Q1.1 Theory [2 points] Prove that softmax is invariant to translation, that is

$$softmax(x) = softmax(x + c)\ \forall c \in \mathbb{R}$$

Softmax is defined as below, for each index i in a vector x.

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

Often we use c = −max(x_i) Why is that a good idea? (Tip: consider the range of values that numerator will have with c = 0 and c = − max x_i)

Solution:

If we add a constant c to every dimension of x, then we have:

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

Therefore, softmax is invariant to translation.

If we let c = −max(x_i), then we will get the range of numerator would be (0, 1]. On the other hand, if we use c = 0, then we will get (0, ∞]. It's a good idea since we can make the range in (0, 1] to avoid overflow while calculation.

Q1.2 Theory [2 points] Softmax can be written as a three step processes, with $s_i = e^{x_i}$, $S = \sum s_i$ and softmax(x_i)=$\frac{1}{S} S_i$.

Q1.2.1 As $x \in R^d$, what are the properties of softmax(x), namely what is the range of each element? What is the sum over all elements?

Ans: The range of each element is (0, 1] and the sum over all elements is 1.

Q1.2.2 One could say that ”softmax takes an arbitrary real valued vector x and turns it into a ____________”.

Ans: Softmax turns an arbitrary real valued vector x into a probability distribution.

Q1.2.3 Can you see the role of each step in the multi-step process now? Explain them.

Ans: The first step is to take the exponential operation and make the numbers into positive. The second one is to sum up for all the results from step1. The third one is to calculate individual result over the sum from step2 to get the probability for each term.

‘

Q1.3 Theory [2 points] Show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Solution:

Suppose we have a multi-layer neural network which has an input layer x with n units , a hidden layer z with m units and an output layer y with size l. We can from the following matrix expression between x and z with a weight matrix w_1 with size n x m and a bias term b_1:

$$z = w_1^T x + b_1$$

Similar, we can from the matrix expression between z and y with w_2 with size m x l and a bias term $b_2$as follow:

$$y = w_2^T z + b_2$$

And we can get the expression between x and y:

$$y = w_2^T (w_1^T x + b_1) + b_2 = w_2^T w_1^T x + w_2^T b_1 + b_2$$

If we now let $w_3^T = w_2^T w_1^T \; and \; b_3 = w_2^T b_1 + b_2$, we have:

$$y = w_3^T x + b_3$$

Therefore, even if we have more than three layers, that is, more than one hidden layer, we can still express the relationship between x and y using the similar form above which is the formula of linear regression. Hence, these two are equivalent if we do not have non-linear activation functions.

Q1.4 Theory [3 points] Given the sigmoid activation function σ(x) = $\frac{1}{1+e^{-x}}$, derive the gradient of the sigmoid function and show that it can be written as a function of σ(x) (without having access to x directly).

Solution:

$$\frac{d\sigma(\mathrm{x})}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{1+e^{-x}}\frac{1}{1+e^{-x}}$$

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

Q1.5

Solution:

First, we have:

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

And the derivatives with scalars would be:

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

According to the chain rule, we have:

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial W}$$

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial x}\frac{\partial x}{\partial W} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial x}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial b}\frac{\partial b}{\partial W} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial b}$$

And we can get the result:

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

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

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

Q1.6

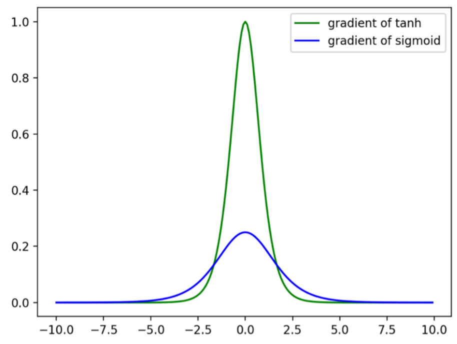
1. Assume that there are certain two layers h_1 *and* h_2. Then we can have the following relationship:

$$h_2(x) = \sigma(h_1(x))$$

$$\frac{dh_2(x)}{dx} = \sigma'(h_1(x))h_1'(x) = (1 - \sigma(h_1(x)))\sigma(h_1(x))h_1'(x)$$

Since we know the fact that each layer can be represented as the product of the following product and $(1 - \sigma(h_1(x)))\sigma(h_1(x)) \in (0,1)$, we can find that there will be the vanished gradient problem if there are many sigmoid as activation functions.

2. The output range of sigmoid function is (0,1) and the output range of tanh is (-1,1). The reason why prefer tanh is that the output of sigmoid function is always positive, but tanh is not. Also, tanh would tend to have larger gradient if the data is centered at zero.
3. The gradient of tanh has a larger value than gradient of sigmoid functions so it has less vanishing gradient problem.



4.

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

Q2.1.1 Theory [2 points] Why is it not a good idea to initialize a network with all zeros? If you imagine that every layer has weights and biases, what can a zero-initialized network output after training?

Solution:

If we initialize a network with all zeros, the results of forward calculation will be all zero. It's will cause the same gradient results while doing backward calculation which is not good since that will lead to the same updated values for our weights and biases.

Q2.1.3 Theory [1 points] Why do we initialize with random numbers? Why do we scale the initialization depending on layer size (see near Fig 6 in the paper)?

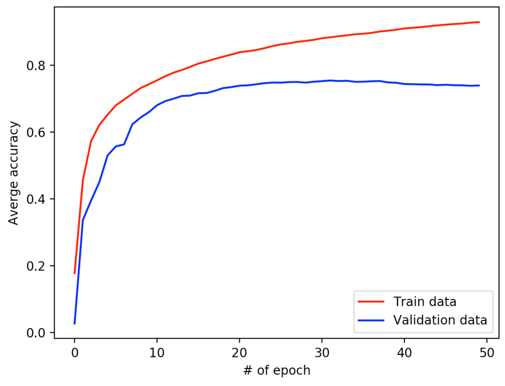
Solution:

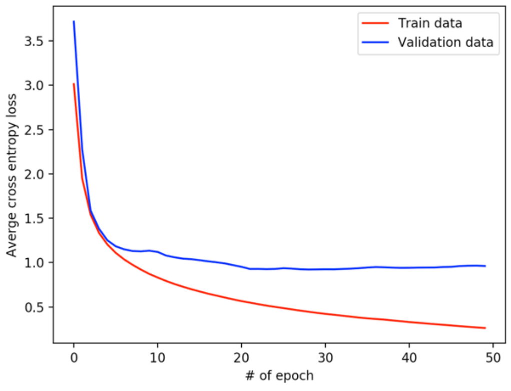
The reason why we initialize our parameters with random numbers is that it can give our parameters with different starting points which would have higher chance to achieve local minimum.

