Lok Yin LEE
20450512
lyleeaf@connect.ust.hk

1 Theory

Q1.1

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

So that the softmax operation is invariant to translation.

When we use c = 0, the numerator of the softmax function have range (0,∞). When we use c = $-$max x_i, the range will become (0, 1]. In such case, we can restrict the number to not too high and the overall outcome remain the same.

Q1.2
As we will use the invariant property c = $-$max x_i, the range of each element is (0, 1]. The sum over all elements is 1.

Like the total sum of probability always is 1, softmax takes an arbitrary real valued vector x and turns it into a probability distribution.

$s_i = e^{x_i}$ change the input from normal form to a exponential form that increase the difference between values. $S = \sum s_i$ create the sum of all exponential form data. softmax$(x_i) = \frac{1}{S} s_i$ divide the input by the sum to find the distribution of input from all values.

Q1.3
For a multi-layer neural networks without a non-linear activation function, we will have:

$$\begin{aligned} y &= x_n W_n + B_n \\ &= (x_{n-1} W_{n-1} + B_{n-1}) W_n + B_n \\ &= (W_{n-1} W_n) x_{n-1} + (B_{n-1} W_n + B_n) \\ &= ..... \\ &= mx + c \end{aligned}$$

We can simply combine all the weight and bias to each matrix such as m and c. This will begin the network back to the linear form.

Q1.4

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

Q1.5

$$
\begin{aligned}
y_j &= \sum_{i=1}^{d} W_{ij} + b_j \\
\frac{\partial y_j}{\partial W_{ij}} &= x_i \\
\frac{\partial y_j}{\partial x_i} &= W_{ij} \\
\frac{\partial y_j}{\partial b_j} &= 1 \\
8\frac{\partial y}{\partial W} &= \begin{bmatrix} x_1 & \dots & x_1 \\ \vdots & \vdots & \vdots \\ x_d & \dots & x_d \end{bmatrix} = \begin{bmatrix} x & \dots & x \end{bmatrix} \in \mathbb{R}^{d \times k} \\
\frac{\partial y_j}{\partial x} &= W_j \in \mathbb{R}^{d \times 1} \\
\frac{\partial y}{\partial b} &= \mathbf{1} \in \mathbb{R}^{k \times 1} \\
\frac{\partial J}{\partial W} &= \sum_{j=1}^{k} \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial W} = x\delta^T \in \mathbb{R}^{d \times k} \\
\frac{\partial J}{\partial x} &= \sum_{j=1}^{k} \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial x} = W\delta \in \mathbb{R}^{d \times 1} \\
\frac{\partial J}{\partial b} &= \sum_{j=1}^{k} \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial b} = \delta \in \mathbb{R}^{k \times 1}
\end{aligned}
$$

Q1.6

1.

$$
\begin{aligned}
y &= \sigma(h_n(\sigma(h_{n-1}(\dots\sigma)\dots) \\
\frac{dy}{dx} &= \sigma'(h_n(\dots))h'_{n-1}(\dots)\sigma'(\dots)\dots
\end{aligned}
$$

As we know the $\sigma'(x)$ is always in range (0,1), the value of gradient will become smaller and smaller as the number of layer increase.

2. The range of sigmoid is (0,1) and range of tanh = (-1,1). The zero mean property of tanh make the network easier to convergence, since the sigmoid can only decrease or increase all weights together and zigzagging if all of the components of an input vector are positive.

3. The range of derivatives of tanh is (0,1) which is higher than the range of sigmoid, and this makes the gradient vanishing slower.

4.

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

2 Implement a Fully connected Network

2.1 Network Initialization

Q2.1.1
As we initialize a network with all zeros, all the weights and biases will be set to zero. In such case, when we first start to forward propagation, all the hidden node return zero signal and generate same output, all the gradient of neural with the same structure will be the same. As the gradient and the loss will be the same, many neural will also updated to a same state, which the neurals cannot result a decision with many consideration.

Q2

We implemented the Xavier initialization to all the weights.

Q2.1.3

The input of the next layer will be the sum of the current layer. If we set all the weights in different layer as the same value, the output of each layer will variate. As variance and the total amount of each layer become similar, when we performing the gradient descends, each layer will have similar gradient for weight updating. So that the gradient descend will let all the layers have similar contribution to the result, we can easier to get the optimal weighting as all the layer are well updated.

