16-720B Homework 4 Write-up

Gu, Qiao

November 23, 2019

Q1.1

The given translation is applied to all x_j for $j \in \mathbb{R}$. Then for each $x_i i$

$$softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c} = \frac{e^{x_i}e^c}{(\sum_j e^{x_j})e^c} = \frac{e^{x_i}}{\sum_j e^{x_j}} = softmax(x_i). \tag{1}$$

This shows that softmax is invariant to translation.

When $c = -\max x_i$, all $x_i + c < 0$ and $e^{x_i+c} \in (0, 1)$, and the difference between e^{x_I} is relatively small. And when $c = 0$, e^{x_i} can be exponentially large, gradient may be dominated by a certain x_i and may cause numerical instability.

Q1.2

- $softmax(x_i)$ is probability, meaning that each element of softmax $softmax(x_i)$ is in range $(0, 1)$, and the sum over all elements is $\sum_j softmax(x_j) = 1$.
- probability distribution.
- $s_i = e^{x_i}$ is to map each x_i to its probability weight. $S = \sum s_i$ is find the sum of the weights. $softmax(x_I) = \frac{1}{S} x_i$ is to normalize each weight by all weights to get probability.

Q1.3

Each layer of a neural network can be written mathmatically as $f_i(\mathbf{x}) = \mathbf{W}_i\mathbf{x}+\mathbf{b}$, and thus is we concatenate n layers together without non-linear layers, we get the output as

$$\begin{aligned}
\mathbf{y} &= f_n(f_{n-1}(\cdots f_1(\mathbf{x})\cdots)) && (2)\\
&= \mathbf{W}_n(\mathbf{W}_{n-1}(\cdots(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)\cdots)+\mathbf{b}_{n-1})+\mathbf{b}_n && (3)\\
&= \mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_1\mathbf{x}+\mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_2\mathbf{b}_1+\mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_3\mathbf{b}_2+\cdots+\mathbf{W}_n\mathbf{b}_{n-1}+\mathbf{b}_n && (4)\\
&= \mathbf{W}\mathbf{x}+b, && (5)
\end{aligned}$$

where $\mathbf{W} = \mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_1$ and $\mathbf{b} = \mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_2\mathbf{b}_1+\mathbf{W}_n\mathbf{W}_{n-1}\cdots\mathbf{W}_3\mathbf{b}_2+\cdots+\mathbf{W}_n\mathbf{b}_{n-1}+\mathbf{b}_n$. This can be regarded as a single linear layer, and the whole network is equivalent to linear regression.

Q1.4

$$\frac{d\sigma(x)}{dx} = \frac{d}{dx}\frac{1}{1+e^{-x}} \tag{6}$$

$$= (-1)\frac{1}{(1+e^{-x})^2}\frac{d}{dx}(1+e^{-x}) \tag{7}$$

$$= (-1)\frac{1}{(1+e^{-x})^2}(-e^{-x}) \tag{8}$$

$$= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \tag{9}$$

$$- \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \tag{10}$$

$$= \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}}) \tag{11}$$

$$= \sigma(x)(1-\sigma(x)) \tag{12}$$

Q1.5

The loss function is unknown and therefore we assume $\frac{\partial J}{\partial y_i} = \delta_i$. Therefore

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial W_{ij}} = \delta_i x_j \tag{13}$$

$$\frac{\partial J}{\partial x_j} = \sum_{i=0}^{k}\frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial x_j} = \delta_i \sum_{i=0}^{k} W_{ij} \tag{14}$$

$$\frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial b_i} = \delta_i \tag{15}$$

And then we can further rewrite it to matrix form

$$\frac{\partial J}{\partial \mathbf{W}} = \delta \mathbf{x}^T \tag{16}$$

$$\frac{\partial J}{\partial \mathbf{x}} = \mathbf{W}^T \delta \tag{17}$$

$$\frac{\partial J}{\partial \mathbf{b}} = \delta \tag{18}$$

Q1.6

1. As shown in Figure. 1, when the input to the sigmoid x is far away from zero, the magnitude of the gradient becomes very close to zero and thus the gradient from higher layers are scaled by a very small number, even "vanishing". Therefore, when we update the network weights, the changes will be very small.

2. The output range of sigmoid function is $(0, 1)$ and the output range of $tanh(x)$ is $(-1, 1)$. If our input data are centered at 0, the output given by tanh are also centered at 0, which will make the input to different layers consistent.

3. From Figure. 1, we can see that $\tanh(x)$ has a stronger gradient (larger magnitude) than $\sigma(x)$ does.

4. $\tanh(x) = 2\sigma(2x) - 1$ as

$$\begin{aligned} \sigma(x) = \frac{1}{1+e^{-x}} \Rightarrow \sigma(2x) = \frac{1}{1+e^{-2x}} \Rightarrow 2\sigma(2x) = \frac{2}{1+e^{-2x}} \\ \Rightarrow 2\sigma(2x) - 1 = \frac{2-1-e^{-2x}}{1+e^{-2x}} = \frac{1-e^{-2x}}{1+e^{-2x}} = \tanh(x) \end{aligned} \tag{19}$$