Why we scale the initialization depending on layer size is to make sure that we can get similar values for each layer. That is, we will not get very large or small values and it will help our model learn faster.

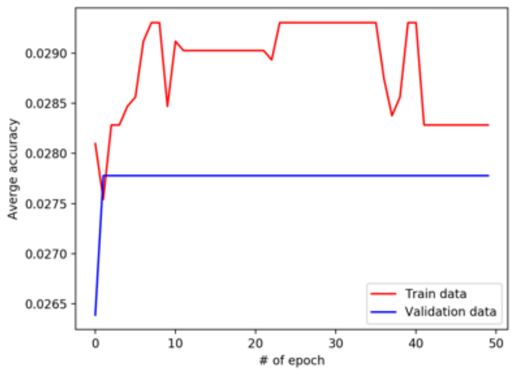
Q3.1.2 Writeup [2 points] Use your modified training script to train three networks, one with your best learning rate, one with 10 times that learning rate and one with one tenth that learning rate. Include all 4 plots in your writeup. Comment on how the learning rates affect the training, and report the final accuracy of the best network on the test set.

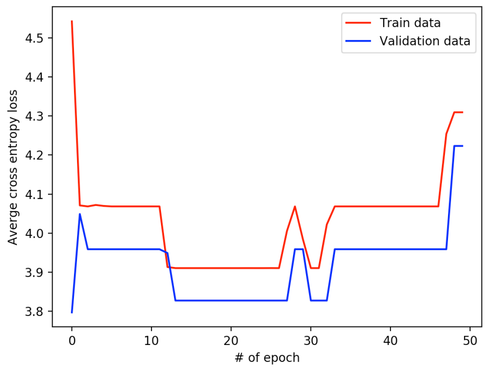
Solution: The best learning rate I choose is 0.01 with accuracies equal to 0.761 and 0.767 for the validation and testing dataset, respectively. The following figures show the average accuracy and cross entropy loss with three different learning rates. Generally, larger learning rate will worse result because the "step" updated gradient is too large to find a local minimum. On the other hand, smaller learning may converge to a local minimum but it converges slower than the best learning rate. Note that I calculated the average cross entropy loss by dividing the loss values by the total number of examples.



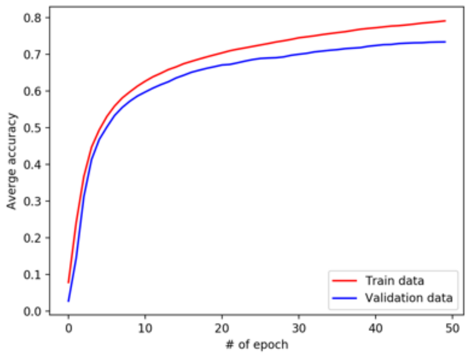


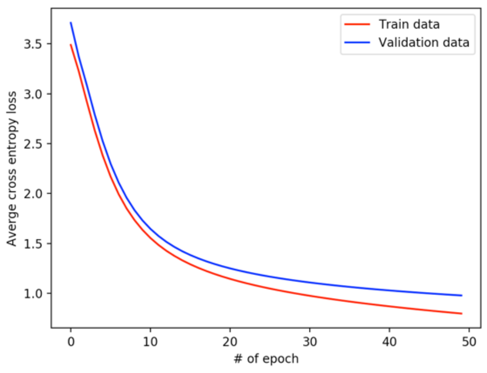
Average accuracy and cross entropy loss with learning rate = 0.01





Average accuracy and cross entropy loss with learning rate = 0.1

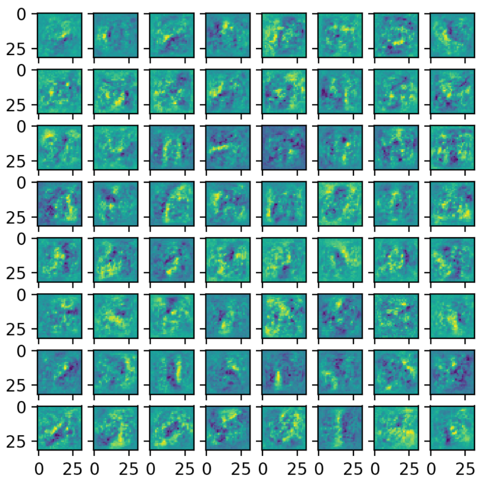




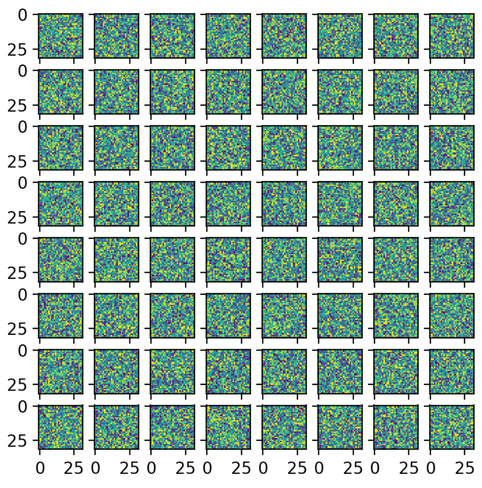
Average accuracy and cross entropy loss with learning rate = 0.001

Q3.1.3 Writeup [3 points] Visualize the first layer weights that your network learned (using reshape and ImageGrid). Compare these to the network weights immediately after initialization. Include both visualizations in your writeup. Comment on the learned weights. Do you notice any patterns?

Solution: The learned weights and weights immediately after initialization are illustrated as the following figures. If we observe on the learned weights, we can find there are several letter-like patterns while weights immediately after initialization are just random noise.