2.2 Forward Propagation

In this section, we create the function for forward propagation that get the activation function, forward propagation and the loss calculation.

Q2.2.1
We implemented the sigmoid function.

Q2.2.2
We implemented the softmax function.

Q2.2.3
We implemented the log loss function.

2.3 Backwards Propagation

Q2.3.1
We implemented the loop for back propagation from the error.

2.4 Training Loop

Q2.4.1
To perform the batch gradient descends, we implemented the function that randomly sample the data to different batches.

2.5 Numerical Gradient Checker

Q2.5.1
To check the whole gradient descend is valid, we implemented the Numerical Gradient Checker by checking the centre differences.

3 Training Models

In this section, we will apply all the previous function and combine them to a training loop and train a model.

Q3.1.1

Epochs	100	Learning Rate	0.02
Hidden Size	64	Batch Size	64

Table 1: Model Parameter for Q3 Model

For the training we use the training parameter shown as table 1. The model training result is shown in table 2.

Training Loss	3843.35	Training Accuracy	92%
Validation Accuracy	77.1%	Test Accuracy	77.3%

Table 2: Model Result for Q3 Model

Q3.1.2

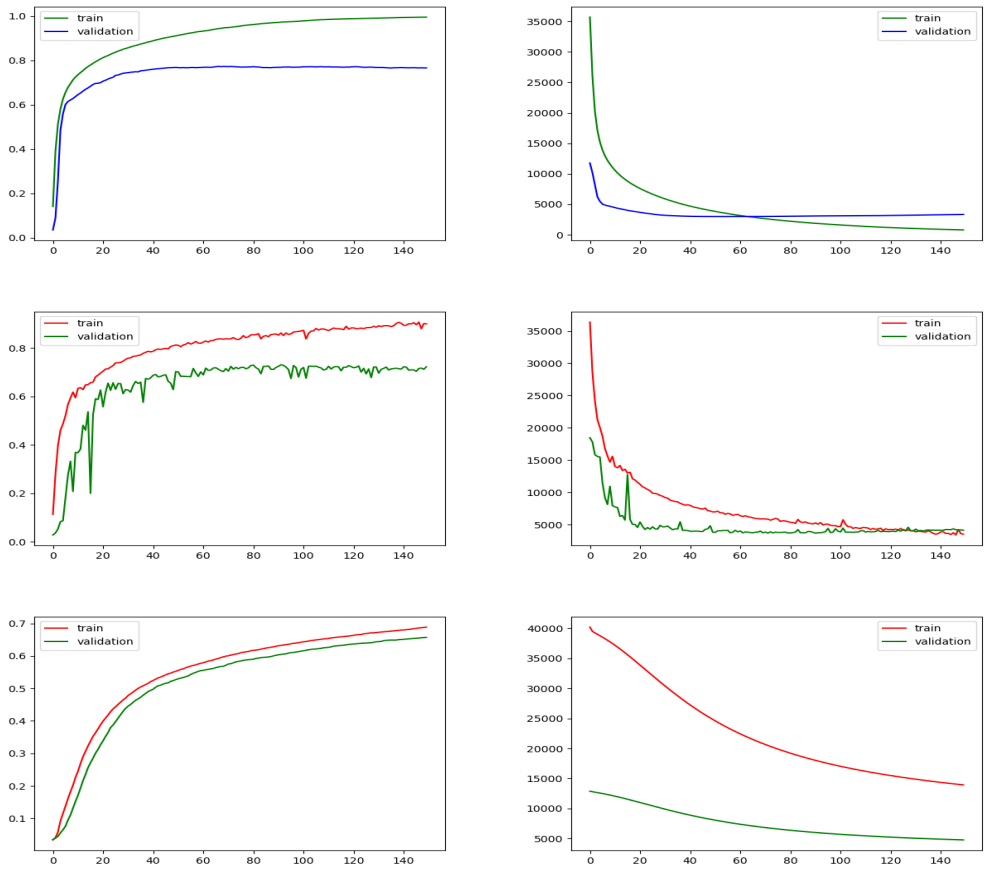

Figure 1: Model performance on three learning rate(Top: prefer, Middle: Larger, Bottom: Smaller)

As the results shown in figure 1, we can see the large learning rate will make the training loss hard to drop to a minimum point and a small learning rate will first drop to a local minimum which is not the optimal minimum that achieve a best accuracy. However, a suitable learning rate will able to get out a local minimum and optimize the value for the optimal minimum. This is the reason that the proper learning will result the lowest loss and best accuracy.

