

Homework 5

Alex Wu, chichiaw@andrew.cmu.edu

1 Part I - Theory

1.1

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

$$\text{if } c = 0 \quad \text{softmax} = \frac{e^{x_i}}{\sum e^{x_j}} \quad \text{range} = [0, +\infty)$$

$$\text{if } c = -\max_i x_i \quad \text{softmax} = \frac{e^{x_i - \max_i x_i}}{\sum e^{x_j - \max_i x_i}} \quad \text{range} = [0, 1]$$

Therefore, the reason for using $c = -\max_i x_i$ is that it will bound the range.

1.2

- a1)* Range of each element $[0, 1]$
- a2)* Sum of all element $\sum \text{softmax}(x) = 1$
- b)* Probabilistic distribution
- c)* First step: Transfer to exponential values.
Second step: Take the sum of all elements.
Third step: Calculate the probability of each element.

1.3

$$f_1 = w_1x_1 + w_2x_2 + w_3x_3 = W_1^T X + b_1$$

$$f_2 = W_2^T f_1 + b_2 = W_2^T (W_1^T X + b_1) + b_2 = W_2^{T'} X + b_2'$$

$$f_3 = W_3^{T'} X + b_3'$$

\vdots

$$y = \theta X + b \Rightarrow \text{linear regression}$$

1.4

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

1.5

$$\begin{aligned}
 W &\in R^{k \times d}, & \therefore \frac{\partial J}{\partial W} &\in R^{k \times d} \\
 x &\in R^{d \times 1}, & \therefore \frac{\partial J}{\partial x} &\in R^{d \times 1} \\
 b &\in R^{k \times 1}, & \therefore \frac{\partial J}{\partial b} &\in R^{k \times 1}
 \end{aligned}$$

take partial in scalars

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}} = \delta_i x_i \Rightarrow \frac{\partial J}{\partial W} = \delta X^T$$

$$\begin{aligned}
 \frac{\partial J}{\partial x_i} &= \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial x_i} = \delta_i W_{ij} \Rightarrow \frac{\partial J}{\partial X} = W^T \delta \\
 \frac{\partial J}{\partial b_i} &= \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial b_i} = \delta_i \Rightarrow \frac{\partial J}{\partial b} = \delta
 \end{aligned}$$

1.6

$$1) \quad y = \sigma(h_n(\sigma(h_{n-1}(\dots(h_2(\sigma(h_1(x)))))))$$

$$\frac{\partial y}{\partial x} = \sigma'(h_n(\dots))h'_n(\dots)\sigma'(h_{n-1}(\dots))\dots\sigma'(h_1(x))h'_1(x)$$

We know $\sigma'(x) \in [0, 1]$, so many $\sigma'(x)$ time with each other would result in a very small value causing the issue of "vanishing gradient".

- 2) As one could see in figure 1, the output range for sigmoid is $[0, 1]$ and $[-1, 1]$ for tanh function. "tanh" function is symmetric which will be easier to converge to a symmetric bias, and it has a stronger gradient around zero.
- 3) As one can see in figure 1, in most area, gradient of tanh is larger than gradient of sigmoid. Therefore, tanh has less of a vanishing gradient problem.

$$4) \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - \frac{1 + e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1$$

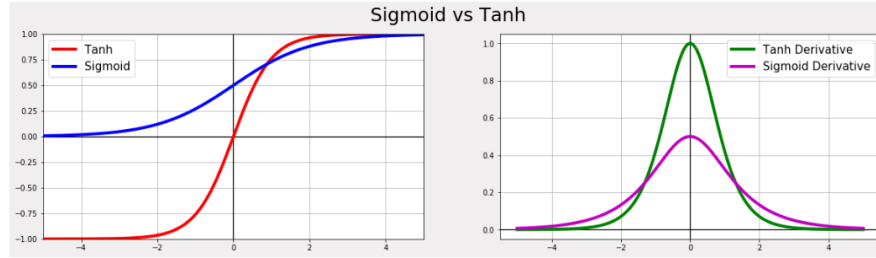


Figure 1: Eight-point algorithm result

2 Part II - Implement a Fully Connected Network

2.1 Network Initialization

2.1.1

With all parameters = 0, then no matter what the inputs are, the outputs are the same. Also when calculating backprops for gradients, all outputs result in the same gradient.

2.1.3

Math: Neural network \Rightarrow a chain of functions applied on top of each other. There are two problems that we want to take care of. First, we do not want any data loss, and secondly, we want to take into account all inputs. Therefore, we would keep the parameters small but not too small to prevent numerical instability. Also, initializing with random numbers increases the chance to be close to optimal.

Scaling the initialization depending on the layer size can maintain the variances of the gradients on the weights from different layers the same. Therefore, it can prevent calculations from vanishing or exploding with deeper network.