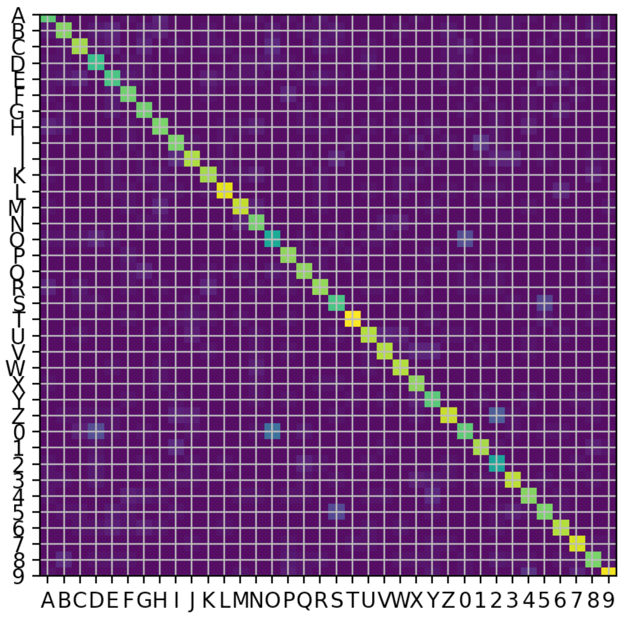
Learned weights



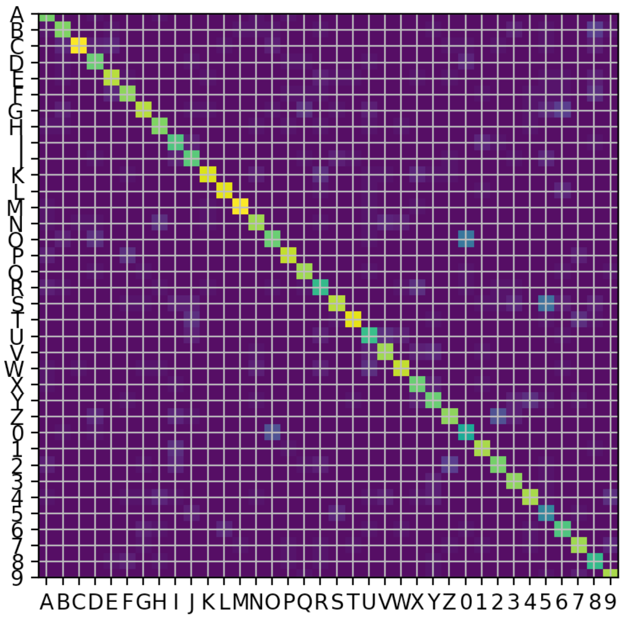
weights immediately after initialization

Q3.1.4 Writeup [2 points] Visualize the confusion matrix for your best model. Comment on the top few pairs of classes that are most commonly confused.

Solution: The confusion matrices of both validation and testing dataset are shown as below. The most pairs that are most commonly confused are {2:Z}, {5:S}, and {0:O} which has high similarity in their shapes.



Confusion matrix for validation dataset



Confusion matrix for testing dataset

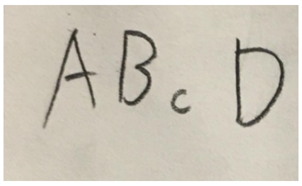
Q4.1 Theory [2 points] The method outlined above is pretty simplistic, and makes several assumptions. What are two big assumptions that the sample method makes. In your writeup, include two example images where you expect the character detection to fail (either miss valid letters, or respond to non-letters).

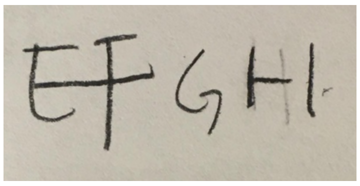
Solution:

The two big assumptions are:

1. All the letters are not connected with any other and all parts of a single letter are connected: only if it holds, then we can find the group of pixels
2. All the letters are in the similar size: only if it holds, then we can ignore small bounding boxes.

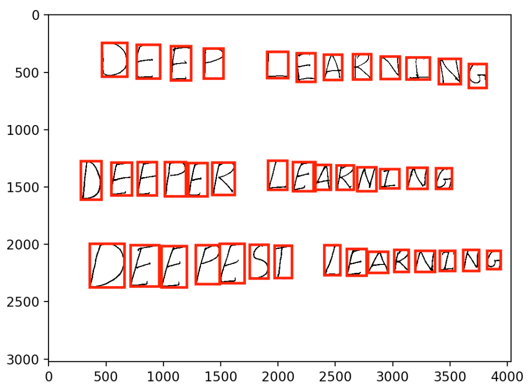
The following figures are some cases that may fail. The left figure has a letter 'c' which is much smaller than the others. The right figure has two connected letters 'E' and 'F'. Besides, the parts of letter 'H' are not all connected.

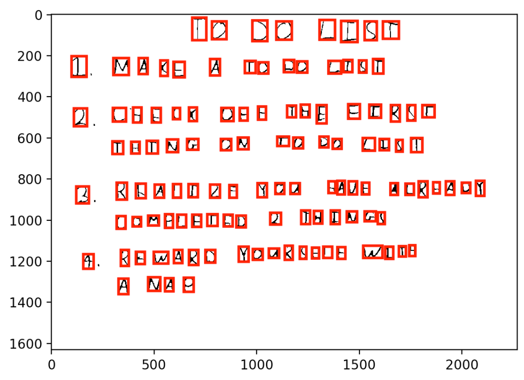


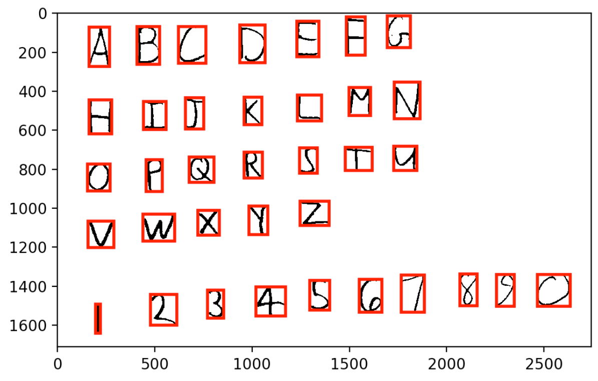


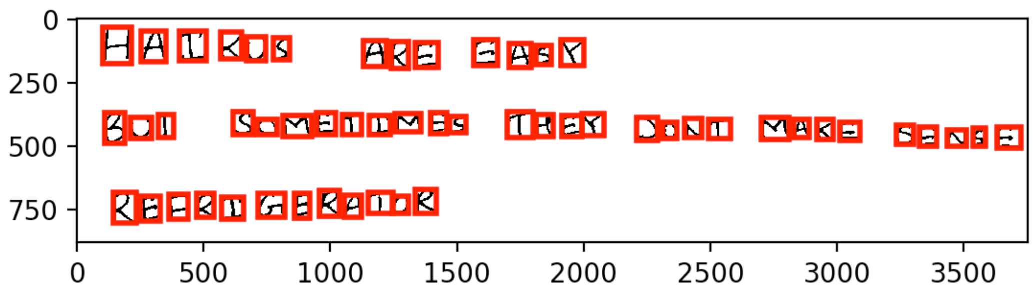
Q4.3 Writeup [3 points] Run findLetters(..) on all of the provided sample images in images/. Plot all of the located boxes on top of the image to show the accuracy of your findLetters(..) function. Include all the result images in your writeup.