Q3.1.3

The visualization of the weight from first layer is shown in figure 2. We can clearly see that the randomly initialized is a random pattern which very different from most image features. After the training, the weight

shows some pattern like the Gaussian or derivative of the Gaussian. As this is machine learned, the filters may have different angle of Gaussian filter or some others similar with edge or corner filters.

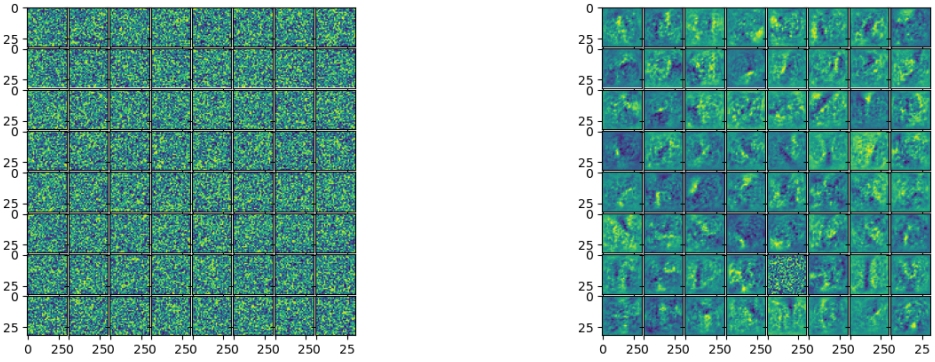

Figure 2: Visualization of the first layer on random and trained model.

Q3.1.4

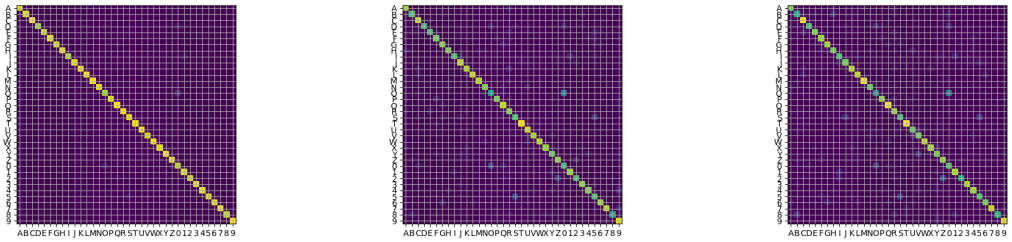

Figure 3: Confusion Matrix(train, validation and test data).

The confusion matrix is shown is figure 3. We can see the most of the prediction is correct, but some similar characters is confused. They are [0, O], [1,I], [2, Z] and [5, Z] which are consider as looking similar. So that it is reasonable to have such confusion.

4 Extract Text from Images

In this section,
Q4.1
The first assumption is that the character is not overlapping or separated, since we cannot detect it is one or two separated character. The second assumption is that the character have enough clearness and size.

As the image shown in figure 4, we can see the character is separated or overlapped. The number '1' maybe considered as two characters and the number '3' and '4' maybe consider as one character. For the right image, all the character is burr and the color is more closer to grey, the low intensity may let the program considered as the background.

Q4.2
We implemented the function that able to locate all the character in the image under our assumption.

Q4.3
The result of character detection is shown in figure 5.
Q4.4
As shown in table 3, we can see most of the character is well detected, and some case we mentioned above is confused.