3 PartII - Training Models

3.2

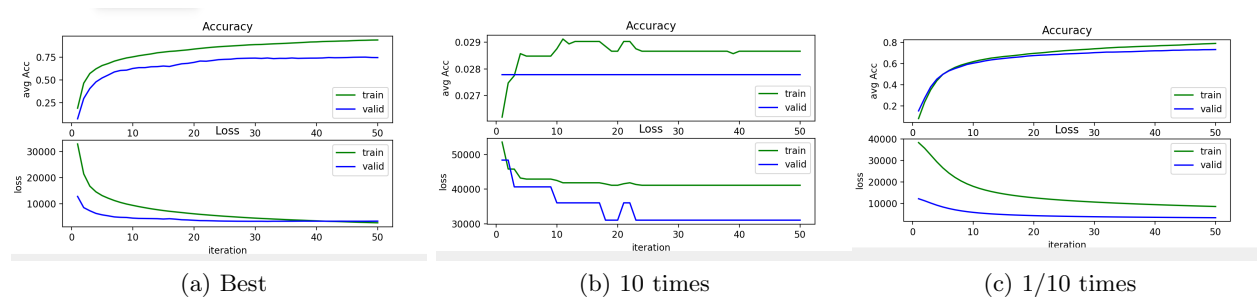


Figure 2: Accuracy and loss with different learning rate

Learning rate is one of the key parameters in learning. A training can be thought of as a ball starting at random location along a convex curve. The goal will then be to let the ball reach the lowest point of the curve (global minimum). Learning rate will effect how big an update step is toward the gradient at each epoch. If the learning rate increases, then the loss will decrease and the accuracy will increase drastically in the early age of training. However, the training may not be able to reach the goal. It is because if the distance to reach optimal is smaller than the update step, then the training will keep oscillating back and forward around the goal, and simply cannot get closer to it. On the other hands, if the learning rate decreases, there will be two effects on the training. First, the training will reach the goal slower, since the update step is smaller at each epoch. Secondly, if there is a additional local minimum in the curve, then the result may be trapped in that region, and miss the global minimum.

Accuracy: Train: 0.93; Valid: 0.74472; Test: 0.7611

3.3

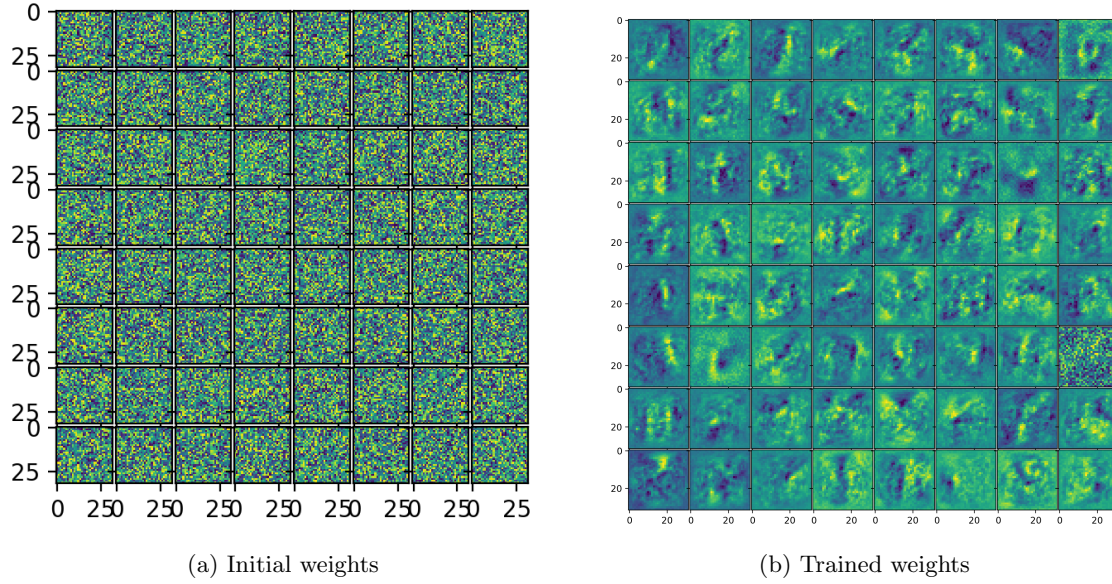
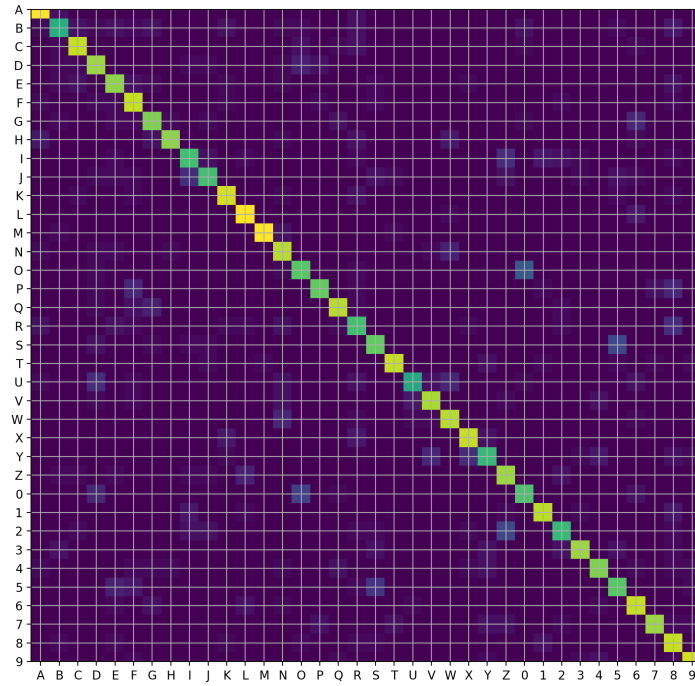


Figure 3: First layer weights after and before training.