Solution: The following figures are the located boxes found by findLetters().







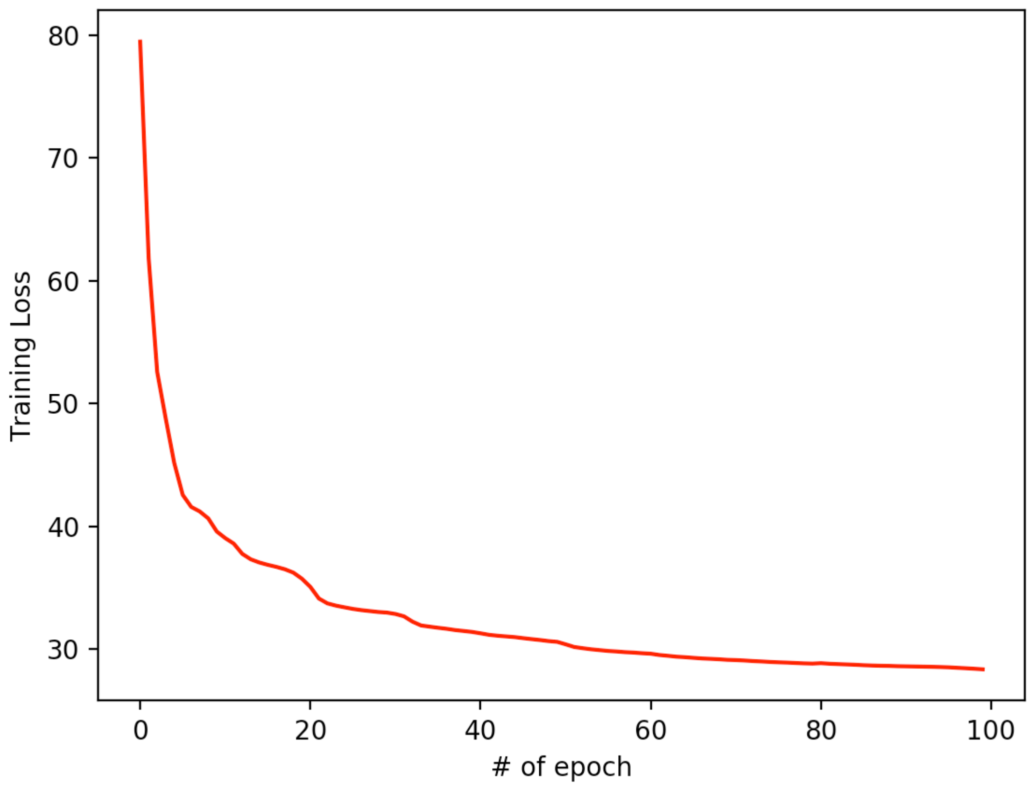


Q4.4 Code/Writeup [8 points] Run your run q4 on all of the provided sample images in images/. Include the extracted text in your writeup.

Figure	Output letters	Actual letters	Accuracy
01_list.jpg	TQ DQ LI5T I MAXE A TD DD LIST 2 LHFCK DFF THE FIRST THING QN TO DQ LIST 3 RIALIZE YQU HAVE RLRGADT CQMPLETLD J THINGI 4 R8W2RD YDURSBLF WITH A NAP	TO DO LIST 1 MAKE A TO DO LIST 2 CHECK OFF THE FIRST THING ON TO DO LIST 3 REALIZE YOU HAVE ALREADY COMLEPETED 2 THINGS 4 REWARD YOURSELF WITH A NAP	78.9%
02_letters.jpg	2BCDEFG HIIKLMN QPQR5TU VWXYZ 1Z3GSG789Q	ABCDEFG HIJKLMN OPQRSTU VWXYZ 1234567890	77.8%
03_haiku.jpg	HAIKUS ARG EAQY BWT SUMETZMEG THET DDWT MAKG BGNGE RBFRIGERATUR	HAIKUS ARE EASY BUT SOMETIMES THEY DONT MAKE SENSE REFRIGERATOR	73.2%
04_deep.jpg	DEEF LEARMING DEBPER LEARXING DECPE5I LEARNING	DEEP LEARNING DEEPER LEARING DEEPEST LEARING	82.1%