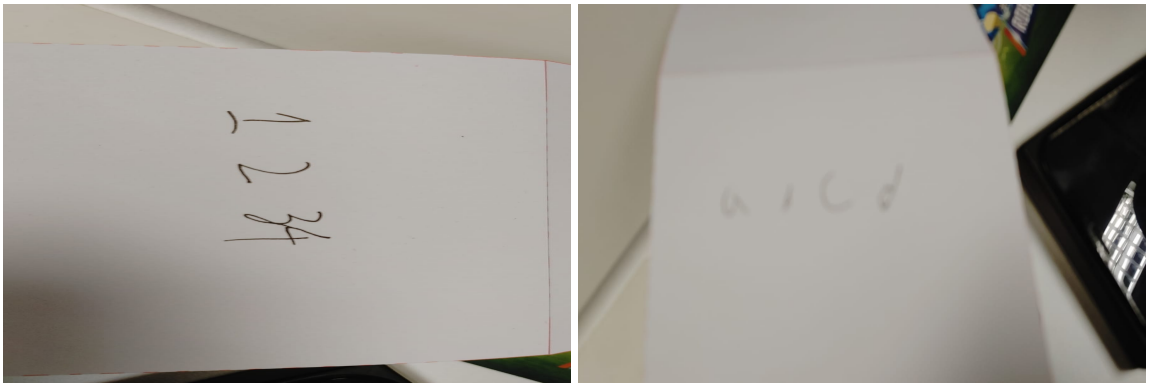

Figure 4: Two image with character that may not be detected.

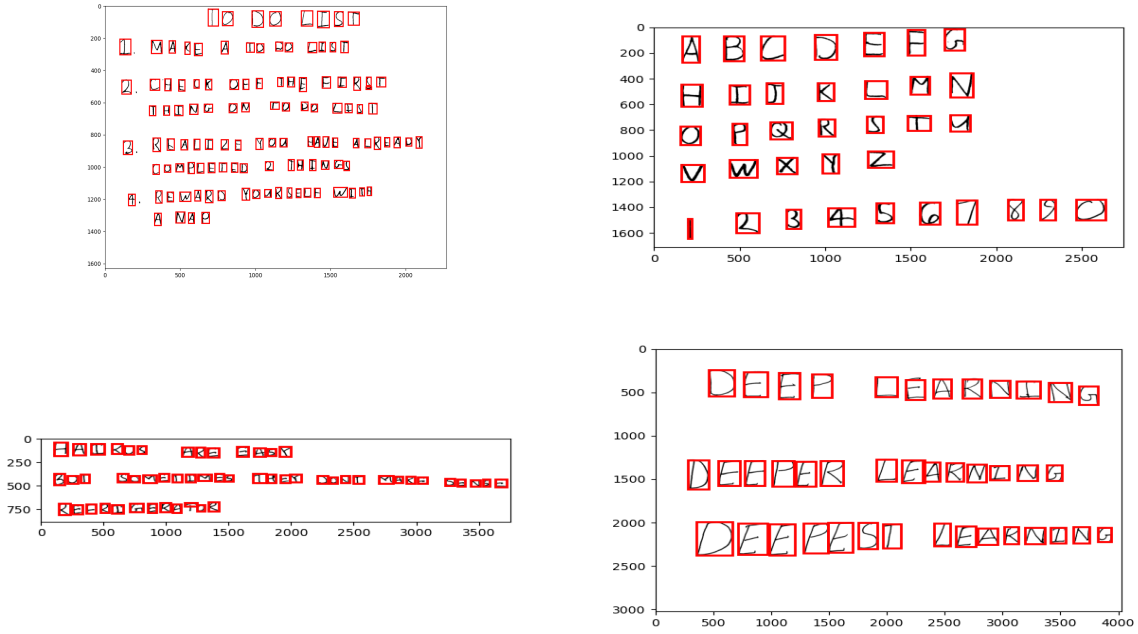

Figure 5: Detection results of the sample images.

5 Image Compression with Autoencoders

In this section, we will create an Autoencoder to reconstruct the charterer from the NIST36 dataset.

5.1 Building the Autoencoder

Q5.1.1
We implemented the autoencoder network with 4 layers and using relu activation function.

Q5.1.2
We implemented the momentum to make the network convergence faster.

5.2 Training the Autoencodedr

Q5.2
We applied the previous functions to train the autoencoder, the training loss is shown in figure 6. The loss is decreasing for the whole training process, but it become slower as pass more iterations. At the end of the training, the loss still maintain on a relative high value, which show the model may not sufficient for this task.

