16-720B Computer Vision: Homework 5

Neural Networks for Recognition

Due: November 26 at 11:59pm

[redacted] (AndrewID: [redacted])

1. Theory

Q1.1

For each index i in a vector $\mathbf{x}$, there is

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

For each index i in a vector $\mathbf{x}$+c, there is

$$\begin{aligned} softmax(x_i + c) &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^{x_i}e^c}{e^c\sum_j e^{x_j}} \\ &= \frac{e^{x_i}}{\sum_j e^{x_j}} \end{aligned}$$

So we have

$$softmax(x) = softmax(x+c)$$

Softmax is invariant to translation.

With $c = 0$, the values of the enumerator would be in range $(0, +\infty)$. With $c = -\max x_i$, the values of the enumerator would be in range $(0, 1]$. So with the translation, it can help to avoid overflow during the calculation and meanwhile obtain the same result.

Q1.2

• The range of each element is $(0, 1]$; The sum over all elements is 1.

• Softmax takes an arbitrary real valued vector x and turns it into a probability distribution with same number of elements in x.

• The first step is to calculate the exponential value for each element in x; The second step is to calculate the total value of all elements; The third step is to calculate the ratio/probability for each element in x.

Q1.3

Assume that 2 successive linear regression layers. The input is $\mathbf{x} \in \mathbb{R}^{N\times 1}$. The first layer has parameters $\mathbf{w}_1^T \in \mathbb{R}^{D\times N}$ and $\mathbf{b}_1 \in \mathbb{R}^{D\times 1}$. The second layer has parameters $\mathbf{w}_2^T \in \mathbb{R}^{M\times D}$ and $\mathbf{b}_2 \in \mathbb{R}^{M\times 1}$. So the outputs of the two layers can be written as:

$$\begin{aligned}\mathbf{h}_1 &= \mathbf{w}_1^T\mathbf{x} + \mathbf{b}_1\\ \mathbf{h}_2 &= \mathbf{w}_2^T\mathbf{h}_1 + \mathbf{b}_2\end{aligned}$$

So we have

$$\begin{aligned}\mathbf{h}_2 &= \mathbf{w}_2^T(\mathbf{w}_1^T\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\\ &= \mathbf{w}_2^T\mathbf{w}_1^T\mathbf{x} + (\mathbf{w}_2^T\mathbf{b}_1 + \mathbf{b}_2)\\ &= \mathbf{w'}_2^T\mathbf{x} + \mathbf{b'}_2\end{aligned}$$

where $\mathbf{w'}_2 = \mathbf{w}_2^T\mathbf{w}_1^T \in \mathbb{R}^{M\times N}$, $\mathbf{b'}_2 = \mathbf{w}_2^T\mathbf{b}_1 + \mathbf{b}_2 \in \mathbb{R}^{M\times 1}$

Similarly, the outputs of the following linear regression layers can be written as

$$\begin{aligned}\mathbf{h}_3 &= \mathbf{w}_3^T(\mathbf{w'}_2^T\mathbf{x} + \mathbf{b'}_2) + \mathbf{b}_3\\ &= \mathbf{w}_3^T\mathbf{w'}_2^T\mathbf{x} + (\mathbf{w}_3^T\mathbf{b'}_2 + \mathbf{b}_3)\\ &= \mathbf{w'}_3^T\mathbf{x} + \mathbf{b'}_3\end{aligned}$$

$$\begin{aligned}\mathbf{h}_4 &= \mathbf{w}_4^T(\mathbf{w'}_3^T\mathbf{x} + \mathbf{b'}_3) + \mathbf{b}_4\\ &= \mathbf{w}_4^T\mathbf{w'}_3^T\mathbf{x} + (\mathbf{w}_4^T\mathbf{b'}_3 + \mathbf{b}_4)\\ &= \mathbf{w'}_4^T\mathbf{x} + \mathbf{b'}_4\end{aligned}$$

$$\vdots$$

So it is equivalent to linear regression. This implies that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Q1.4

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

Q1.5

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

So we have

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

So, there is

$$\frac{\partial y}{\partial W} = \begin{bmatrix} x_1 & \cdots & x_1 \\ x_2 & \cdots & x_2 \\ \vdots & \ddots & \vdots \\ x_d & \cdots & x_d \end{bmatrix} = \begin{bmatrix} x & \cdots & x \end{bmatrix} \in \mathbb{R}^{d\times k}$$

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

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

By using the chain rule, we get

$$\frac{\partial J}{\partial W} = \sum_{j=1}^{k} \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial W} = x\delta^T \in \mathbb{R}^{d\times k}$$

$$\frac{\partial J}{\partial x} = \sum_{j=1}^{k} \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial x} = W\delta \in \mathbb{R}^{d\times 1}$$

$$\frac{\partial J}{\partial b} = \sum_{j=1}^{k} \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial b} = \delta \in \mathbb{R}^{k\times 1}$$

Q1.6

(a) Let's denote that the linear layers of the network written as $h_1(x)$, $h_2(x)$, ... When using sigmoid function as the activation function. The output of the last layer would be

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

Then when we calculate the gradients during the backpropogation, the gradient would be

$$\frac{dy}{dx} = \sigma'(h_n(\sigma(h_{n-1}(\cdots))))h_n'(\sigma(h_{n-1}(\cdots)))\sigma'(h_{n-1}(\cdots))\cdots\sigma'(h_1(x))h_1'(x)$$

And we know that

$$\sigma'(x) = \sigma(x)(1-\sigma(x)) \in (0,1)$$

So with deep neural networks, with many sigmoid activation layers, the gradient tends to be zero when performing the backpropogation by multiplying values less than 1.