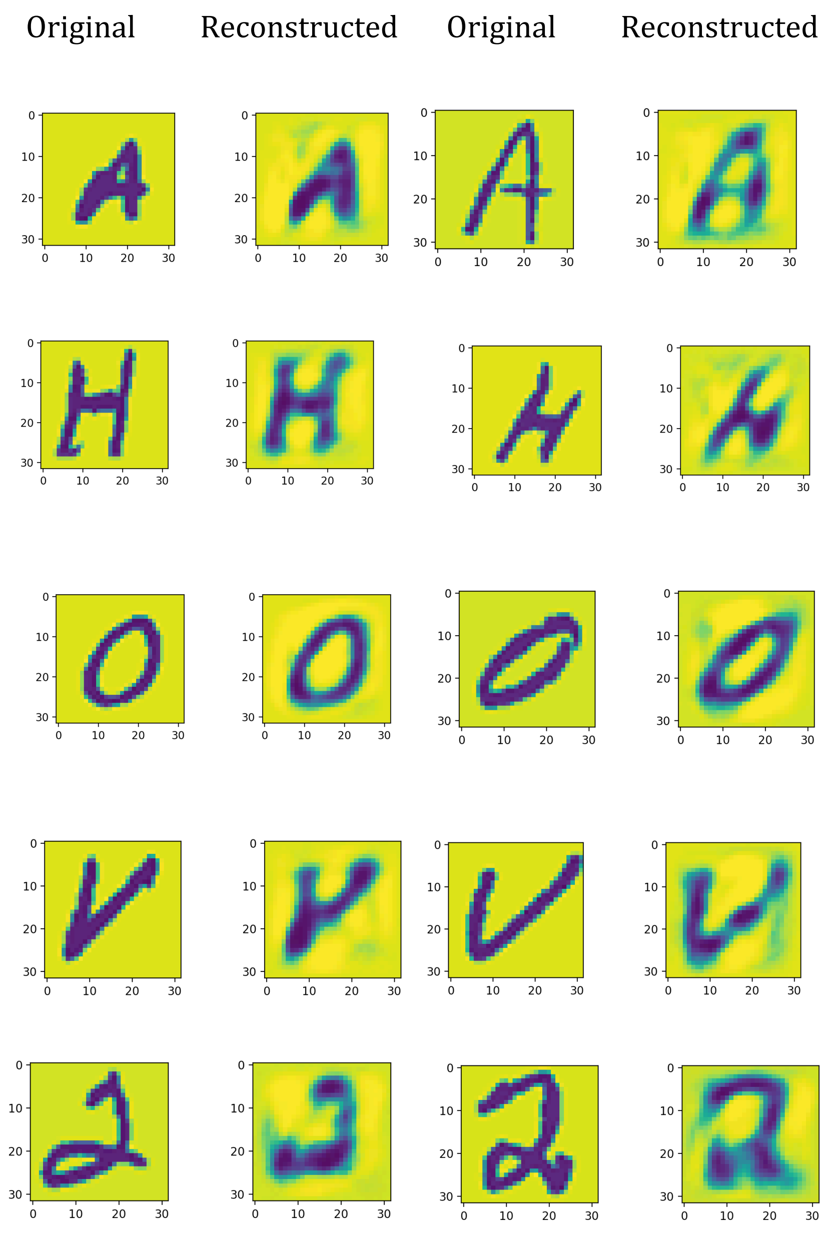
Q5.2 Writeup/Code [2 points] Using the provided default settings, train the network for 100 epochs. What do you observe in the plotted training loss curve as it progresses?

Solution: The plotted training loss curve is shown as below. As the train process goes, the training loss decreases dramatically at the beginning and finally converges to a local minimum.



Q5.3.1 Writeup/Code [2 points] Now let's evaluate how well the autoencoder has been trained. Select 5 classes from the total 36 classes in the validation set and for each selected class include in your report 2 validation images and their reconstruction. What differences do you observe that exist in the reconstructed validation images, compared to the original ones?

Solution: The following figures show the chosen 5 class. The left figures in the pairs are the validation images and the right figures are the reconstructed images. The reconstructed ones are blurred and also noisy or lose some portion of the original letters.



Q5.3.2 Writeup [2 points] Report the average PSNR you get from the autoencoder across all images in the validation set.

Solution: The average PSNR I get from the autoencoder across all images in the validation set is 15.75.

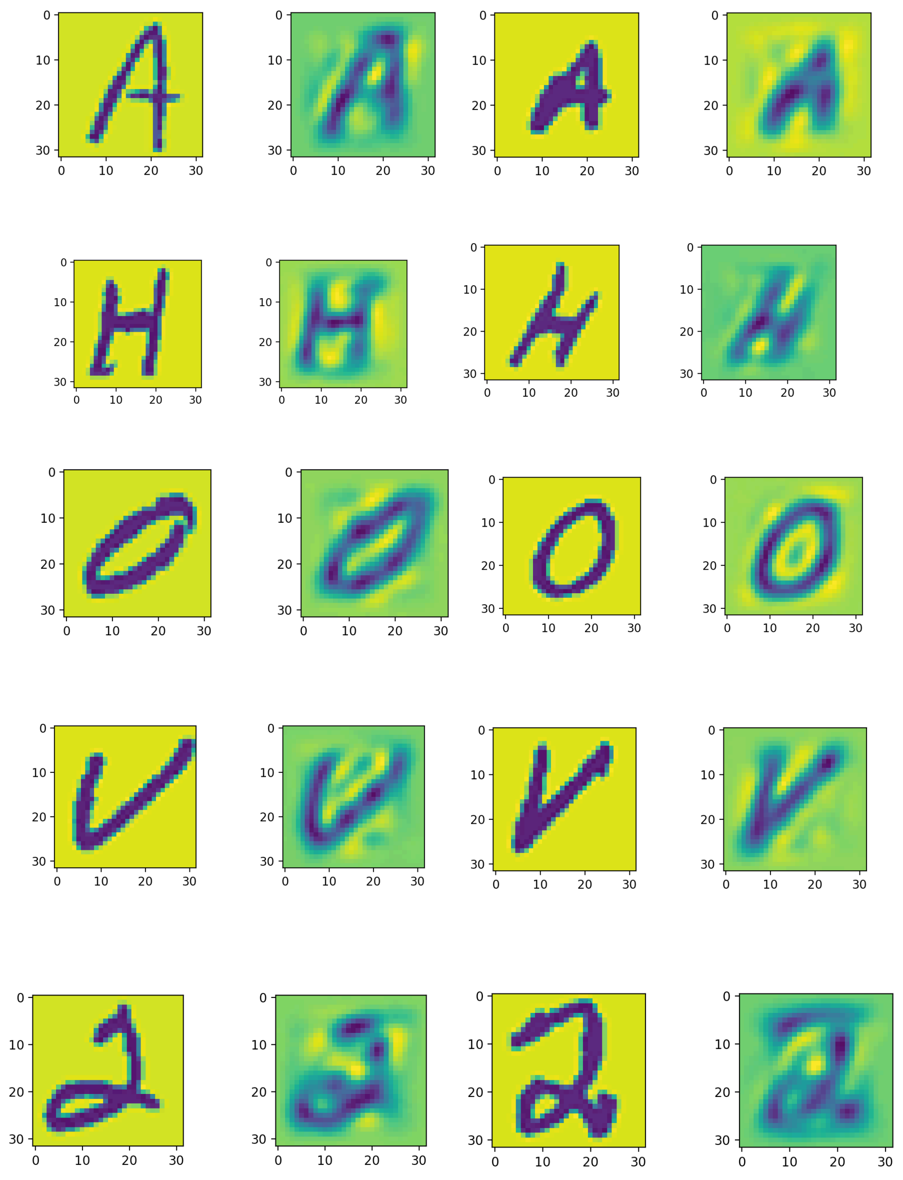
Q6.1 Writeup [2 points] What is the size of your projection matrix? What is its rank?