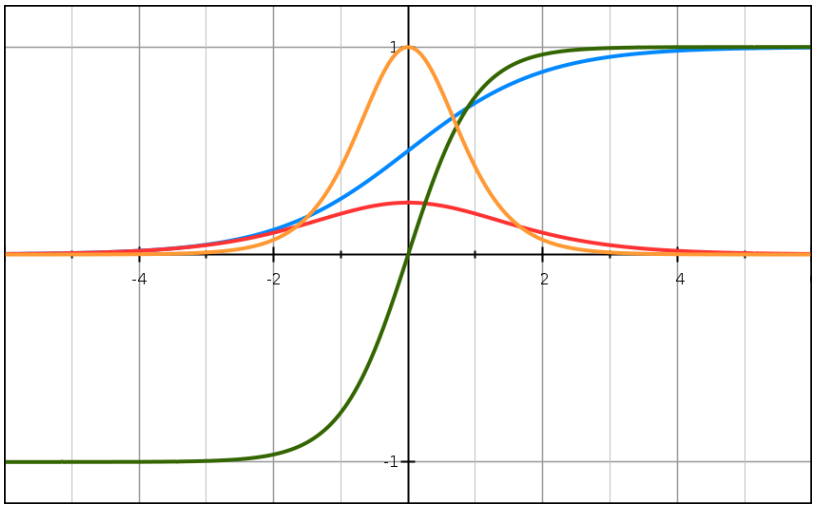


Figure 1: The plot of sigmoid function $\sigma(x)$ (in blue), its gradient $(1 - \sigma(x))\sigma(x)$ (in red), the $\tanh(x)$ function (in green) and its gradient $1 - \tanh^2(x)$ (in orange).

Q2.1.1

If the weights and biases of every layer are initialized with zeros, the outpuut layer pre-activation values will be zeros regardless the input data and the post-activation values will be a uniform probability distribution. In this case, there will still be gradients in the network due to the softmax loss, but they will be very small. Therefore the update of the network is very slow, the training process may terminate without convergence or the network may be stuck at local optima.

Q2.1.3

Because if we initialize all the weights with the same number, some of them will have the same effect on the output values, and the gradients on them will be the same during backpropagation. Therefore, they will move together and hard to converge during training.

Because in one layer of neural net $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, the variance of the output $\mathbf{y}$ depends on both the variance of input $\mathbf{x}$ and variance of elements of the weight $\mathbf{W}$ (output variance will be scaled by the number of input/output channels w.r.t the input variance). This is also true during the backpropagation of gradients. Therefore we scale the weight variance down by the layer size, To make the output variance of each layer equal to the input variance.

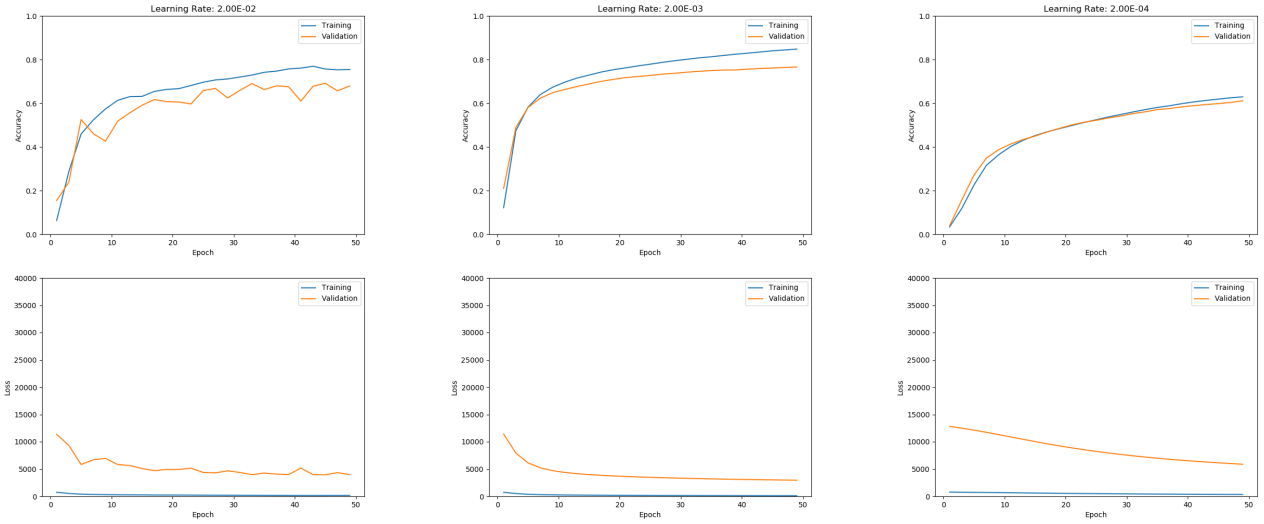
Q3.1.2

The progression of the loss and accuracy during different training processes is shown in Figure. 2. Here the best learning rate I found is 2×10^{-3}.

Learning rates affect training process in the sense that the best learning rate gives the best convergence. Overly high or overly low learning rates will both slow down the convergence or make the training not able to converge to local optima.

Additionally, we can notice that large learning rates may cause the learning process to overshoot, which causes accuracy and loss to fluctuate during the training process.

The final test accuracy of the best network is 78%.



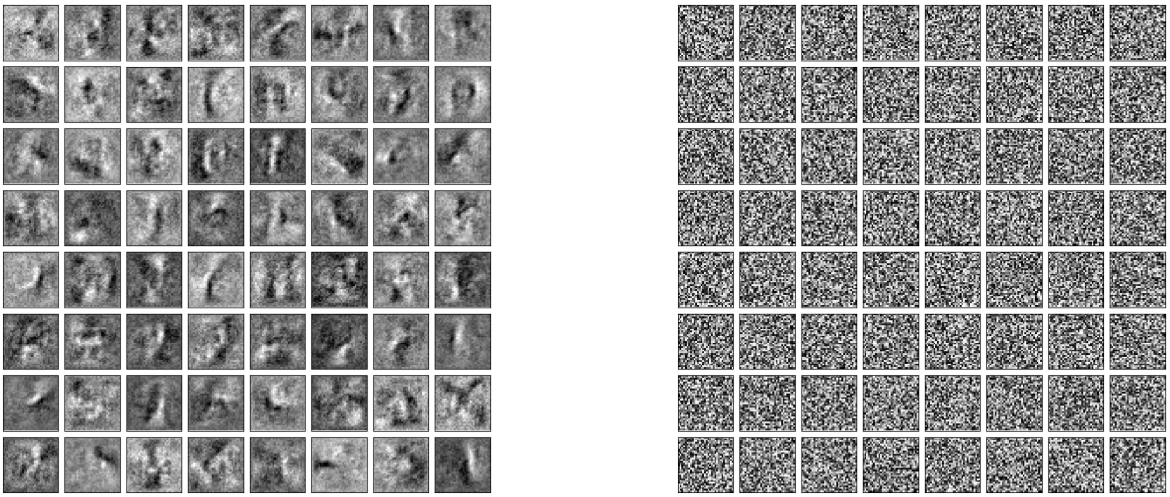
(a) Learning rate $= 2 \times 10^{-2}$ (b) Learning rate $= 2 \times 10^{-3}$ (c) Learning rate $= 2 \times 10^{-4}$

Figure 2: Training progress on loss and accuracy using different learning rates.