Ground Truth	Prediction
TODOLIST	T0D0LIST
1MAKEATODOLIST	LMAIEATODOLIST
2CHECKOFFTHEFIRST	2CHECKOFE3H82IRQUT
THINGONTODOLIST	3HINGONTQDQLIST
3REALIZEYOUHAVEALREADY	3REALIZEY0UHAVEALR6ADT
COMPLETED2THINGS	C0MPLETEDZYHINGS
4REWARDYOURSELFWITH	9REWARDYOURSELFWIIH
ANAP	ANAP
ABCDEFG	ABLDEF6
HIJKLMN	HIIKLMN
OPQRSTU	QPQRSTU
VWXYZ	VWXYZ
1234567890	1Z3GS6789O
HAIKUSAREEASY	HAIKUSARGGAGY
BUTSOMETIMESTHEYDONTMAKESENSE	BWTSQMETIMEGTAEYDOWTMAKGSGNGE
REFRIGERATOR	RBFRIGERAMQR
DEEPLEARNING	DEEPLEARBING
DEEPERLEARNING	DEEPERLEARAING
DEEPESTLEARNING	DEEPESILEARNING

Table 3: The Prediction of images.

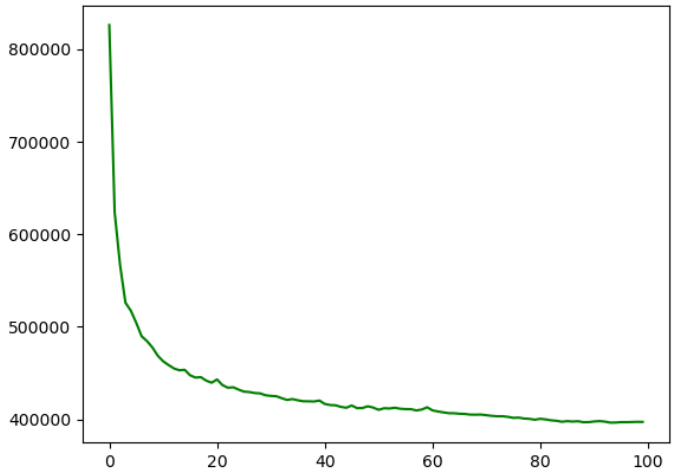

Figure 6: Loss function of the Autoencoder.

5.3 Evaluating the Autoencoder

Q5.3.1
The visualization of 10 images and their reconstruction is shown in figure 7.
Q5.3.2
As we using the function peak signal noise ratio function, we get the average RSNR 15.17860 on all validation image.

6 Comparing against PCA

In this section, we will use the PCA to reconstruct the image from the dataset.

Q6.1
The projection matrix have size 32×1024 with rank 32.

Q6.2

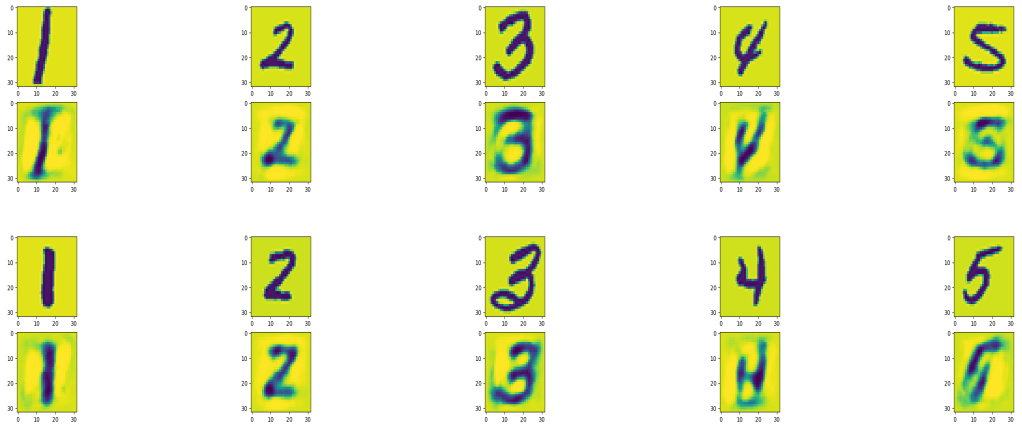

Figure 7: Input Image and their Reconstruction using Autoencoder.