Comparing the two images in fig 3, one could definitely tell there are some patterns in the weights after training. However, by simply looking at the patterns in figure 3(b), it is hard to make more explanation.

3.4



(a) Initial weights

Figure 4: Confusion matrix.

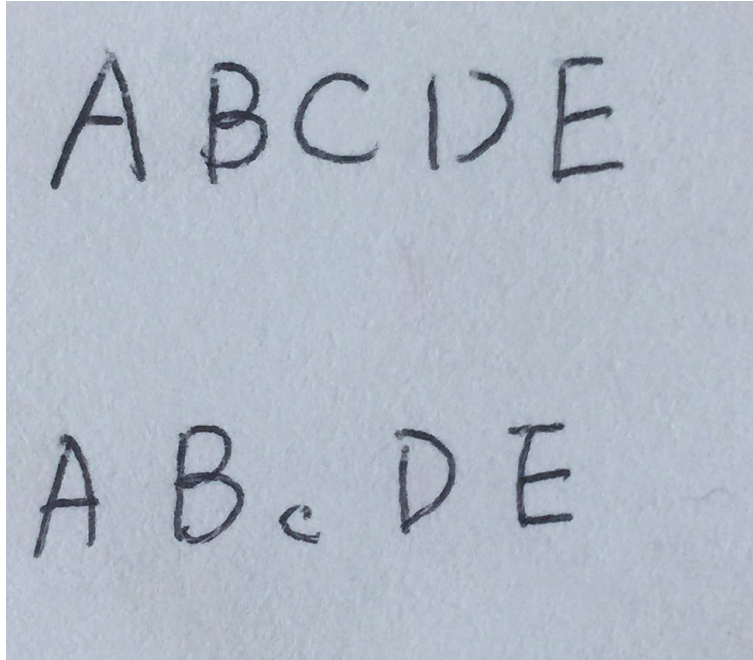
The most commonly confused pair in the model is "O"(alphabet) and "0"(number). They are basically the same in most cases. Other examples are 1) "Z"(alphabet) and "2"(number) 2) "S"(alphabet) and "5"(number). The characters in both cases have similar shapes, and the only difference is that one is with sharp edges and the other is with round curves.

4 Extract Text from Images

4.1 Theory

Assumptions:

- 1) All the strokes within a character are connected.
- 2) All the characters are similar in their sizes.



(a) Detection fail example

Figure 5: Detection fail example.

In figure 5 the strokes in "D" is separated, and it is possible to be seen as two characters by training model. The size of "C" is smaller than other characters, and it is likely to be seen as noise in the image.

4.3

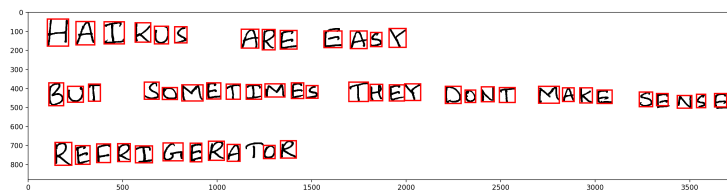


Figure 6: Results of extracting characters from images.

4.4

D	E	E	P	L	E	A	R	N	I	N	G			
G	K	E	P	L	K	A	R	N	I	N	2			
D	E	E	P	E	R	L	E	A	R	N	I	N	G	
D	E	E	9	E	R	L	E	A	R	N	2	N	G	
D	E	E	P	E	S	T	L	E	A	R	N	I	N	G
U	E	E	R	E	S	T	L	E	A	R	N	I	N	G

Accuracy: 0.80

T	O	D	O	L	I	S	T														
M	0	D	0	L	I	I	T														
1	M	A	K	E	A	T	O	D	O	L	I	S	T								
1	4	A	K	E	A	7	D	P	0	G	2	L	T								
2	C	H	E	C	K	O	F	F	T	H	E	F	I	R	S	T					
2	C	H	S	I	K	0	F	E	7	N	R	F	I	R	5	T					
T	H	I	N	G	O	N	T	O	D	O	L	I	S	T							
7	H	I	N	G	0	N	T	O	P	O	L	X	Q	T							
3	R	E	A	L	I	Z	E	Y	O	U	H	A	V	E	A	L	R	E	A	D	Y
3	R	I	A	L	I	2	E	Y	0	U	B	W	V	E	A	L	R	E	A	R	Y
C	O	M	P	L	E	T	E	D	2	T	H	I	N	G	S						
C	O	M	B	V	E	T	I	D	2	Y	4	I	N	4	G						
4	R	E	W	A	R	D	Y	O	U	R	S	E	L	F	W	I	T	H			
4	R	E	W	A	R	D	Y	O	4	R	5	E	L	F	W	I	M	R			
A	N	A	P																		
A	N	A	P																		