Solution: Since we pick the first 32 principal elements, the size of the projection matrix is 32x1024 and its rank is 32.

Q6.2 Writeup/Code [6 points] Use the classes you selected in Q5.3.1, and for each of these 5 classes, include in your report 2 validation images and their reconstruction. You may use validation labels to help you find the corresponding classes. What differences do you observe that exist in the reconstructed validation images, compared to the original ones? Also, report the average PSNR on the whole validation set with PCA.

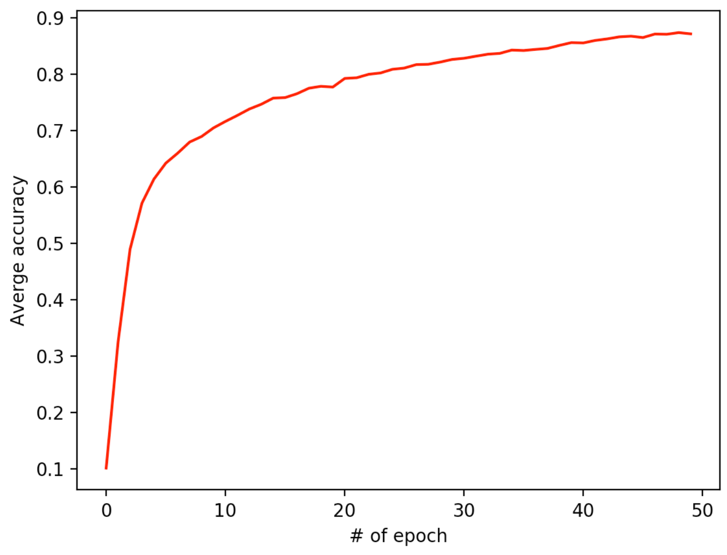
Solution: The following figures show the chosen 5 class. The left figures in the pairs are the validation images and the right figures are the reconstructed images. The reconstructed ones are blurred and also noisy nearing the letters or lose some portion of the original letters. The average PSNR is 16.35.

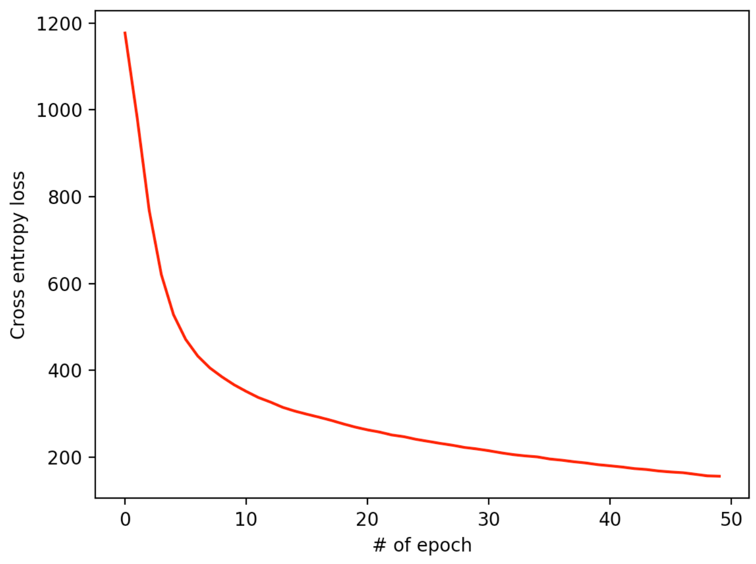
Original Reconstructed Original Reconstructed



Q7.1.1 Code/Writeup [5 points] Re-write and re-train your fully-connected network on the included NIST36 in PyTorch. Plot training accuracy and loss over time.

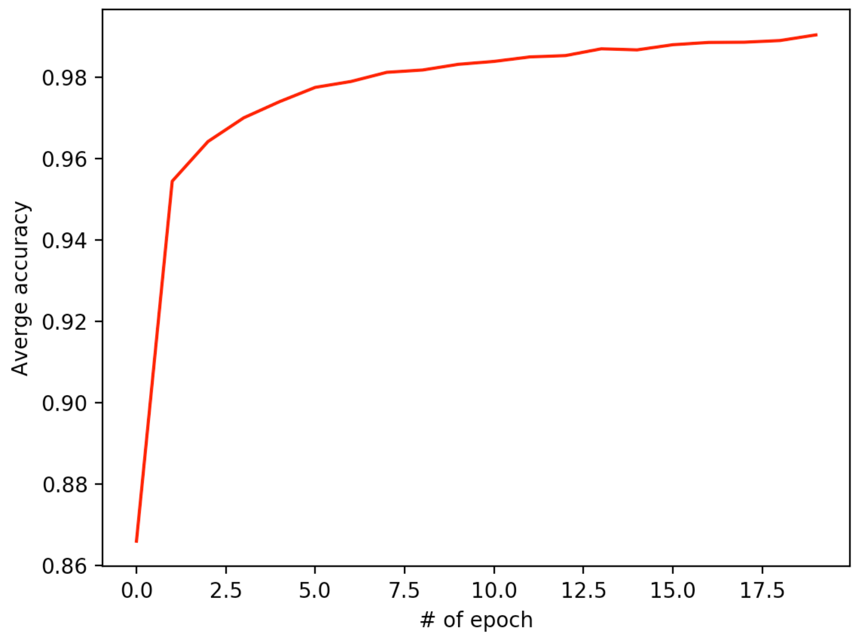
Solution: The following figures show the accuracy and cross entropy loss for training and note that the accuracy on testing is 78.1%.