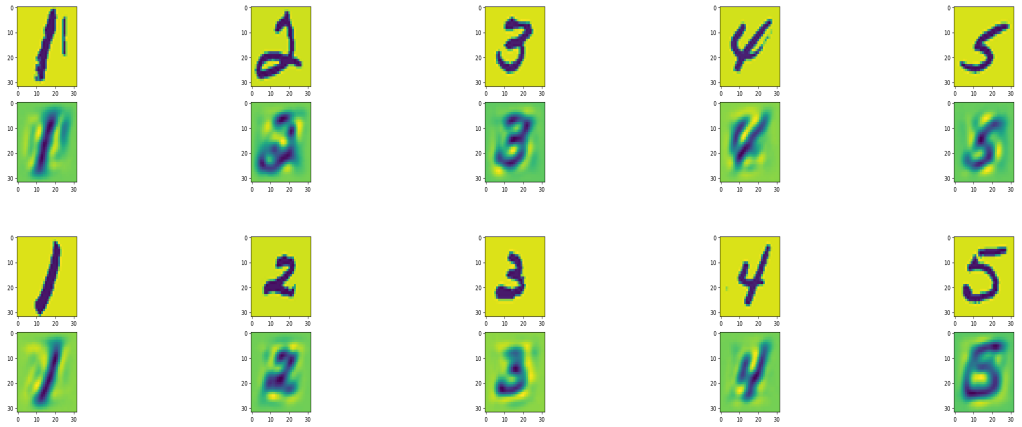

Figure 8: Input Image and their Reconstruction Using PCA.

As shown in figure 8, the PCA reconstructed image have a lot of noise and shadow beside the characters. It is clear that the characters are are also burr than the autoencoder one.

Q6.3
The PSNR score of the PCA method is 16.34842, which is higher than the one from autoencoder. Since PSNR is spatial sensitive, although the PCA images is more burr, the dominant shape of main frame of character exist and it give a higher score. Therefore, for this kind of image generation task, the PSNR or others metrics consist with human score.

Q6.4
The number of parameter of autoencoder and PCA are 68704 and 32768 respectively. Since the parameter of the autoencoder is more and it have activation function to create lot of non-linearity, the autoencoder should perform better.

7 PyTorch

7.1 Train a neural network in PyTorch

For this section, we implemented all the model in a single file `run_q7.py`.

Q7.1.1 We implemented the same fully-connected network in `PyTorch`. Except we re-use the data loader from previous questions, we add cross entropy loss, SGD optimizer, cosine annealing scheduler to train the model. Since the model is the same, we can only get a little improvement, which the accuracy of train, validation and test are 88.22186%, 88.22186% and 78.125%. The result is shown in figure 9.

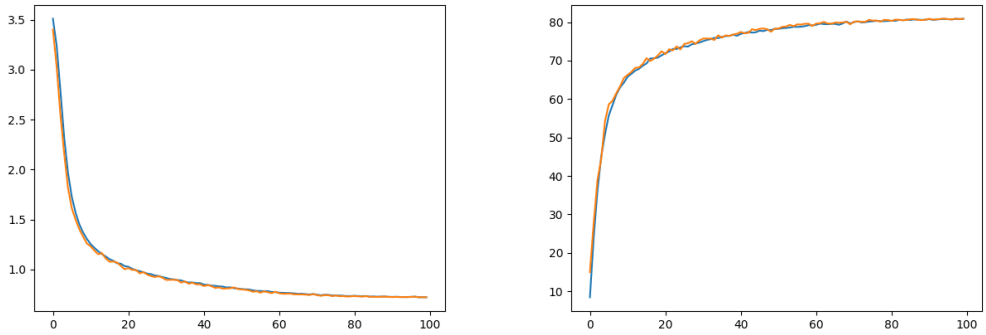

Figure 9: Two layer network for NIST36.(Left: Loss, Right: Accuracy)

Q7.1.2
We use the `MNIST` and `wide_resnet50_2` from `PyTorch` to achieve the best result. The accuracy of train, validation and test are 90.27333%, 98.5% and 99.413%. The result is shown in figure 10.