Accuracy: 0.65

H	A	I	K	U	S	A	R	E	E	A	S	Y											
H	A	Z	K	U	S	A	R	E	B	A	G	Y											
B	U	T	S	O	M	E	T	I	M	E	S	T	H	E	Y	D	O	N	T	M	A	K	E
S	E	N	S	E																			
D	U	T	S	O	M	E	T	I	M	E	G	T	R	E	X	D	0	N	T	M	A	K	R
S	6	N	S	G																			
R	E	F	R	I	G	E	R	A	T	O	R												
R	8	F	R	I	G	8	R	A	T	D	R												

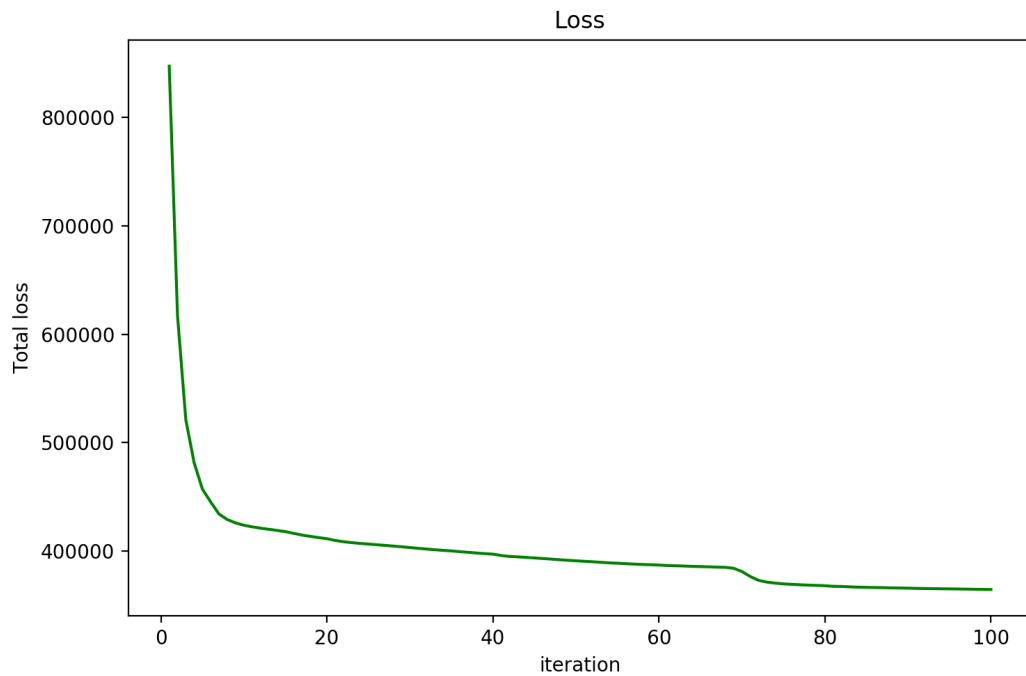
Accuracy: 0.74

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

Accuracy: 0.81

5 Building the Autoencoder

5.2



(a) Total loss

Figure 7: Training error for the autoencoder.

The loss of the training dropped drastically in the first few epochs. Then the drop became mild after around the 10th epochs. The curve finally converged at the very end of the training.