Q3.1.3

The weights of the first layer of the network is visualized in Figure. 3. As we can observe, the weights after training is more structured and some of them resemble stroke curves of different digits and characters, and thus when we apply these weights, the input image that have similar strokes will have higher activation.

And before training, the visualized weights are just random noise and no structures or patterns can be observed.



(a) After training

(b) Before training

Figure 3: Visualization of weights before and after training.

Q3.1.4

The confusion matrix of the best model on the testing dataset is shown in Figure. 4. As shown by some entries with high values in the matrix, the network is likely mix the digit "0" with the character ''O", the character "S" with the digit "5", the character "Z" with the digit "2" and digit "1" with character "I". These results are quite reasonable as these pairs have very similar shape and even some human can get confused by them.

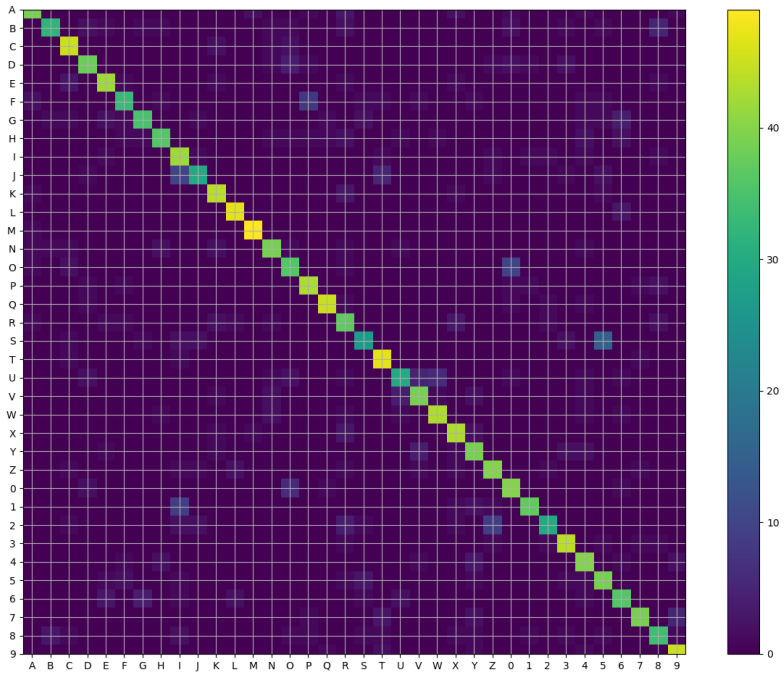


Figure 4: The confusion matrix of the best model on the testing dataset ($N = 1800$).

Q4.1

The first assumption made is that each character is connected within itself and well separate from others, although the morphological operations can help this to some extend. This means the method could fail if a character is splitted by background pixels or is connected to others by foreground pixels. Some examples where the method could fail are shown in Figure. 5 (a).

The second assumption is that all character are written in the upright form (no large rotation is applied), each line of text is written horizontallly and text is written line by line. The method could fail if the text are written with large rotation, as shown in Figure. 5 (b).

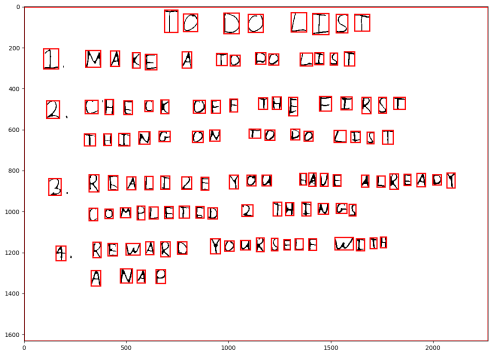
(a) The characters that are connected to each other ("ABC") or divided into multiple parts ("D").

(b) The characters that are not written in horizontal and rotated.

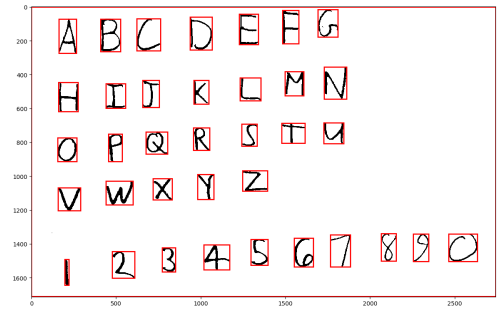
Figure 5: Possible failure cases of the text extraction pipeline.

Q4.3

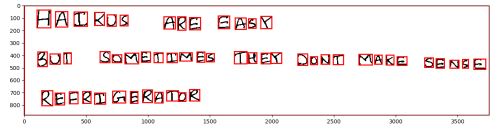
The character detection results are shown in Figure. 6. Note that the bounding boxes are directly estimate from the connected pixels and thus they are not padded by now. And we can see that all characters and digits are successfully detected and located, which means the accuracy of `findLetters(..)` is 100%.