(b)

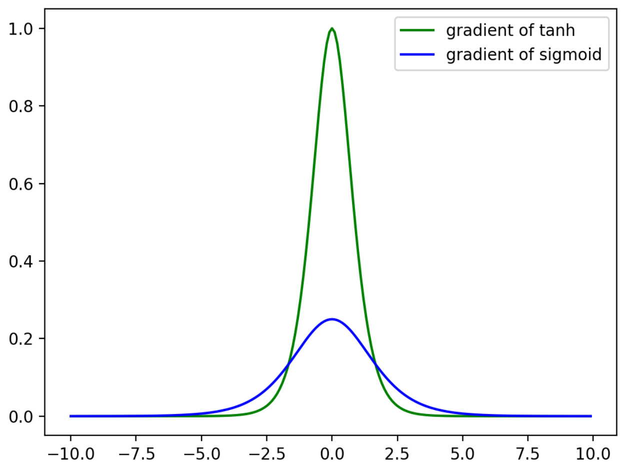
$$\begin{aligned}
\tanh(x) &= \frac{1-e^{-2x}}{1+e^{-2x}} \\
&= 1 - \frac{2e^{-2x}}{1+e^{-2x}} \\
&= 1 - \frac{2}{1+e^{2x}} \in (-1, 1)
\end{aligned}$$

So the output range of $\tanh(x)$ is $(-1, 1)$, the output range of $\sigma(x)$ is $(0, 1)$
So the sigmoid is non-symmetric, which would introduce a systematic bias for the neurons. The tanh is symmetric and it is more likely that its mean value to be zero, which would have a faster convergence. On the other hand, we have that

$$\begin{aligned}
\frac{\partial \tanh x}{\partial x} &= \frac{4e^{2x}}{(1+e^{2x})^2} \\
&= 1 - \frac{(1-e^{-2x})^2}{(1+e^{-2x})^2} \\
&= 1 - (\tanh x)^2
\end{aligned}$$

Assuming that the data has been normalized, then the data would be centered around 0. The gradient of tanh is larger than that of sigmoid, which makes that it would be easier to train the model with tanh.

(c) As mentioned above in the previous question, the gradient of tanh around zero is larger than that of sigmoid. The derivatives of these two functions are shown as below:



Thus, with larger gradient, the tanh has less of a vanishing gradient problem.

(d)

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

2. Implement a Fully Connected Network

2.1. Network Initialization

Q2.1.1

If a network is initialized with all zeros, it means that all the parameters are set to zero at initial. Then during the forward propogation, each hidden node would just get the zero signal and generate the same output, no matter what the input is. When performing the backpropogation, the same output would cause the same gradients and thus would do the same updates to the parameters. This is not what we want during the training. So it is not a good idea to initialize all the weights to zero.

Q2.1.2 Please refer to the code.

Q2.1.3

The random initialization increase the chance to reach to the optimal parameters that can lead to low loss. It helps to find an optimal set of weights for the mapping functions from the input data to the output. It also meets the requirement of stochastic gradient descent method. This can help to break the network symmetry and makes it learn faster.

By scaling the initialization depending on layer size, it makes the variance of the gradient on the weights is the same for all layers. In such a way, the signals both during the forward propogation and the backpropogation won't expand to a very large value or diminish to a very small one, thus can avoid the gradient explosion and vanishing problems.

2.2 Forward Propagation

Q2.2.1 Please refer to the code.

Q2.2.2 Please refer to the code.

Q2.2.3 Please refer to the code.

2.3 Backwards Propagation

Q2.3.1 Please refer to the code.

2.4 Training Loop

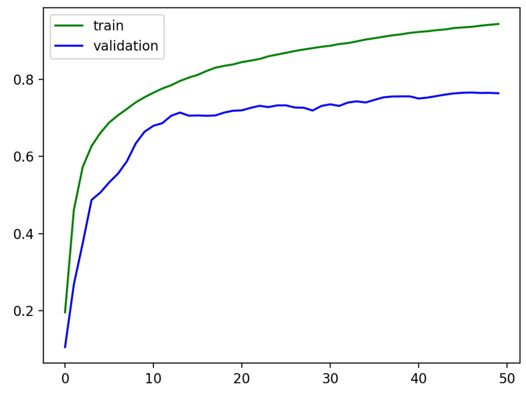
Q2.4.1 Please refer to the code.

2.5 Numerical Gradient Checker

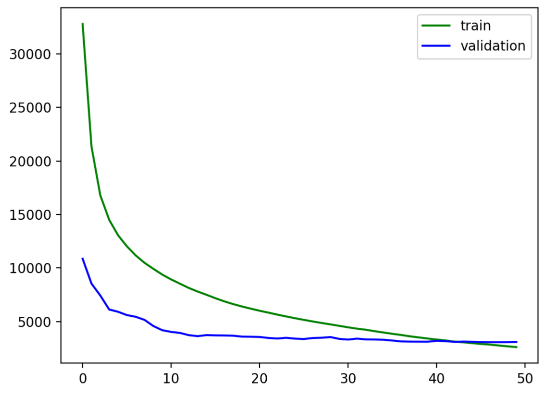
Q2.5.1 Please refer to the code in `run_q2.py`.

3. Training Models

Q3.1.1 The plotted accuracy and loss on both the training and validation sets are shown in Fig.1. The learning rate here is set to 0.01. The validation accuracy is 76.39%.