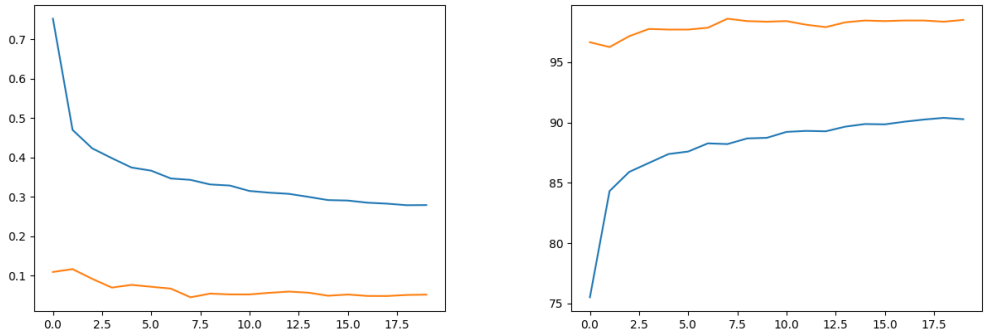

Figure 10: Convolutional neural network for MNIST.(Left: Loss, Right: Accuracy)

Q7.1.3
We use the `wide_resnet50_2` from `PyTorch` to achieve the best result. The accuracy of train, validation and test are 99.13194%, 99.82638% and 94.22669%. The result is shown in figure 11.

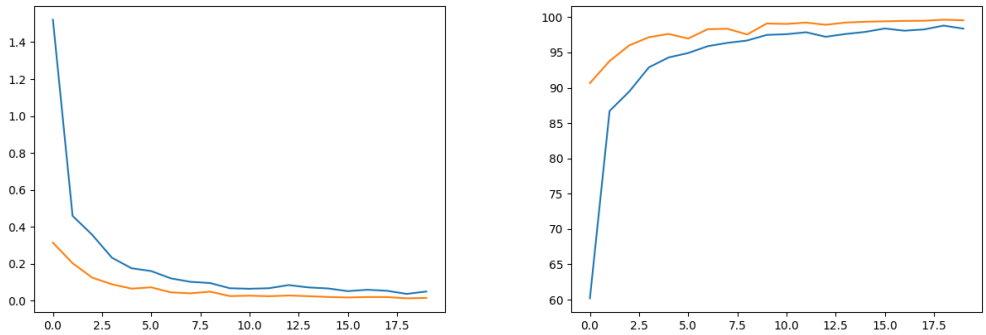

Figure 11: Convolutional neural network for NIST36 dataset.(Left: Loss, Right: Accuracy)

Q7.1.4
We use the `wide_resnet50_2` from `PyTorch` to achieve the best result. The accuracy of train, validation and test are 92.89628%, 90% and 90.44548%. The result is shown in figure 12.

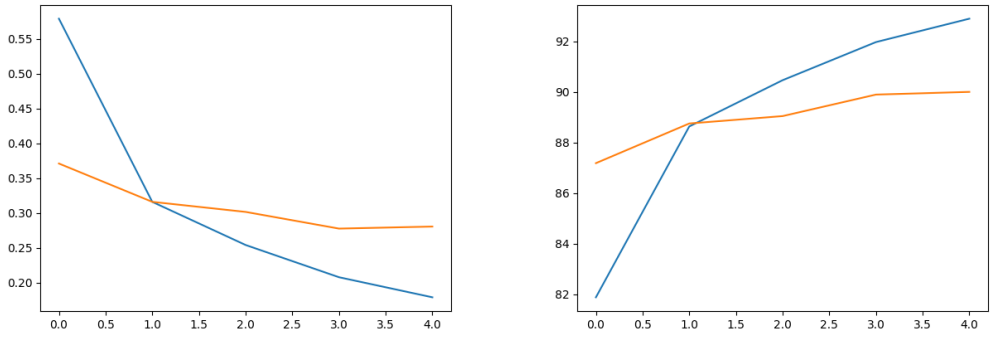

Figure 12: Convolutional neural network for EMNIST dataset.(Left: Loss, Right: Accuracy)

The prediction of sample images in shown in table 4.