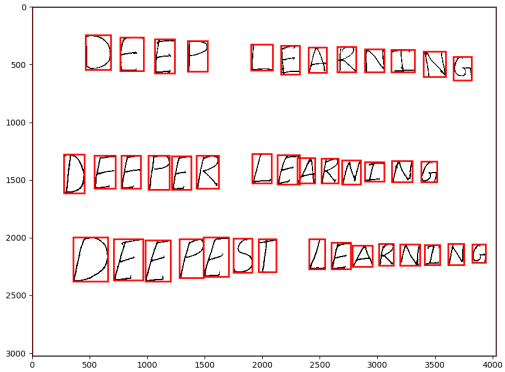
(a) The detection result of the first image.



(b) The detection result of the second image.



(c) The detection result of the third image.



(d) The detection result of the fourth image.

Figure 6: Detection results of characters in images in bounding boxes.

Q4.4

The extracted texts from each image and their accuracies are shown as follows:

```
Image: 01_list.jpg
T0D0LI5T
IMAKEAT0DOLIST
2CHFCK0FETHIFIRST
THING0NT000LIST
3RIALIZEY0UHAVEALR2ADT
C0MPL5T5D2THINGS
4XXWARDY0URSELFWITH
ANAP
Accuracy: 0.800

Image: 02_letters.jpg
XBCDEFG
HITKLMN
0PQRSTU
VWXYZ
1X3GS67X90
Accuracy: 0.806

Image: 03_haiku.jpg
HAIKUSAREEASY
EUTSQMETIMESTREYDONTMAK2SENQE
REFRIGERAT0R
Accuracy: 0.852

Image: 04_deep.jpg
DEEPLEARMING
CEEPTRLEARNING
5EEAE5TIEARNING
Accuracy: 0.829
```

Q5.2

The plotted training loss is shown in Figure. 7.

As training process progresses, the training loss first drops quickly and then plateau at the low level. The overall training loss curve is very smooth. This shows the momentum used in gradient descent can help speed up the loss reduction process.

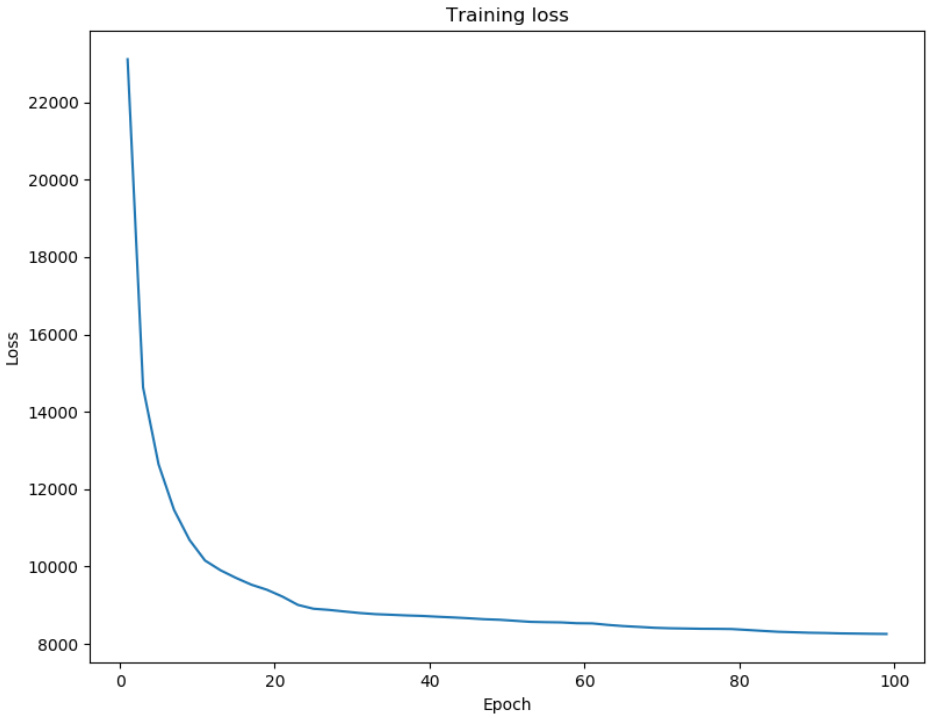


Figure 7: The training loss curve of the Autoencoder.

Q5.3.1

The validation images and their reconstructions results are shown in Figure. 8. As shown in the figure, the reconstructions results is more blur compared to the input images. Moreover, there is circle "area of reconstruction", and the contents within that circle can be better reconstructed than those outside (As shown by the first row of Figure. 8). This is probably because the contents of training images are mostly centered in that circle.