(a) accuracy



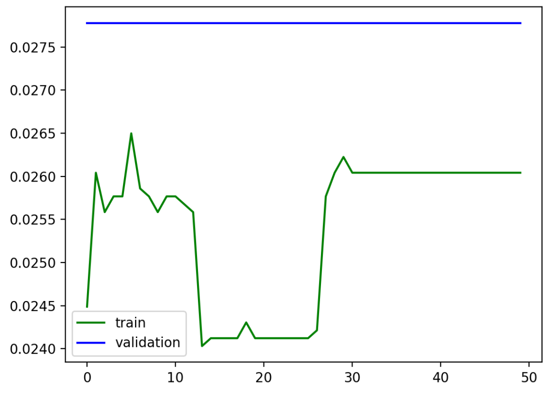
(b) cross-entropy loss

Figure 1: The accuracy and cross-entropy loss on training and validation sets.

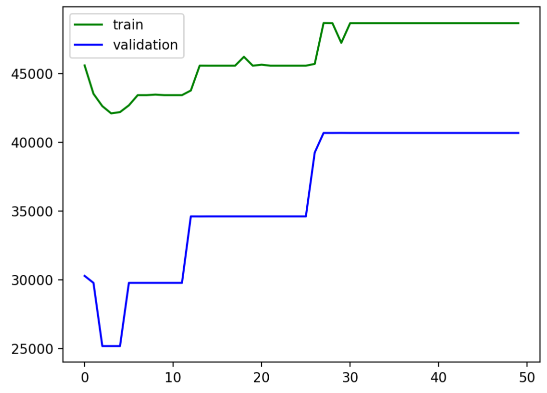
Q3.1.2 The accuracy and loss with the best learning rate (learning rate = 0.01) are shown as above in **Q3.1.1** in Fig.1. The accuracy and loss with 10 times of it (learning rate = 0.1) and with one tenth of it (learning rate = 0.001) are shown as Fig.2 and Fig.3 respectively.

We can see from the images that with proper learning rate, the model can gradually converge to the local minimum to learn the appropriate parameters and has low loss and high accuracy. With large learning rate, it is possible that the model would be away from the local minimum because of too much "energy" to search it, the parameters might just bouncing around and might never reach to a local minimum. With low learning rate, the model would converge to a local minimum gradually, but it converges lower than that with a good learning rate.

The final accuracy of the best network on the validation set is 76.39%, and the final accuracy on the test set is 76.06%.

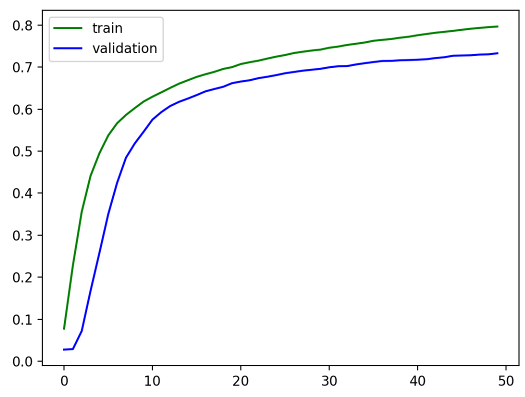


(a) accuracy

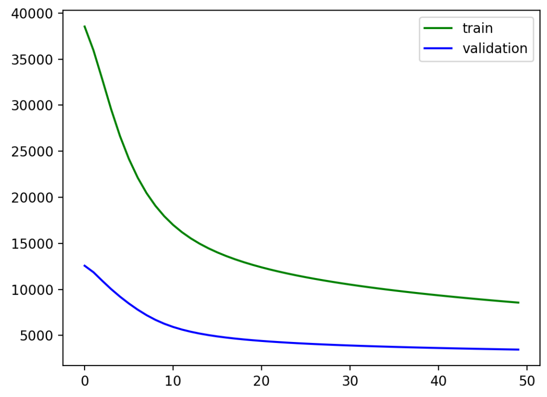


(b) cross-entropy loss

Figure 2: The accuracy and cross-entropy loss with learning rate 0.1.



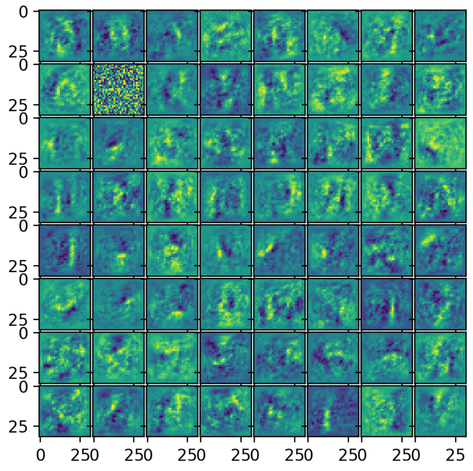
(a) accuracy



(b) cross-entropy loss

Figure 3: The accuracy and cross-entropy loss with learning rate 0.001.

Q3.1.3 The visualizations of the first layer weights that the network learned and the network weights immediately after initialization are shown as Fig.4. With the learned weights, we can clearly see some patterns while the weights just after initialization just show random patterns.


(a) weights after learning

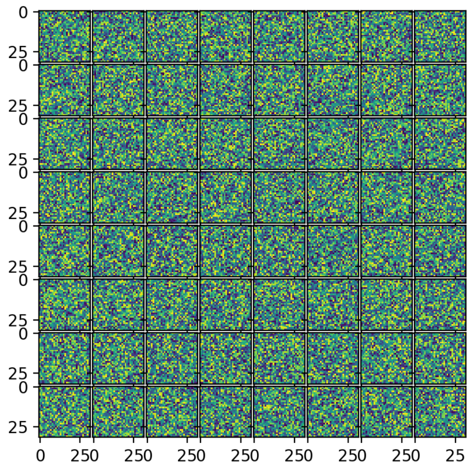
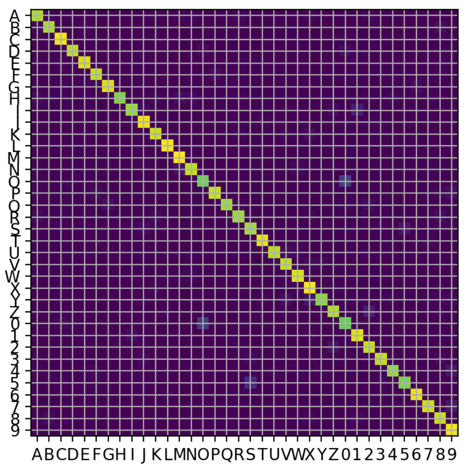

(b) weights of initialization

Figure 4: The visualization of weights after learning and weights of initialization.