5.3

5.3.1

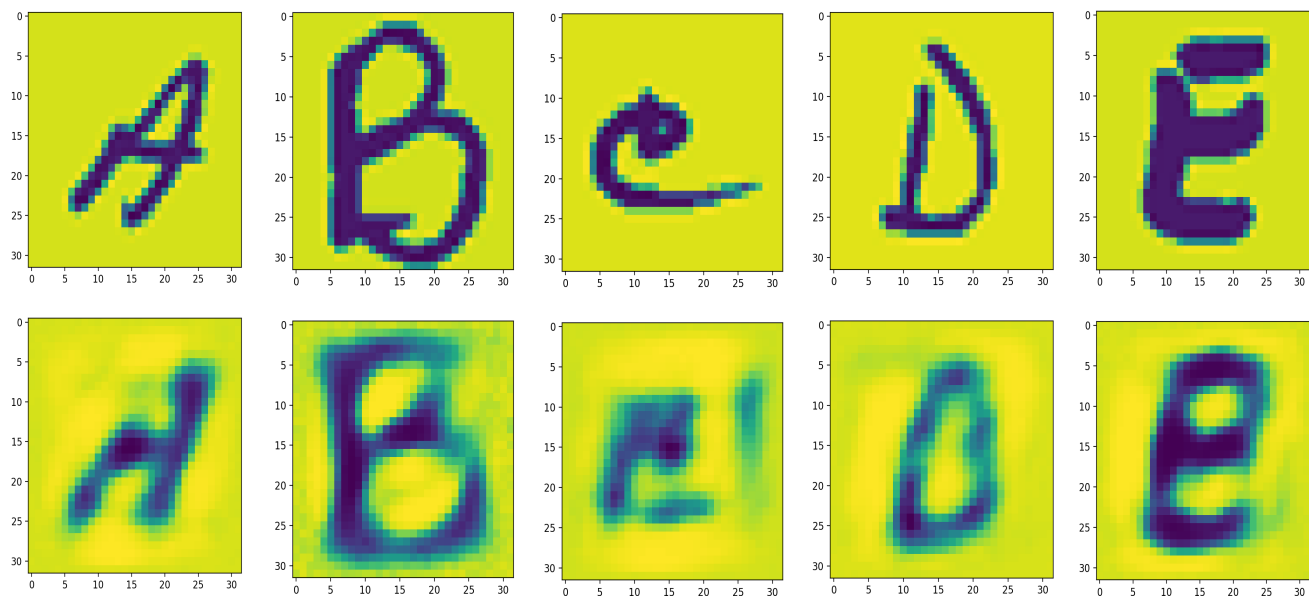
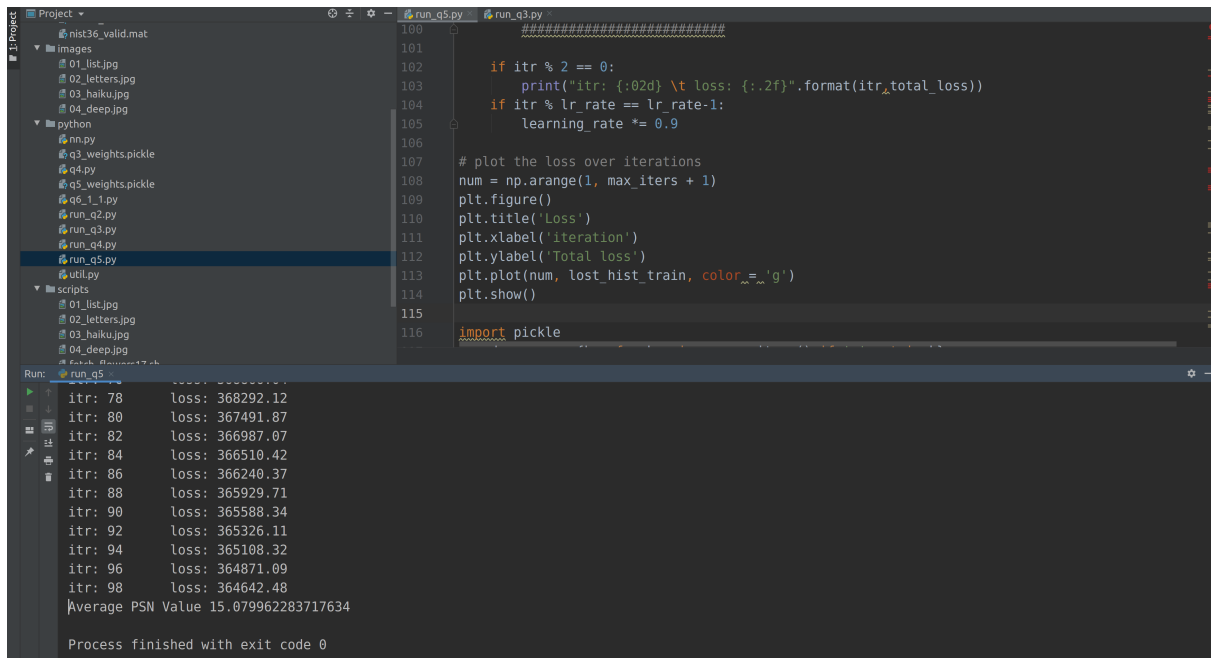


Figure 8: Comparison between the original input images(above) and reconstructed validation images(below).

It is clear that the reconstructed images have far less resolution compared to the original input images. It is reasonable, since we forced the model to learn represent the data with a limited number of hidden nodes. There must be some features that were left out. It might cause some confusion in recognizing the inputs. For example, in figure 8 the reconstructed "B" and "D" are similar in their contours.

5.3.2



The screenshot shows a Jupyter Notebook environment. The left sidebar displays a project directory with files like `nist36_valid.mat`, `images` (containing `01_list.jpg`, `02_letters.jpg`, `03_haiku.jpg`, `04_deep.jpg`), `python` (containing `run.py`, `q3_weights.pickle`, `q4.py`, `q5_weights.pickle`, `q6_1_1.py`, `run_q2.py`, `run_q3.py`, `run_q4.py`, `run_q5.py`), `utils.py`, and `scripts` (containing `01_list.jpg`, `02_letters.jpg`, `03_haiku.jpg`, `04_deep.jpg`). The main area shows the code for `run_q5.py`, which includes a loop for training iterations, a plot of loss over iterations, and an import for `pickle`. The bottom output area shows the results of the training process.

```
#####
100
101
102
103     if itr % 2 == 0:
104         print("itr: {:02d} \t loss: {:.2f}".format(itr,total_loss))
105     if itr % lr_rate == lr_rate-1:
106         learning_rate *= 0.9
107
108 # plot the loss over iterations
109 num = np.arange(1, max_iters + 1)
110 plt.figure()
111 plt.title('Loss')
112 plt.xlabel('iteration')
113 plt.ylabel('Total loss')
114 plt.plot(num, lost_hist_train, color_='g')
115 plt.show()
116 import pickle
```

Run: `run_q5.py`

itr	loss
78	368292.12
80	367491.87
82	366987.07
84	366510.42
86	366240.37
88	365929.71
90	365588.34
92	365326.11
94	365108.32
96	364871.09
98	364642.48

Average PSN Value 15.079962283717634

Process finished with exit code 0

Figure 9: PSN Value.