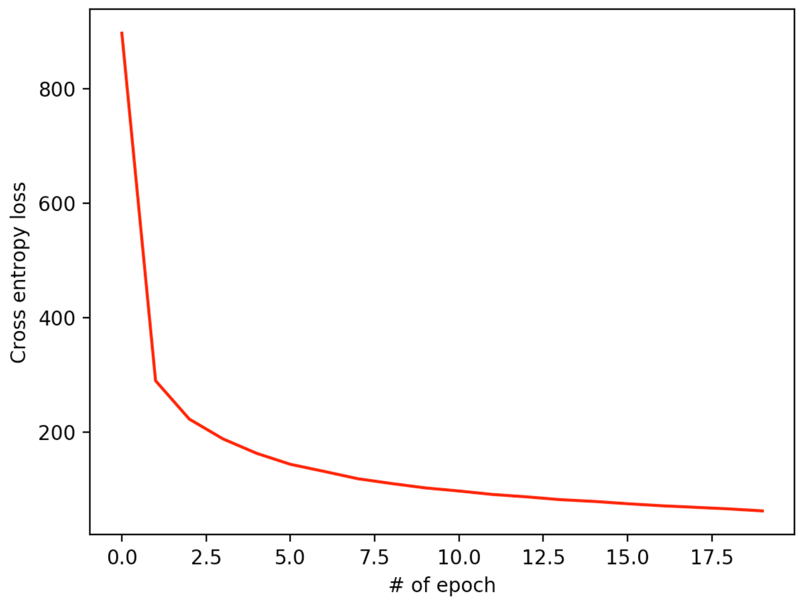




Q7.1.2 Code/Writeup [2 points] Train a convolutional neural network with PyTorch on MNIST (available in torchvision.datasets). Plot training accuracy and loss over time.

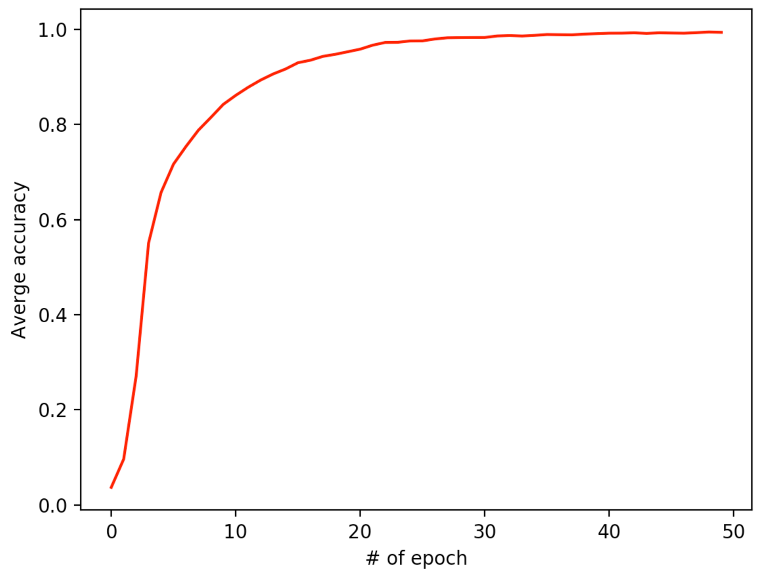
Solution: The following images shows the accuracy and the cross entropy loss in the training phase. Note that the accuracy on test dataset is 98.3%.

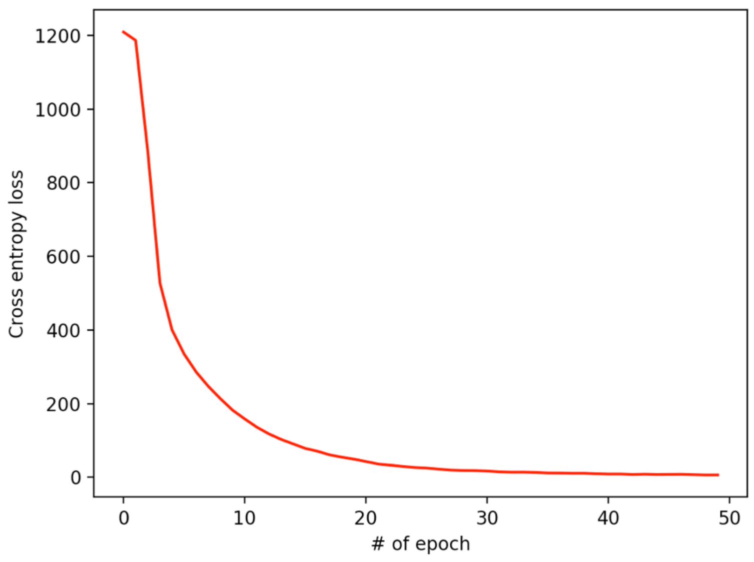




7.1.3 Code/Writeup [3 points] Train a convolutional neural network with PyTorch on the included NIST36 dataset.

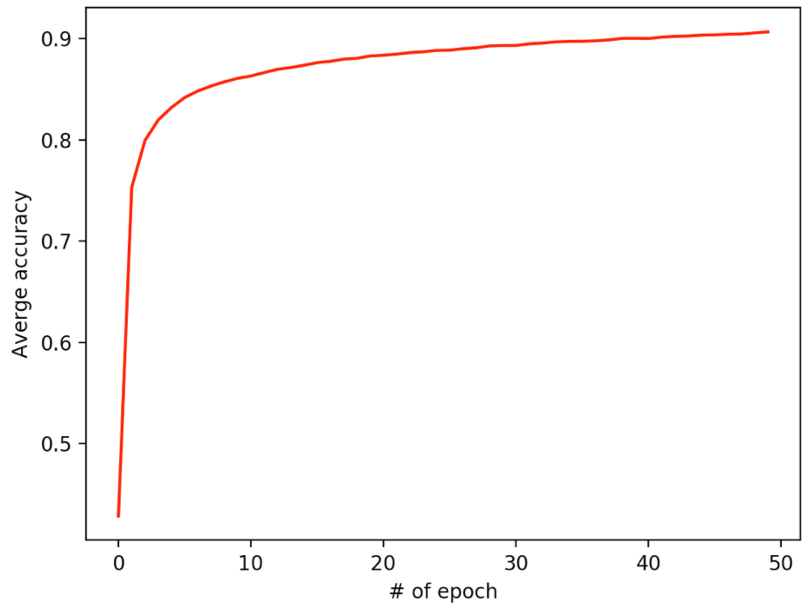
Solution: The following images shows the accuracy and the cross entropy loss in the training phase. Note that the accuracy on test dataset is 88.4% and I use the accumulation of the cross entropy function as the output cross entropy loss.

