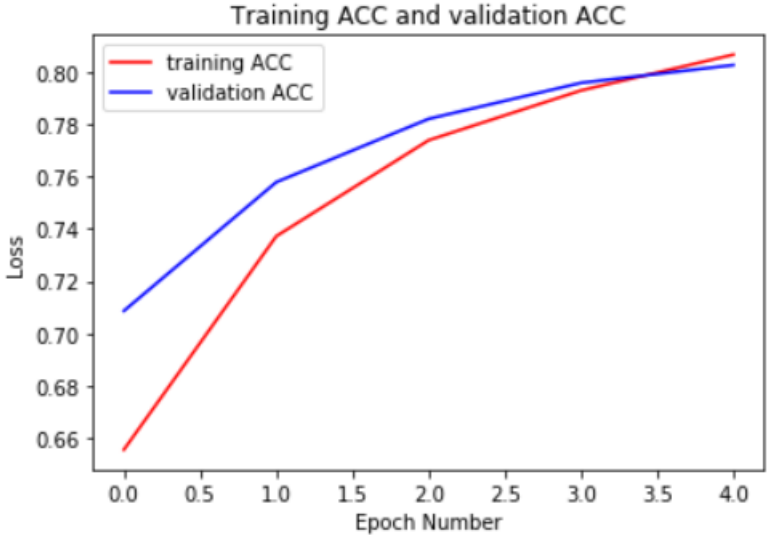


7.1.3 CNN on NIST36

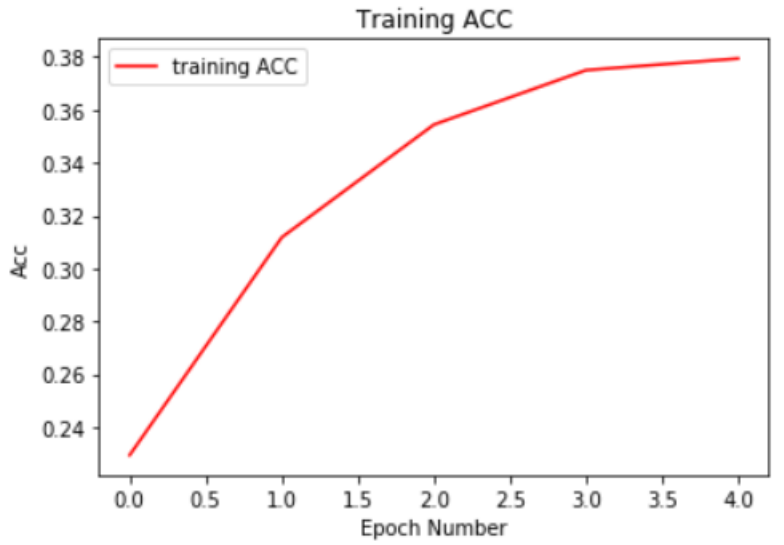


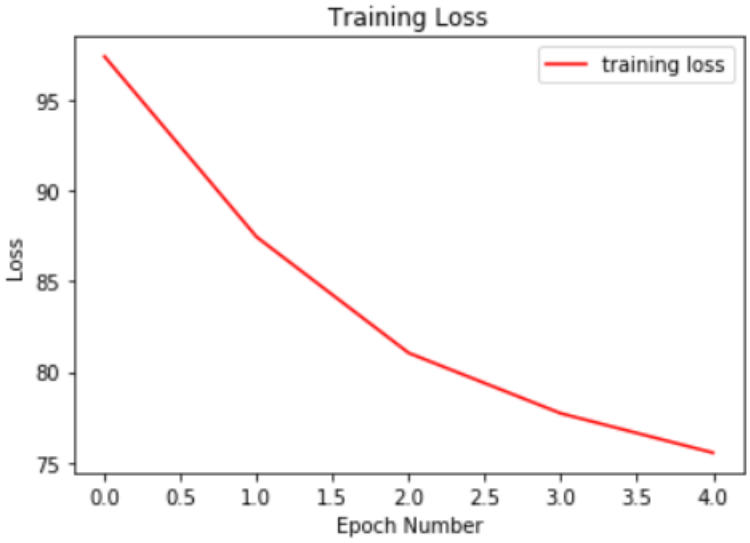
7.1.4 CNN on EMNIST





7.2.1 Custom Implementation





7.2.2 Fine tuning Squeeze Net