Q3.1.4 The confusion matrix for the best model on train and test set are shown as Fig.5.

The top pairs of classes that are most commonly confused include pair {0, O}, {5, S}, {2, Z}, {1, I}. They all look similar to each other in appearance, especially in hand-written. It would be harder to separate them compared to other letters and alphabets.


(a) confusion matrix on training set

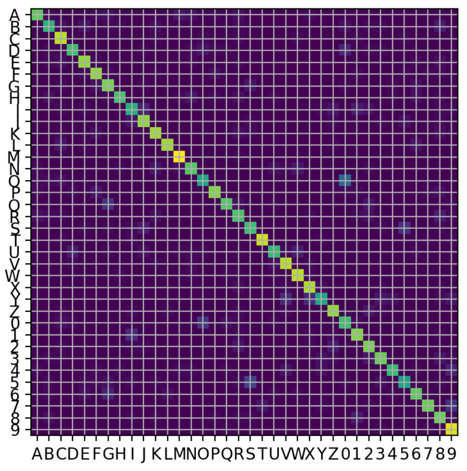

(b) confusion matrix on test set

Figure 5: The confusion matrix for the best model.

4. Extract Text from Images

Q4.1 The assumptions are as below:

(1) Each character forms a connected component; Every two characters are isolated from each other.

(2) The characters are in similar size.

Fig.6 shows some examples that might fail in detection. In the first image, it is possible that the detection might fail in detecting the letters ‘T’ and ‘R’ since each of them do not form a connected component; In the second image, it is possible that the detection might fail in detecting all the letters since ‘A’ is connected with ‘N’ and ‘X’ is connected with ‘V’, each connected component would be detected as one letter; In the third image, the letter ‘E’ is much smaller than other letters and it might be filtered during the processing.

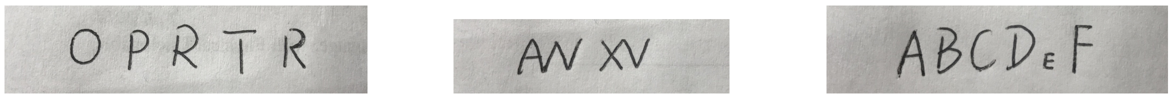


Figure 6: Examples of cases that might fail in detection.

Q4.2 Please refer to the code.

Q4.3 The result images are shown in Fig.7.

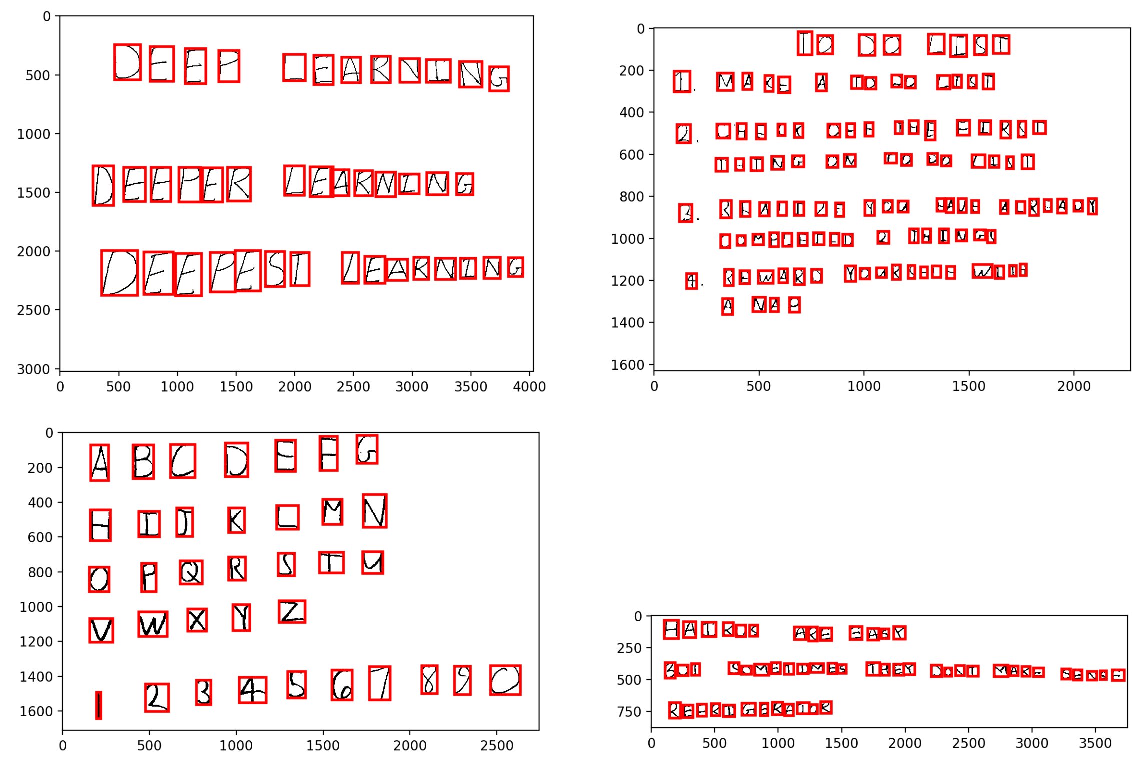


Figure 7: Result images of finding letters.