The average PSNR I got from the autoencoder across all images in the validation set was 15.08.

6 Pytorch

6.1 Train a neural network with Pytorch

6.1.1

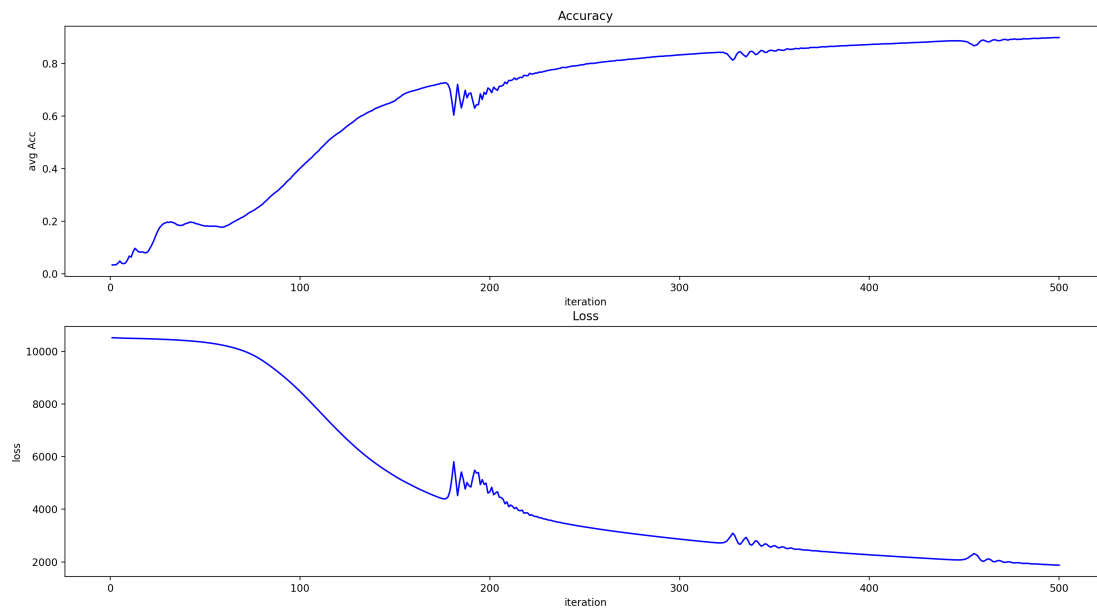


Figure 10: Fully-connected network training error and average accuracy

6.1.2

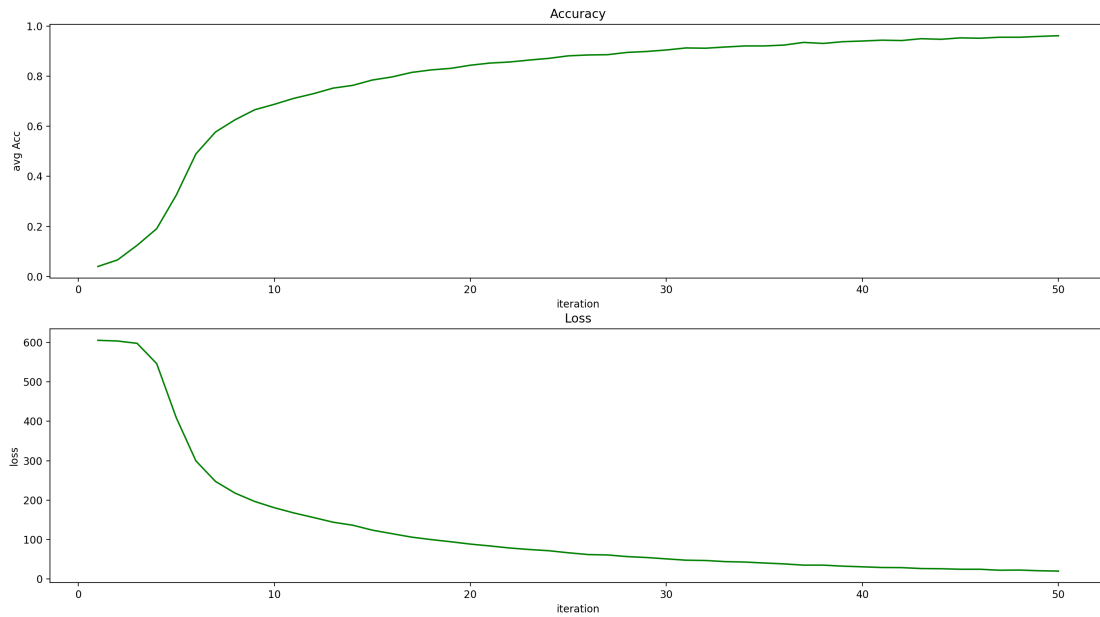


Figure 11: Convolutional neural network with Pytorch

At the beginning of the convolutional neural network, the total loss dropped and the average accuracy rose faster compared to the fully-connected one. As you can see in figure 11 the accuracy reached 0.9 within 50 epochs. However, the process time for each epoch was longer than that for fully-connected neural network.