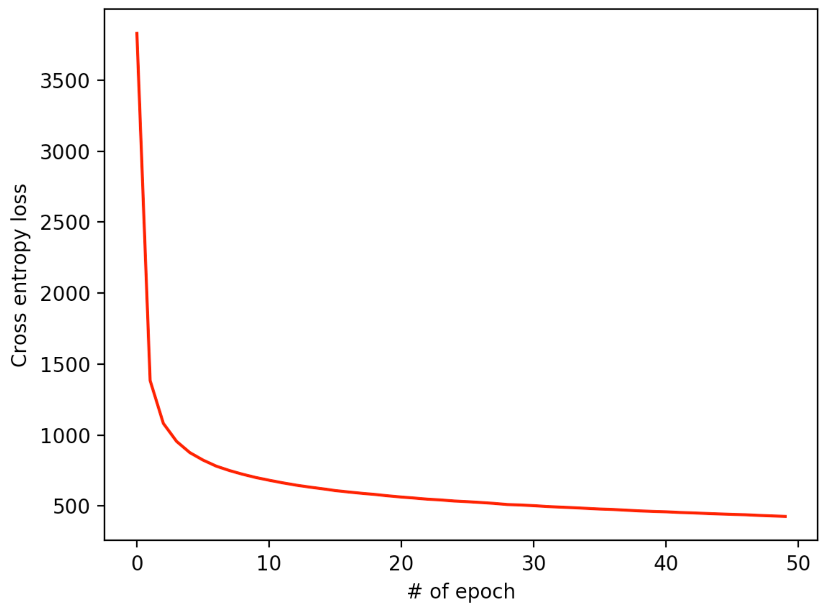




Q7.1.4 Code/Writeup [10 points] Train a convolutional neural network with PyTorch on the EMNIST Balanced (available in torchvision.datasets, use balanced split) dataset and evaluate it on the findLetters bounded boxes from the images folder.

Solution: The following images shows the accuracy and the cross entropy loss in the training phase. Note that the accuracy on test dataset is 86.6% and I use the accumulation of the cross entropy function as the output cross entropy loss.



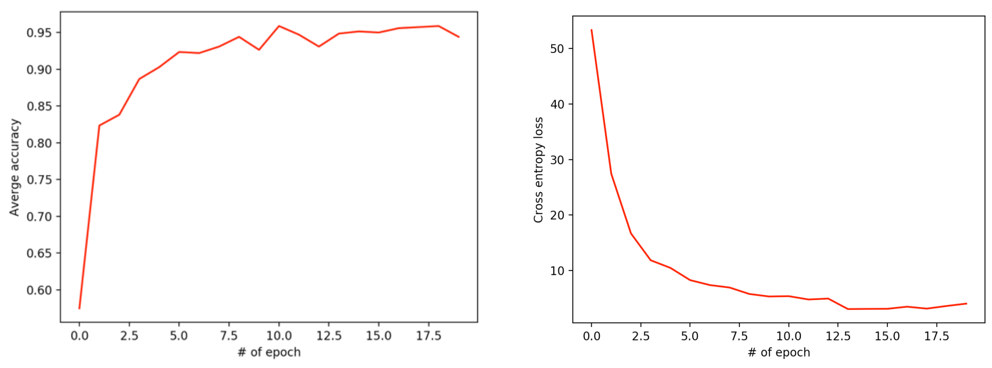


Also, the table shows the evaluation on the findLetters bounded boxes from the images folder.

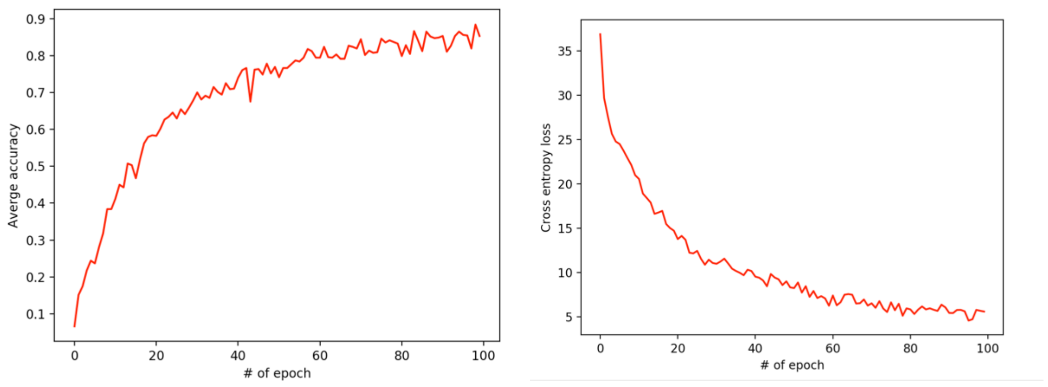
Figure	Output letters	Actual letters	Accuracy
01_list.jpg	T0 D0 LI5T 1 MAKE A TD D0 LIST 2 CHECK 0FF THE FIRST THING ON T0 D0 LI5T 3 RRALIZF Y0U HAVE ALREADY C0MPLETID 2 T4IWGF 4 REWARD Y0URSELe WITH A NAP	TO DO LIST 1 MAKE A TO DO LIST 2 CHECK OFF THE FIRST THING ON TO DO LIST 3 REALIZE YOU HAVE ALREADY COMLEPETED 2 THINGS 4 REWARD YOURSELF WITH A NAP	85.7%
02_letters.jpg	ABCDEFG HIJKLMN 0PQRSTU VWXYZ I2345678f0	ABCDEFG HIJKLMN OPQRSTU VWXYZ 1234567890	94.4%
03_haiku.jpg	HAIKU5 ARE EA5X BUT SQMETIME5 TKEY DcNT MAKG SEN5E REFRIGERAT0R	HAIKUS ARE EASY BUT SOMETIMES THEY DONT MAKE SENSE REFRIGERATOR	82.1%
04_deep.jpg	JEEP LEARNING DEEPER LEARNING JEEPE5T LEARNING	DEEP LEARNING DEEPER LEARING DEEPEST LEARING	92.3%

Q7.2.1 Code/Writeup [5 points] Fine-tune a single layer classifier using pytorch on the flowers 17 (or flowers 102!) dataset using squeezenet1 1, as well as an architecture you've designed yourself (3 conv layers, followed 2 fc layers, it's standard slide 6) and trained from scratch. How do they compare?

Solution: The validation and test accuracies of fine-tune model are 94.4% and 93.5%, respectively. The validation and test accuracies of the custom CNN are 71.8% and 73.8%, respectively. The fine-tune model converges in only 20 epoch (10 for training the single layer classifier and 10 for train all parameters in the model) and yields better accuracy than the one from the scratch. Note that I use the accumulation of the cross entropy function as the output cross entropy loss.



Accuracy and cross entropy loss of squeezenet1_1 with a single layer classifier



Accuracy and cross entropy loss of custom CNN