Q4.4 Please refet to the code in `run_q4.py`.

The extracted texts for each image in `images/` are shown as below. In the table, the characters that are in black are groundtruth and those in blue are the extracted texts. The recognition accuracy for each image is shown in the table. The overall accuracy is 80.08%.

01_list.jpg	02_letters,jpg
TO DO LIST TO DQ LIST 1 MAKE A TO DO LIST I N3KE A TD DS LIST 2 CHECK OFF THE FIRST 2 LHFCK DFF 7HE FIRST THING ON TO DO LIST THING QN TO DO LIST 3 REALIZE YOU HAVE ALREADY Z RIALIZE YQU HAVE ALR6ADT COMPLETED 2 THINGS C0MPLFTID I THINGS 4 REWARD YOURSELF WITH 9 RFWARD YOURSFLF WITB A NAP A NAP	ABCDEFG ABCDEFG HIJKLMN HIIKLMN OPQRSTU QPQRSTU VWXYZ VWXYZ 1234567890 1Z3GS6789S
Accuracy: 79.13%	**Accuracy: 83.33%**
03_haiku.jpg	
HAIKUS ARE EASY HAIKWG ARG GAGY BUT SOMETIMES THEY DONT MAKE SENSE BWT SDMETIMES THEX DDWT MAKG SGNGE REFRIGERATOR RBGRIGBRATQR	
Accuracy: 68.52%	
04_deep.jpg	
DEEP LEARNING DEEP LEARMING DEEPER LEARNING DEEPER LEARQING DEEPEST LEARNING DEEPEST LEARNING	
Accuracy: 95.12%	

5. Image Compression with Autoencoders

5.1 Building the Autoencoder

Q5.1.1 Please refer to the code.

Q5.1.2 Please refer to the code.

5.2 Training the Autoencoder

Q5.2 The plotted training loss curve is shown as Fig.8.

The training loss gradually decreases with more training iterations. At the beginning, the loss decreases dramatically, then it decreases slower and finally it tends to converge.

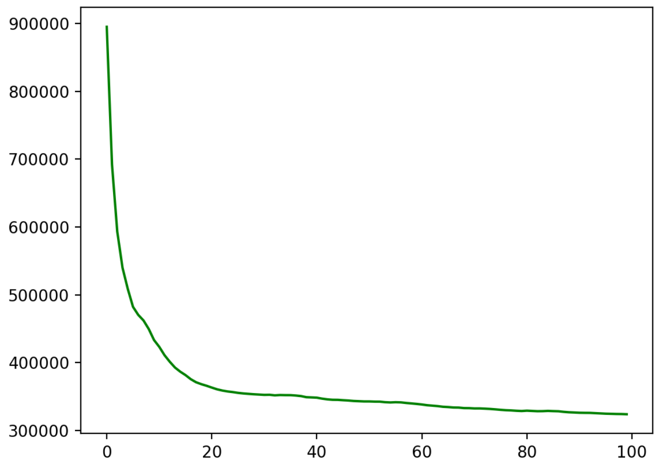


Figure 8: The training loss curve of autoencoder.

5.3 Evaluating the Autoencoder

Q5.3.1 Fig.9 shows some examples of the reconstruction results.

The reconstructed images are blurrer than the original images. There are some noises around the characters.

Q5.3.2 The average PSNR obtained from the autoencoder across all validation images is **15.6456**

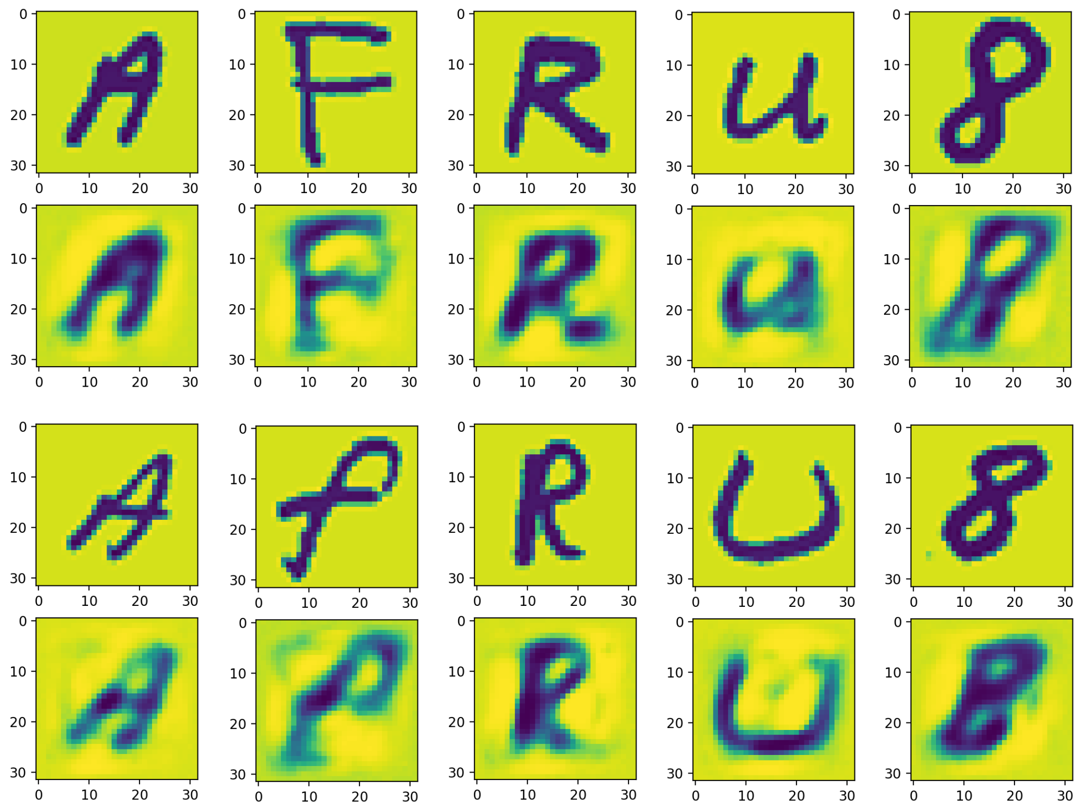


Figure 9: Examples of the reconstruction results.