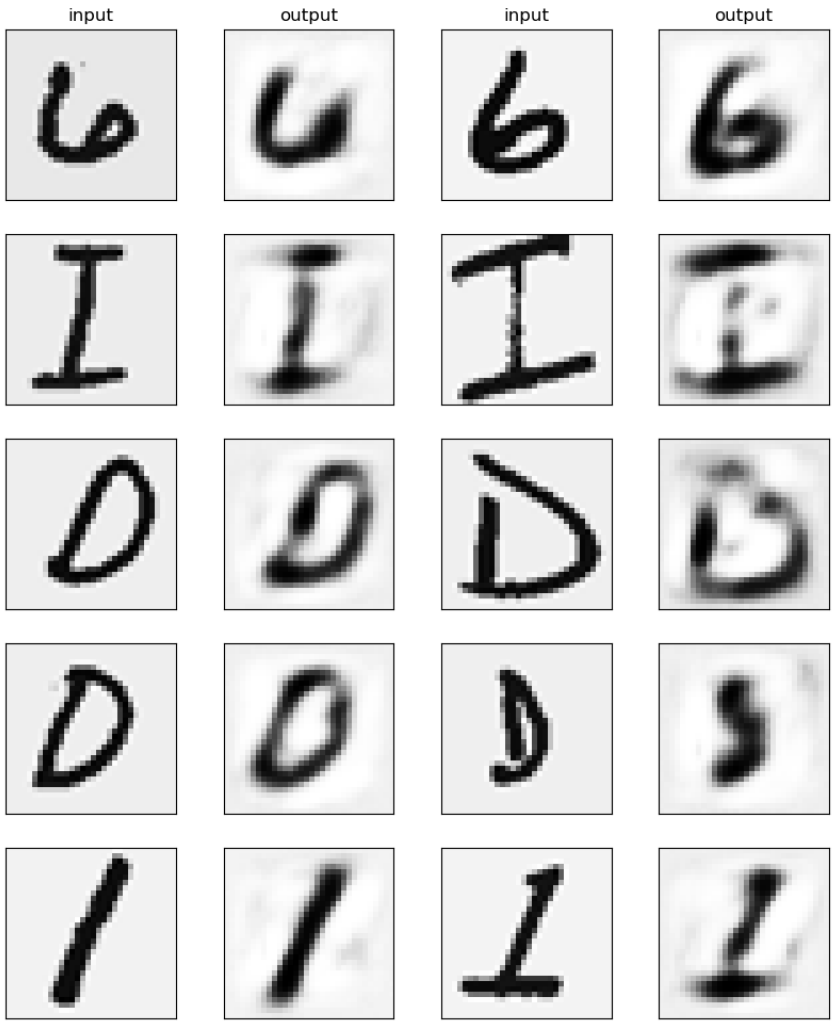


Figure 8: The validation images (input) and their reconstructions (output) using Autoencoder.

Q5.3.2

The average PSNR over the validation set is 16.085.

Q6.1

Note in this theory question, I treat the dimension of the data matrix to be (number of features, number of datapoints), and thus during the whole deduction, the dimension is the transpose of default setting in implementation.

Suppose our data matrix is $\mathbf{A} \in \mathbb{R}^{M \times N}$, where N is the number of data points in the dataset and M is number of dimension for each data point (here is the number of pixels in each image). Then each column of $\mathbf{X}$ is a datapoint, and consider the SVD decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \sum_{i=1}^{K} \mathbf{u}_i \sigma_i \mathbf{v}_i^T, \tag{20}$$

where $K = \min\{M, N\}$, $\mathbf{u}_i$, $\mathbf{v}_i$ is the i^{th} column of $\mathbf{U}$ and $\mathbf{V}$ respectively, and σ_i is the i^{th} diagonal element in $\mathbf{S}$.

To give the best k-rank approximation of $\mathbf{A}$ in terms of frobenius, we can simply take the first k terms in the summation in Eq. 20, which means

$$\hat{\mathbf{A}} = \sum_{i=1}^{k} \mathbf{u}_i \sigma_i \mathbf{v}_i^T = \mathbf{X}\mathbf{C}, \tag{21}$$

where the principle components $\mathbf{X}$ and the compressed data matrix $\mathbf{C}$ are defined to be

$$\mathbf{X} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \dots \ \mathbf{u}_k] \quad \mathbf{C} = \begin{bmatrix} \sigma_1 \mathbf{v}_1^T \\ \sigma_2 \mathbf{v}_2^T \\ \dots \\ \sigma_k \mathbf{v}_k^T \end{bmatrix}, \tag{22}$$

where each column of $\mathbf{C}$ is a compressed representation of the original data point in the corresponding column of $\mathbf{A}$.

To get $\mathbf{C}$ for any other input $\mathbf{A}'$, we can simply simply calculate $\mathbf{X}^T\mathbf{A}'$, where we can define $\mathbf{P} = \mathbf{X}^T$ as the projection matrix In our case, $M = 1024$ and $k = 32$, and thus the size of the projection matrix $\mathbf{P}$ is (32×1024) and its rank is 32.

Note: In the case where the first dimension is the number of datapoint (as in our implementation), the shape of the projection matrix will be (1024×32) and needs to be right-multiplied to the input data matrix.

Q6.2

The validation images and their reconstructions results using PCA are shown in Figure. 8. As we can see, besides that the output are blur, the reconstructed images are generally more gray.

The average PSNR over the validation set is 16.347, which is higher than that obtained from Autoencoder, and means better reconstruction quality.