6.1.3

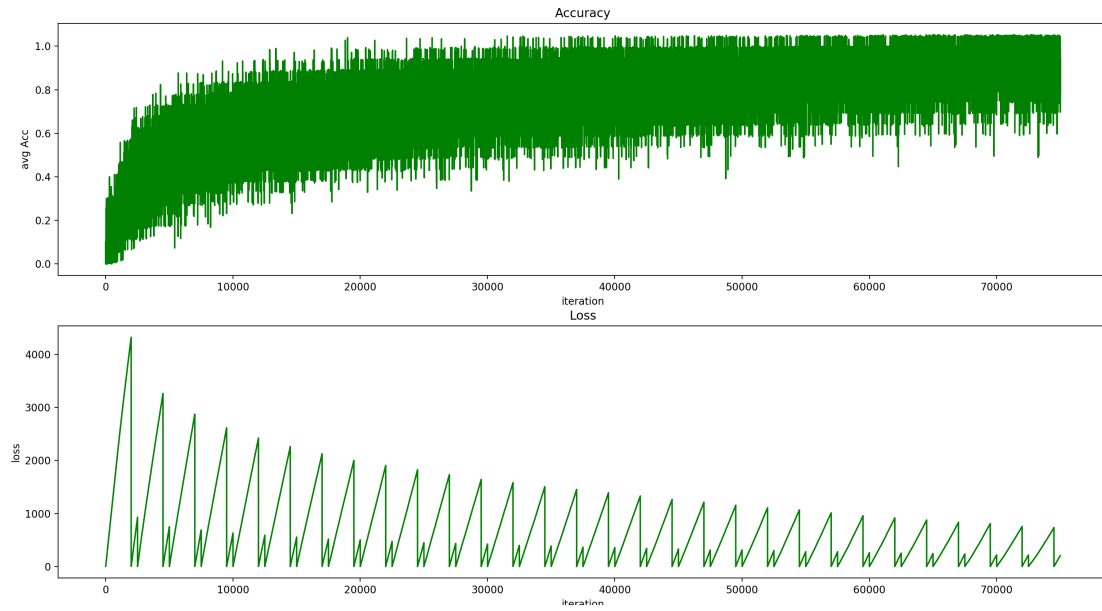


Figure 12: Convolutional neural network with Pytorch on CIFAR-10 data

6.1.4

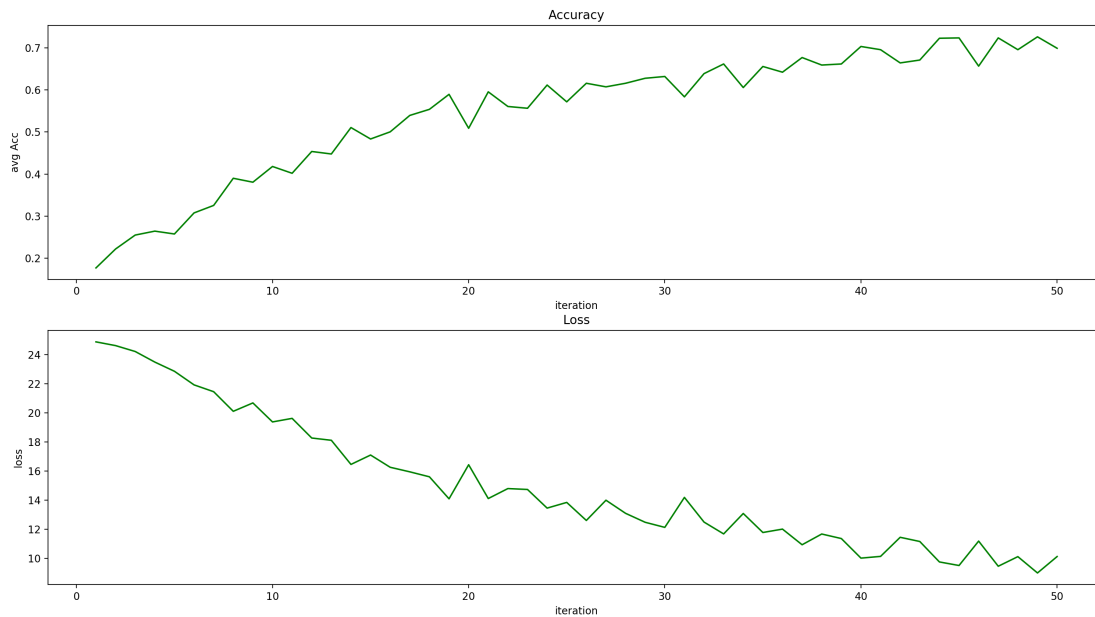


Figure 13: Convolutional neural network with Pytorch on SUN data

```
140 avg_acc = 0
141 for test in test_batch:
142     xb = test[0]
143     yb = test[1]
144
145     y_pred = model(xb)
146
147     predicted = torch.argmax(y_pred, 1)
148     avg_acc += torch.sum(predicted == yb).item()
149
150 avg_acc = avg_acc / test_y.shape[0]
151
152 print('Test accuracy: {}'.format(avg_acc))
153
154
```

```
Run: q6_1.4.py
Test accuracy: 0.625
Process finished with exit code 0
```

Figure 14: Test accuracy

Test Accuracy : 0.625 Using convolutional neural network is much more easier in implementation, faster in running time, and higher accuracy. In HW1, the accuracy