Homework 5

Q 1.1.1 “Theory” [5 points] In training deep networks, the ReLU activation function is generally preferred to the sigmoid activation function. Why might this be the case?

1. The computation of ReLU is less and easier than sigmoid function.
2. ReLu will cause some of the output zeroes, which in turn prevent the dataset from overfitting
3. Using sigmoid function, the gradient will be zeroes when reaching infinity. In this case, the gradient might “disappear” or lose information in back propagation.

Q1.1.2 Theory [5 points] All types of deep networks use non-linear activation functions for their hidden layers. Suppose we have a neural network with input dimension N and output dimension C and T hidden layers. Prove that if we have a linear activation function g, then the number of hidden layers has no effect on the representation capability of the network (i.e., that the set of functions that can be represented by a T layer network is exactly the same as the set that can be represented by a $T' \neq T$ layer network).

If the activation function is a linear one, we can assume it as:

$$g(x) = kx$$

For the pre-activation of first layer:

$$a^{(1)}(x) = w^{(1)}x + b^{(1)}$$

For the post-activation of the first hidden layer(second layer including input layer):

$$\hbar^{1}(x) = g\left(a^{(1)}(x)\right) = k(w^{(1)}x + b^{(1)})$$

For the post-activation of the nth hidden layer:

$$\hbar^{n}(x) = g\left(a^{(n)}(x)\right) = k(w^{(n)}\hbar^{n-1} + b^{(n)})$$

Thus, as a result, the nth layer value is just the linear combination of the first input value no matter how many hidden layers it has, in this case linear function has no effect on the representation capability of the network.

Q2.1.1 Theory [5 points] Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value? (Hint: Consider what the gradients from backpropagation will look like.)

If we set initialization as all zeros or same constant, the hidden units will become the same for each input unit. Therefore, when we are trying to update the gradient using backpropagation, all of the weights and bias we wish to figure out will be the same. As a result, the parameters per unit what we l learn are only few, and are not enough but redundant.

Q2.1.3 Writeup [5 points] Describe the initialization you implemented in Q2.1.2 and any reasoning behind why you chose that strategy.

To prevent from causing learning failure, I randomly initialize the weights and bias instead of setting all zeros, using W=0.01*rand(unitNum2,unitNum1)/sqrt(unitNum1), where unitNum1 is the number of elements of input units, and unitNum2 is the number of output layer units. I also use variances calibration with 1/sqrt(N), so that I can scale the output into a limited range without accumulating them after continuous layers’ computation.

Q2.4.1 Theory [5 points] Give pros and cons for both stochastic and batch gradient descent. In general, which one is faster to train in terms of number of epochs? Which one is faster in terms of number of iterations?

Stochastic:

Pros: 1. Much faster, it only computes gradient one time for each training sample, so the number of epochs equals to the number of training samples. 2. It possibly can skip the shallow local minimum.

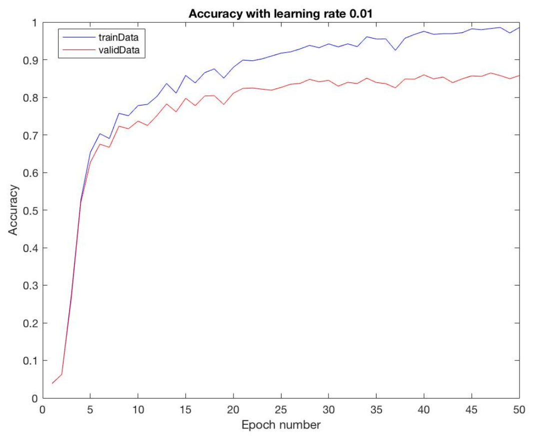
Cons: 1. The random training sample is noisy resulting into learning failure

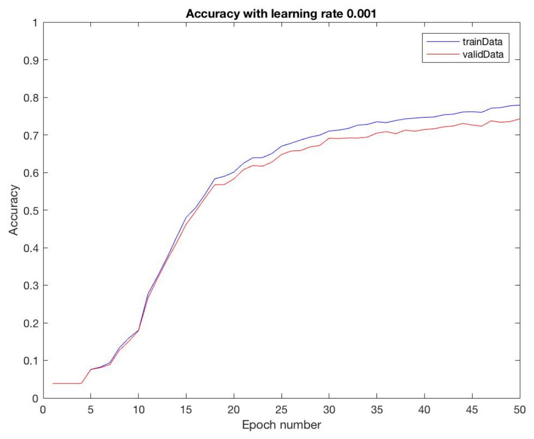
Batch gradient:

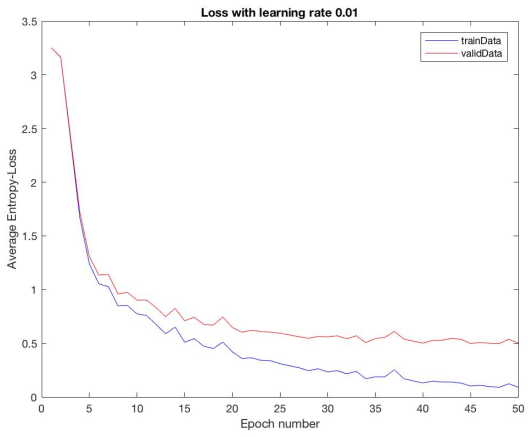
Pros: 1. Stable and accurate. It can find the local minimum accurately with an appropriate learning rate.

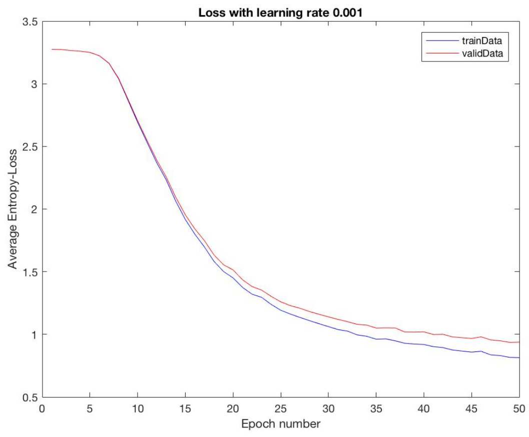
Cons: 1. For one updating, gradient needs to be calculated using all dataset. So the number of epochs is much larger than stochastic.

Q3.1.2 Writeup [5 points] Use your modified training script to train two networks, one with learning rate 0.01, and another with learning rate 0.001. 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.







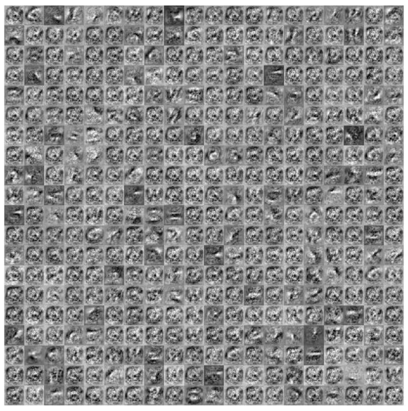


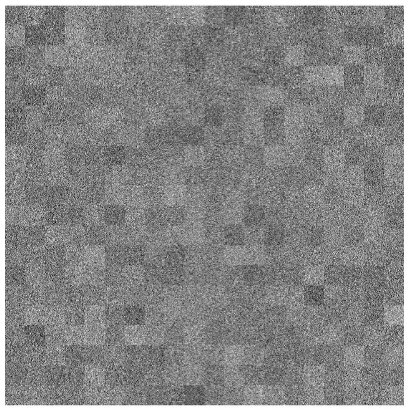
Using a smaller learning rate makes learning speed slower. As accuracy with learning rate:0.001 shown on the top right, the final accuracy is about 75%, after 50 epochs learning. However, larger learning rate: 0.01 allows a much faster learning speed, as shown on the top left. The final accuracy is about 95%.

Additionally, accuracy curve with smaller learning rate is smoother than with larger one. That’s because smaller learning rate creates slight gradient descent per epoch.

Q3.1.3 Writeup [5 points] Using the best network from the previous question, report the accuracy and cross-entropy loss on the test set, and visualize the first layer weights that your network learned (using reshape and montage). 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?

The accuracy of test set with best learning network is 88.15%, loss is 0.4860.

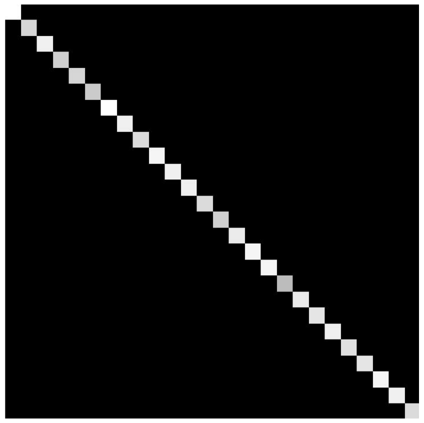




The initial weights are random distribution, so they seem like noises without any regular pattern.