Ground Truth	Prediction
TODOLIST	T0DOLIST
1MAKEATODOLIST	1MaKEaTODOLIST
2CHECKOFFTHEFIRST	2EHEeK0EEYHEEIReJT
THINGONTODOLIST	THINGONZ0DOLI5T
3REALIZEYOUHAVEALREADY	3REqLIZEY0UHqUEaZREaDY
COMPLETED2THINGS	e0MPLETEDZTHINg5
4REWARDYOURSELFWITH	qREWqRDY0URSELEWITH
ANAP	ANaP
ABCDEFG	qBCDEEG
HIJKLMN	HIJKLMN
OPQRSTU	0PQRSTY
VWXYZ	VWXYZ
1234567890	123q5678q0
HAIKUSAREEASY	HAIKUgAREEAgX
BUTSOMETIMESTHEYDONTMAKESENSE	BUTS0METIaEgTHEqD0NTMAKESENgE
REFRIGERATOR	REERIaERaZ0R
DEEPLEARNING	DEEPLEARNING
DEEPERLEARNING	DEEPERLEARNING
DEEPESTLEARNING	DEEPESTLEARNING

Table 4: The Prediction of images.

7.2 Fine Tuning

Q7.2.1

We use the `flowers_17` and `squeezenet1_1`. For the model train from scratch, the accuracy of train, validation and test are 80.58824%, 62.05882% and 63.82353%. For the model train from pre-trained model, the accuracy of train, validation and test are 100%, 93.82353% and 95.29412%. The result is shown in figure 13.

We can see that the one train from scratch not yet arrive the optimal point even we train it for 100 epoch. But the pre-trained model can fit our dataset with only 20 epoch. Therefore, the fine tune can help us to train network faster and get a better result.

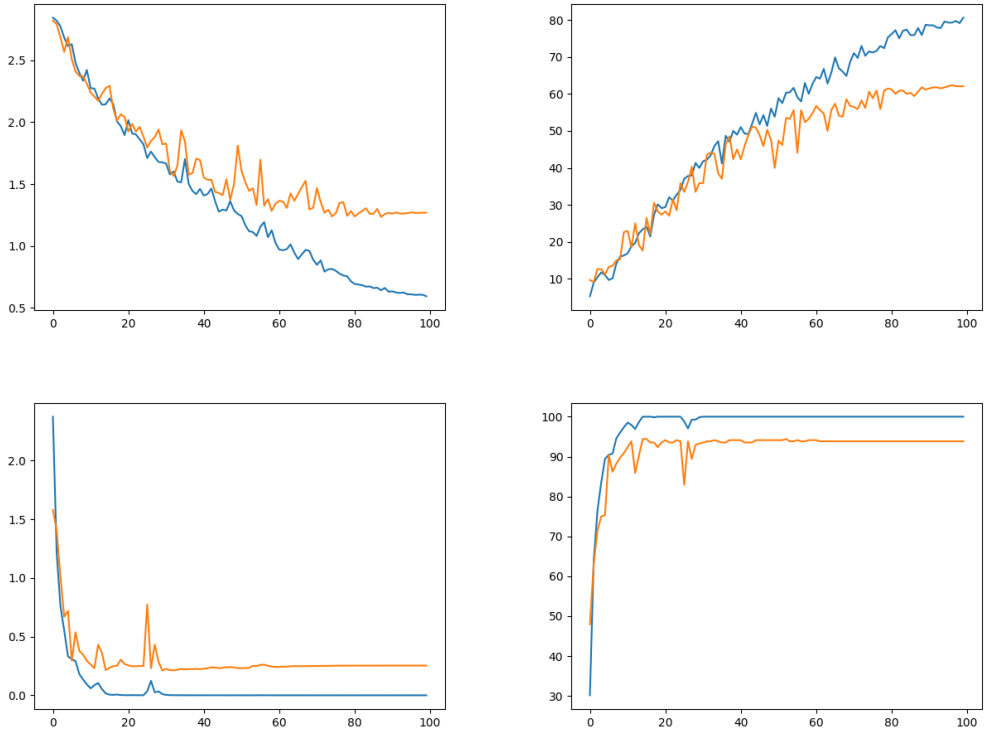

Figure 13: Comparison for fine tuning.(Left: Loss, Right: Accuracy, Top: Scratch, Bottom: Fine Tune)