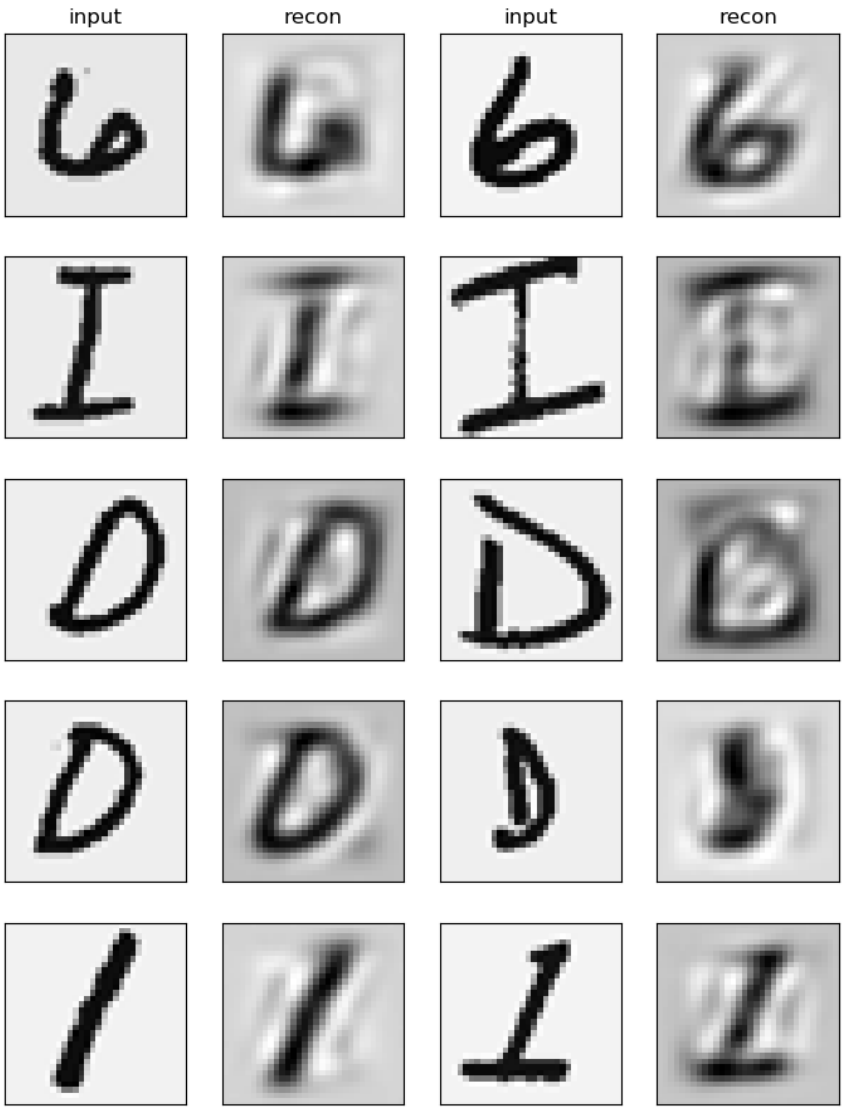


Figure 9: The validation images (input) and their reconstructions (recon) using PCA.

Q7.1.1

For this question, please run `python run_q7_1.py 7.1.1`.

The loss and accuracy during the training process is show in Figure. 10

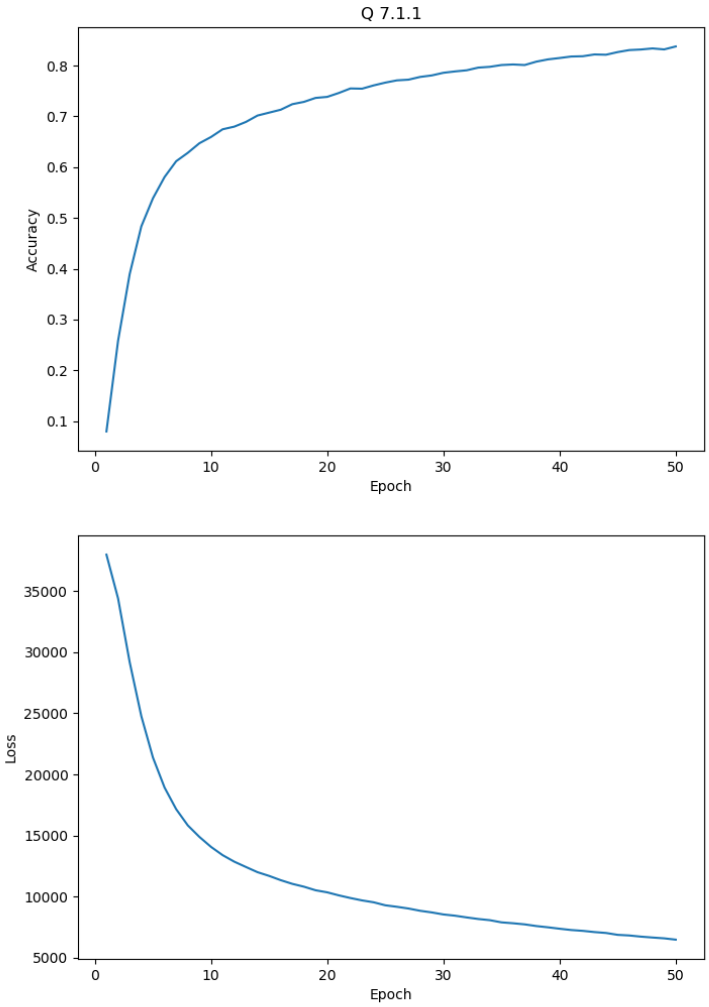


Figure 10: The training progress of the fully-connected network on NIST36 dataset.

Q7.1.2

For this question, please run `python run_q7_1.py 7.1.2`.

The loss and accuracy during the training process is show in Figure. 11

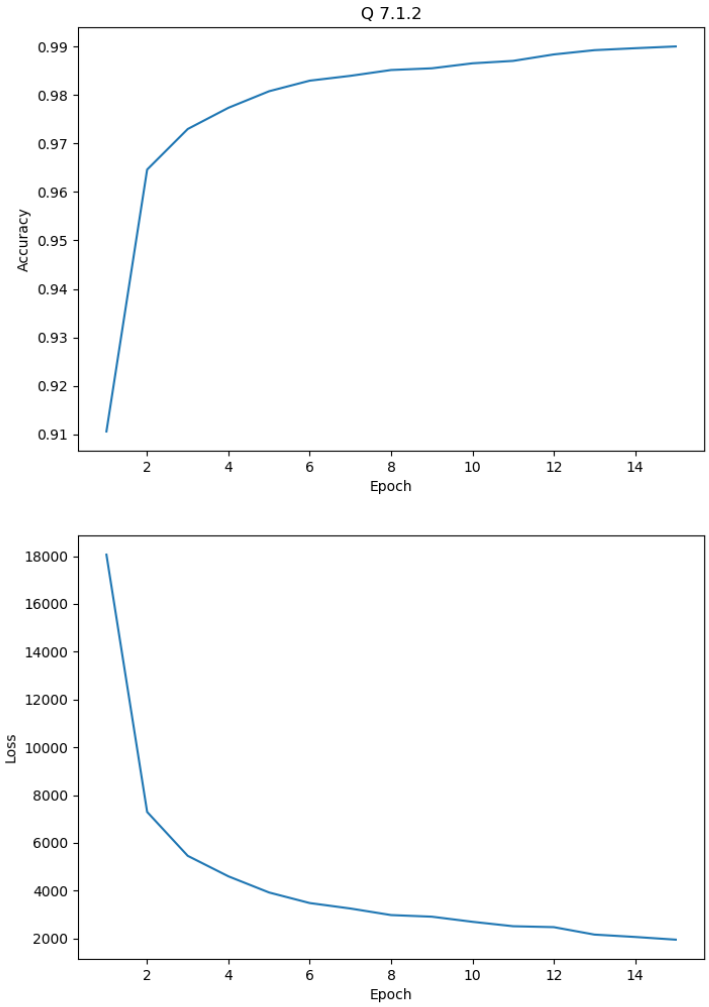


Figure 11: The training progress of a CNN on MNIST dataset.