6. **Comparing against PCA**

Q6.1 The size of the projection matrix is 32×1024. The rank is 32.

Q6.2 The examples of reconstructed images are shown as Fig.10.

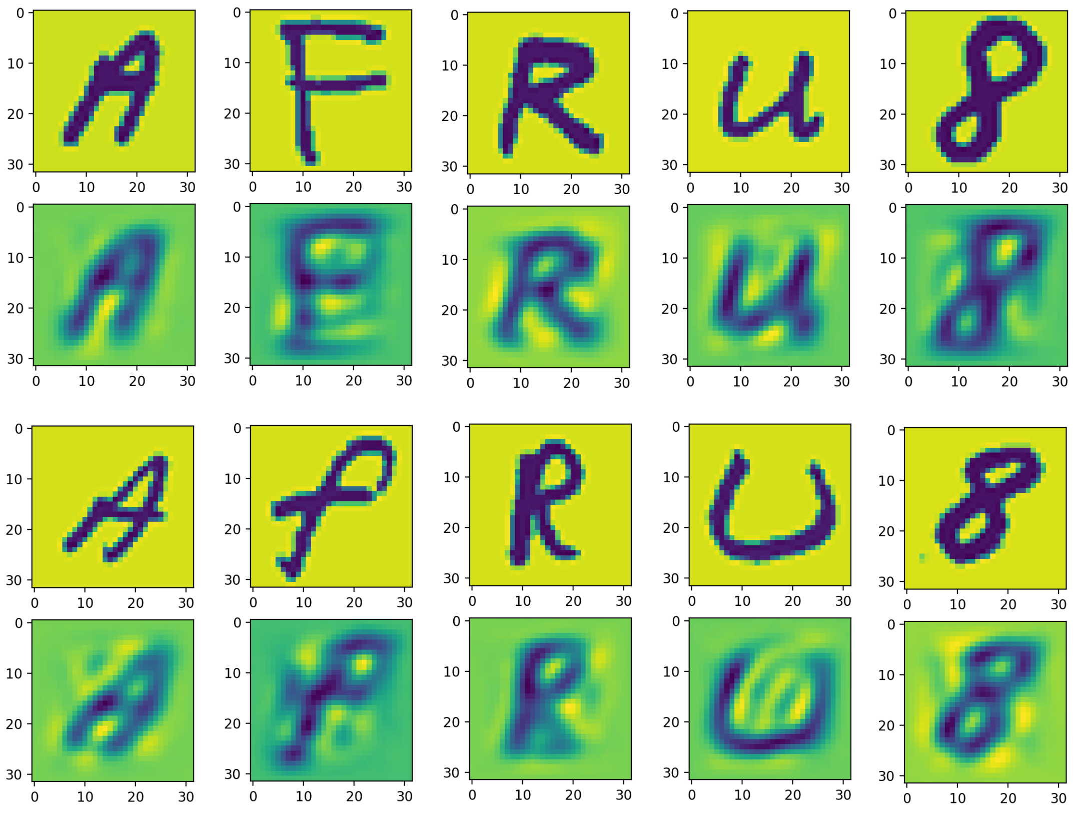


Figure 10: Examples of the reconstruction results using PCA.

There are many noises in the reconstructed images around the background compared to the original images. The appearances of the characters are roughly the same as the original one. The reconstruction result from autoencoder looks better than the result from PCA with less noises in visualization.

Q6.3 The average PSNR on validation set obtained from PCA is **16.3484**. It is better than that of autoencoder. Because a higher PSNR not always means better visualization result. Since the PSNR is calculated over the whole image, images that are shifted a little bit would cause a low PSNR even the result looks better.

Q6.4 The number of learned parameters for autoencoder is $(1024 \times 32 + 32 \times 32 + 32 \times 32 + 32 \times 1024) + (32 + 32 + 32 + 1024) = 68704$;

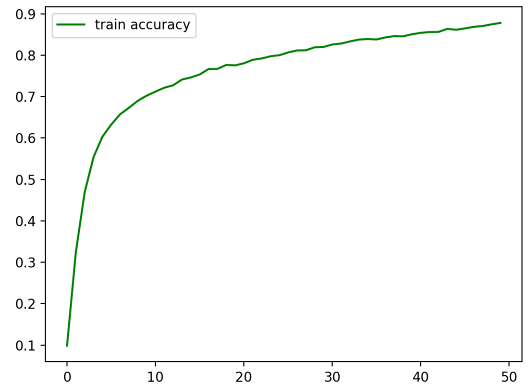
The number of learned parameters for PCA is $32 \times 1024 = 32768$.

The autoencoder has more parameters compared to PCA model. With more parameters, combined with linear and non-linear representation, the autoencoder can extract the features better thus has better performance.

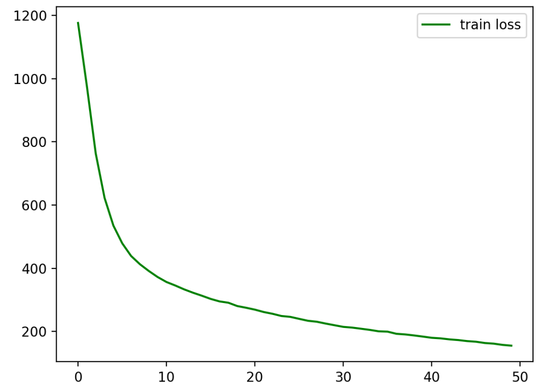
7. PyTorch

Q7.1.1 Please refer to the code in `run_q7_1_1.py`.