reached 0.605 after fine tune, and here I got 0.625 almost with some random set of parameters. The whole process time including loading image data is under 2 minutes, where in HW1 it took almost 5-10 minutes to run the whole process.

6.2 Fine Tuning

6.2.1

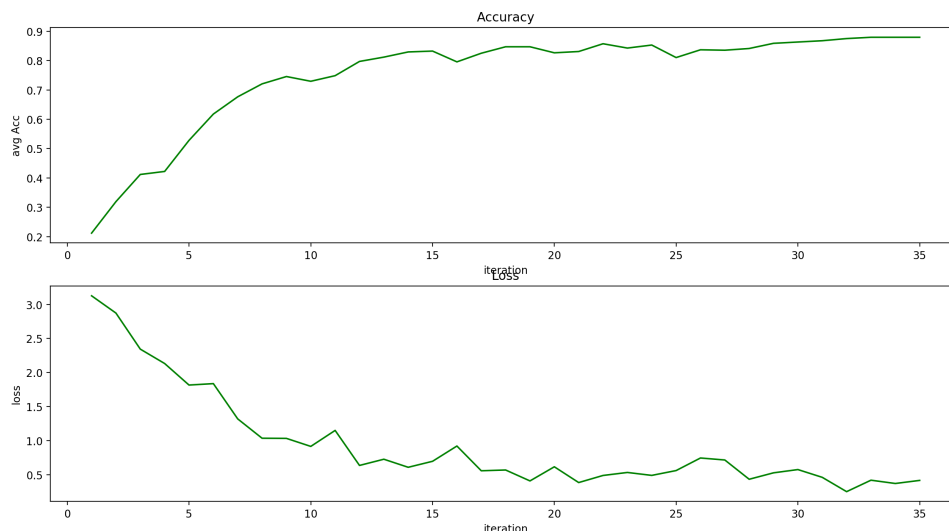


Figure 15: Training with Squeezenet

Final Train Accuracy : 0.879 / Val Accuracy: 0.844

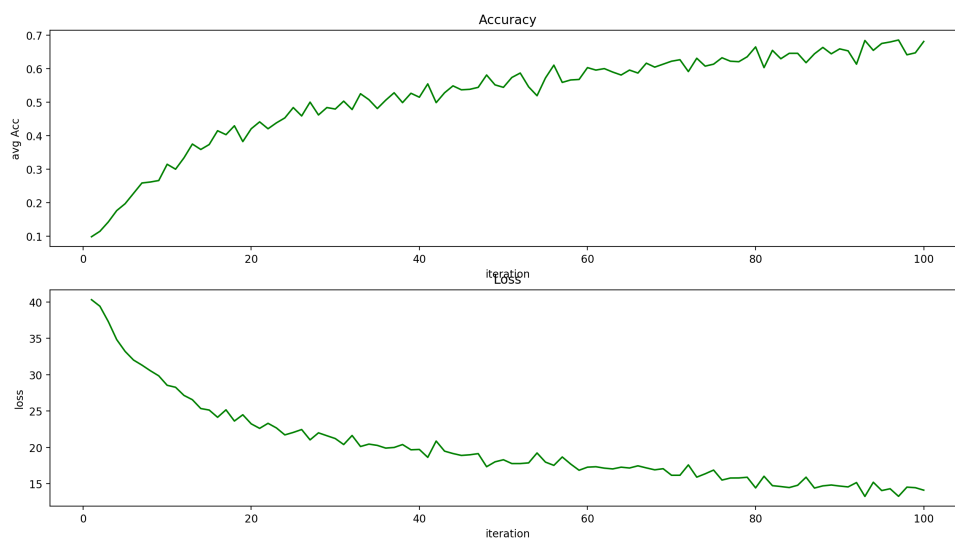


Figure 16: Training with scratch model

Final Train Accuracy : 0.69

Using Squeezenet to train the data set gives about a better result in terms of accuracy. With some tuning, the training accuracy could reach 88 % by using Squeezenet. However, it is almost impossible for the scratch model to reach an accuracy of 70%. Also the time it takes to run the training is much faster with Squeezenet.