Q7.1.3

For this question, please run `python run_q7_1.py 7.1.3`.

The loss and accuracy during the training process is show in Figure. 12

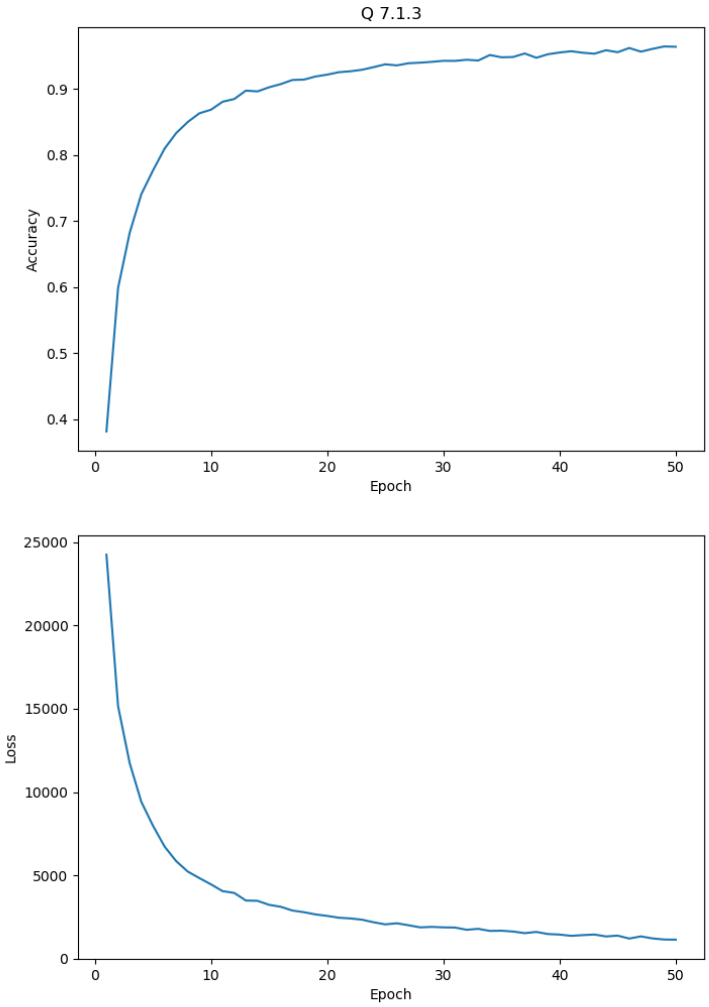


Figure 12: The training progress of a CNN on NIST36 dataset.

Q7.1.4

For this question, please run `python run_q7_1.py 7.1.4 train` to train the CNN on EMNIST Balanced dataset and `python run_q7_1.py 7.1.4 test` to test the trained CNN on the texts detected and cropped from the given 4 images.

The loss and accuracy during the training process is show in Figure. 13. The test results and accuracy on the text given by findLetters is shown as follows:

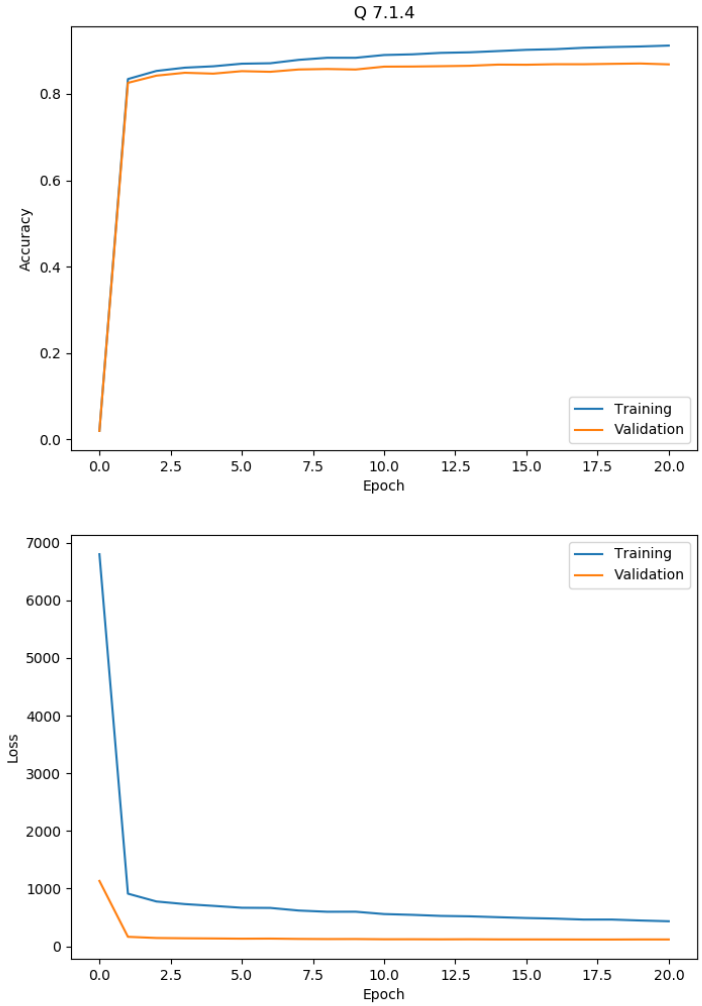


Figure 13: The training progress of a CNN on EMNIST Balanced dataset.

```
Image: 01_list.jpg
TODOLZST
IMgXEATOBOLISF
2CHECKOFFTHEFERST
```