The plotted accuracy and loss over time are shown as Fig.11.
The accuracy of training set on this trained model is 87.75%; The accuracy of test set is 79.39%.



(a) accuracy

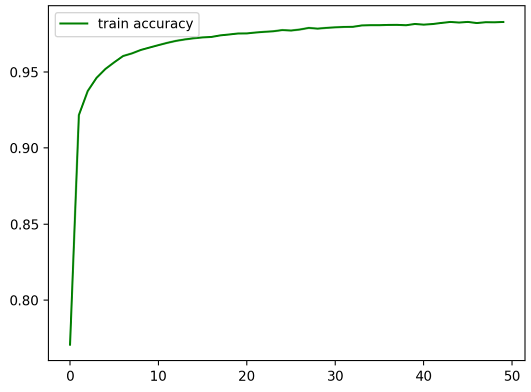


(b) cross-entropy loss

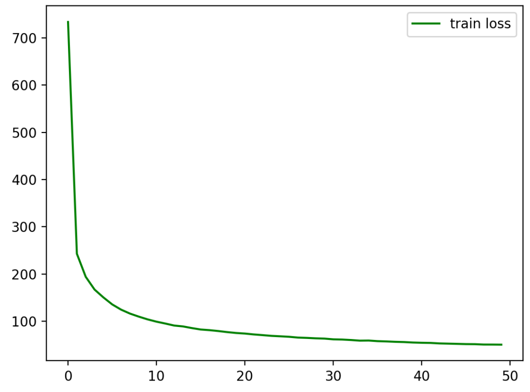
Figure 11: The accuracy and cross-entropy loss on training sets for the fully-connected network on NIST36 in PyTorch.

Q7.1.2 Please refer to the code in `q7_1_2.py`.

The plotted accuracy and loss over time are shown as Fig.12.
The accuracy of training set on this trained model is 98.49%; The accuracy of test set is 97.79%.



(a) accuracy

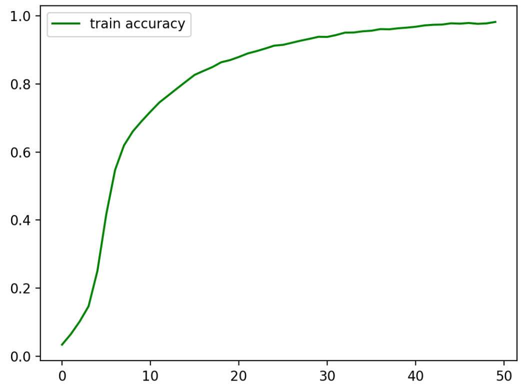


(b) cross-entropy loss

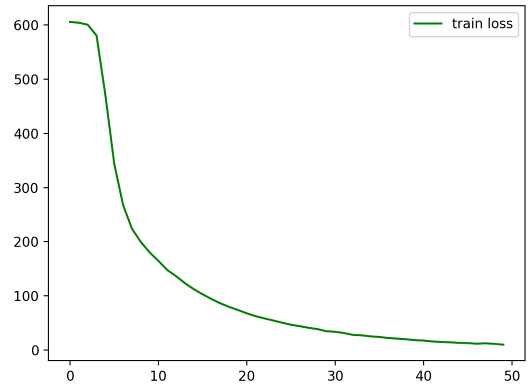
Figure 12: The accuracy and cross-entropy loss on training sets for the convolutional neural network with PyTorch on MNIST.

Q7.1.3 Please refer to the code in `q7_1_3.py`.

The plotted accuracy and loss over time are shown as Fig.13.
The accuracy of training set on this trained model is 98.21%; The accuracy of test set is 88.78%.



(a) accuracy

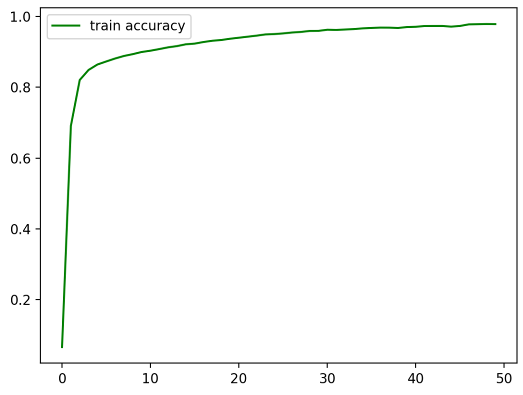


(b) cross-entropy loss

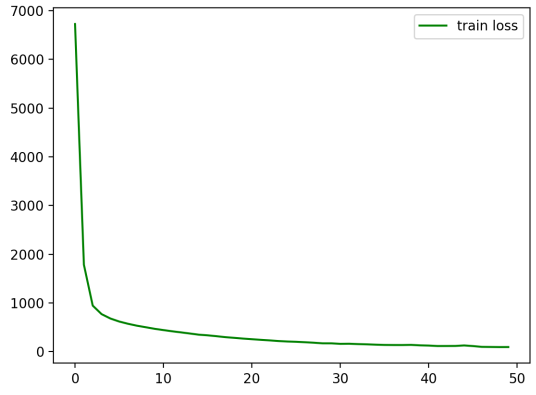
Figure 13: The accuracy and cross-entropy loss on training sets for the convolutional neural network with PyTorch on NIST36 dataset.

Q7.1.4 Please refer to the code in `q7_1_4.py`.