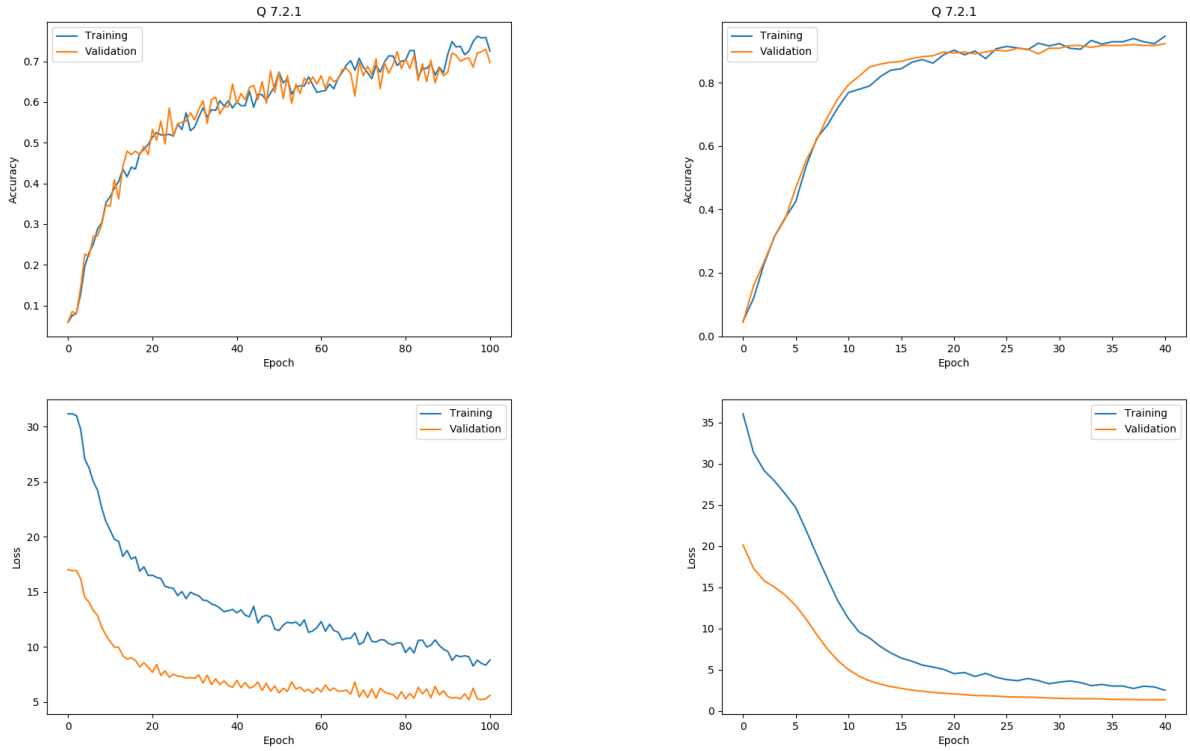
```
THINGQNTODgLE5T
3REALI2EYOUHBVEgLREADY
COMPLETED2THEWGS
gREWARDYOWRSELFWITH
ANgP
Accuracy: 0.757

Image: 02_letters.jpg
gBCDEFG
HIJKLMN
OPQRSTU
VWXYZ
I23q5G78gC
Accuracy: 0.806

Image: 03_haiku.jpg
HAIKUSAREEAgY
BUFSOMETqMEgTHEYDaNTMAKESEMgE
REfRgGERQTOR
Accuracy: 0.759

Image: 04_deep.jpg
DEEPLEARnIHG
BEEPERLEARNING
DEEPESJZEgRQING
Accuracy: 0.829
```

Q7.2.1

Please use one of the following lines of script to run the experiment using SqueezeNet or a small CNN on Oxford flowers 17 or flowers 102 dataset.

```
python run_q7_2.py 17 small
python run_q7_2.py 102 small
python run_q7_2.py 17 squeeze
python run_q7_2.py 102 squeeze
```

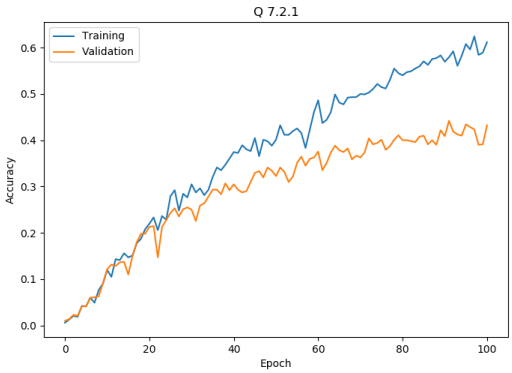
In this question, I designed a small CNN containing 3 convolution layers with ReLU activation, and dropout layers, followed by two-layer fully connected network. The training progress of above experiment are shown in Figure. 14 and Figure. 15. As we can see from those figures, the network based on pretrained SqueezeNet yeild faster training progress and better training results than the small CNN trained from scratch. Moreover, interestingly, during finetuning the SqueezeNet, we can notice a slowdown around epoch 5.



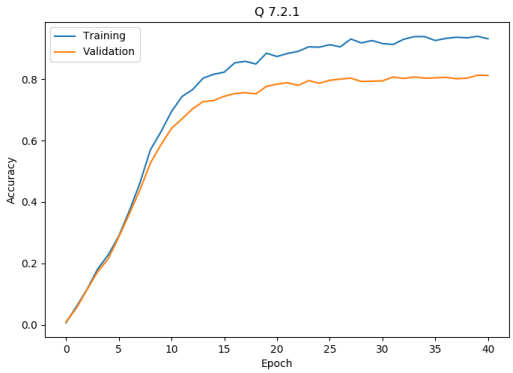
(a) Using a small CNN.

(b) Using pretrained SqueezeNet.

Figure 14: Training progress on Oxford flower 17 dataset.



(a) Using a small CNN.



(b) Using pretrained SqueezeNet.

Figure 15: Training progress on Oxford flower 102 dataset.