The plotted accuracy and loss over time are shown as Fig.14.
The accuracy of training set on this trained model is 97.84%; The accuracy of test set is 85.90%.



(a) accuracy



(b) cross-entropy loss

Figure 14: The accuracy and cross-entropy loss on training sets for the convolutional neural network with PyTorch on EMNIST Balanced dataset.

The recognition results on the findLetters bounded boxes from the images folder are shown as below. The recognition accuracy for each image is provided in the table. The overall accuracy is 84.15%.

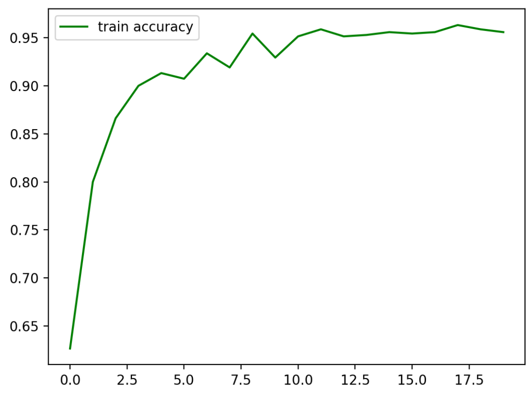
`01_list.jpg`	`02_letters,jpg`
TO DO LIST	ABCDEFG
TD DO LZST	ABCDEFG
1 MAKE A TO DO LIST	HIJKLMN
D MAKE A TD WQ LIST	HIIKLMN
2 CHECK OFF THE FIRST	OPQRSTU
2 CHECK DFF THE FIRST	QPQRSTW
THING ON TO DO LIST	VWXYZ
THDNG QN TO DO CIST	VWXYZ
3 REALIZE YOU HAVE ALREADY	1234567890
3 READIBE YOQ HAVE ALREADY	FR345G789Q
COMPLETED 2 THINGS	
COMPBETED D THIMGF	
4 REWARD YOURSELF WITH	
4 REWARD YOURSEBF WITH	
A NAP	
A NAP	
Accuracy: 84.35%	**Accuracy: 80.56%**
`03_haiku.jpg`	
HAIKUS ARE EASY	
HAIKNS ARE EASY	
BUT SOMETIMES THEY DONT MAKE SENSE	
BWT S0METIME5 THEY DOMT MAKG SENSE	
REFRIGERATOR	
REERIGERATOR	
Accuracy: 87.04%	
`04_deep.jpg`	
DEEP LEARNING	
DFEP UEARNING	
DEEPER LEARNING	
DEFPFR LEARNDMG	
DEEPEST LEARNING	
DEEPEST LEARQING	
Accuracy: 82.93%	

7.2 **Fine Tuning**

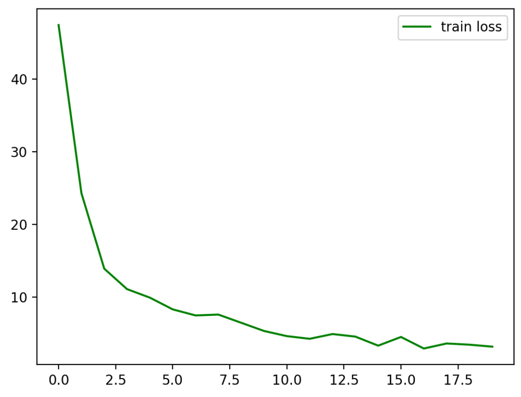
Q7.2.1

For the fine-tuned model using squeezenet1_1, please refer to the code in `run_q7_2_finetune.py`. After training, the accuracy on the test dataset is 92.65%.

The training precision and loss over the training set are shown as Fig.15.



(a) accuracy

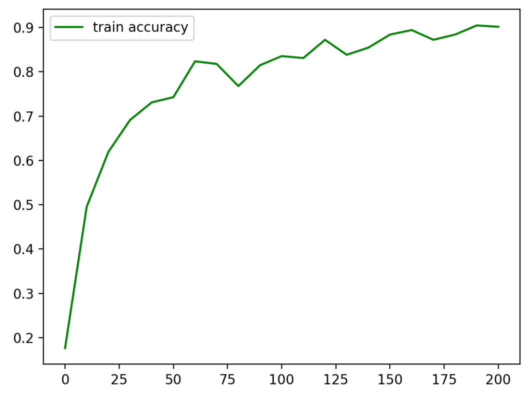


(b) cross-entropy loss

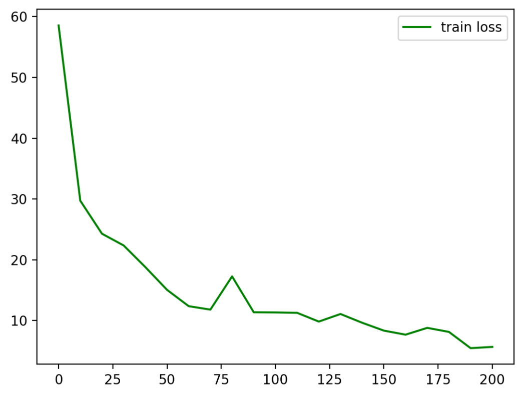
Figure 15: The accuracy and cross-entropy loss on training sets for the fine-tuned model.

For the model trained from the scratch, please refer to the code in `run_q7_2_scratch.py`. After training, the accuracy on the test dataset is 79.41%.

The training precision and loss over the training set are shown as Fig.16.



(a) accuracy



(b) cross-entropy loss

Figure 16: The accuracy and cross-entropy loss on training sets for the convolution network from scratch.

Compared to the self-designed network that was trained from the scratch, the fine-tuned model performs much better. The fine-tuned model converges much quicker with only few epochs, and it also reaches higher recognition precision. The self-designed model needs a lot more epochs to converge and the precision is not so high.