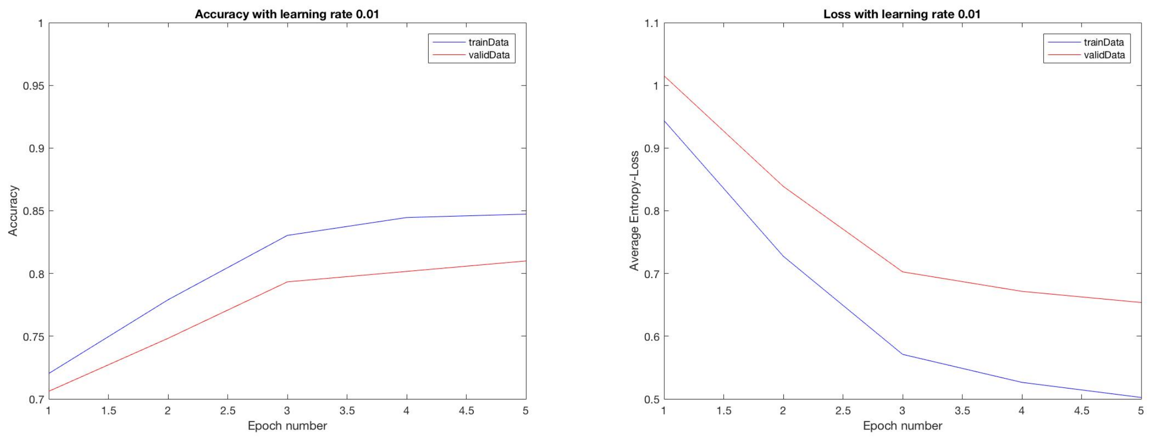
The best weights have the similar edge, circle or point-like features.

Q3.1.4 Writeup [5 points] Visualize the confusion matrix for your best model as a 26 × 26 image (upscale the image so we can actually see it). Comment on the top two pairs of classes that are most commonly confused.



In the test prediction result, letter R is the most confused one with several letters, like A, B, P. These capital letters have a similar feature, which have a small circle as a part. This might be the reason why R is classified into different other classes and has a low accuracy.

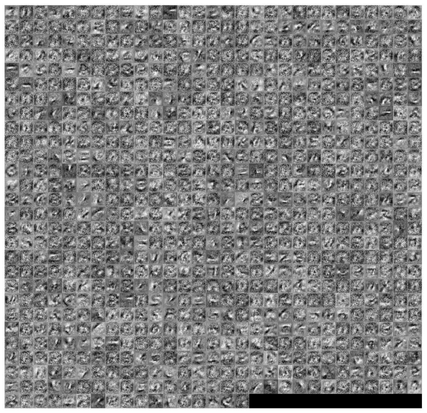
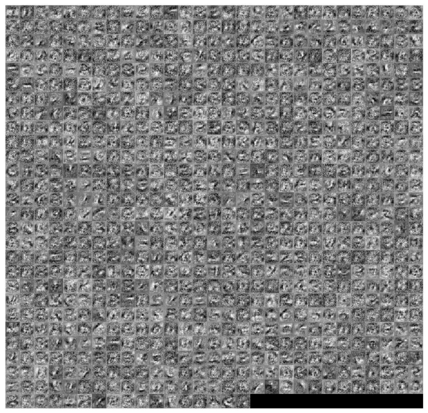
Q3.2.1 Code/Writeup [10 points] Make a copy of train26.m and name it finetune36.m. Modify this script to load the data from nist36 *.mat, and train a network to classify both written letters and numbers. Finetune (train) this network for 5 epochs with learning rate 0.01, and include plots of the accuracy and cross-entropy loss in your writeup.



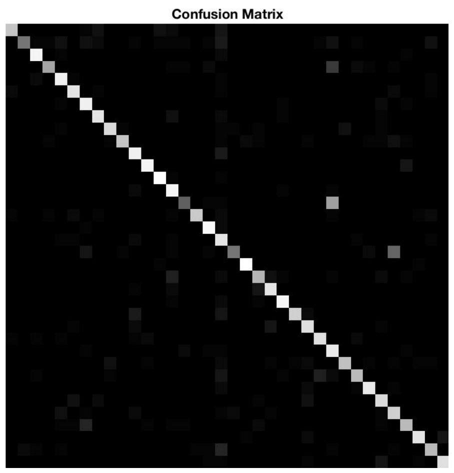
Q3.2.2 Writeup [5 points] Once again, visualize the network’ s first layer weights before and after training. Comment on the differences you see. Also report the network’ s accuracy and loss on the test set.

The accuracy of test set is 80.06%, and the cross-entropy loss is 0.6485 after 5 epochs;

The left one is the initial weights initialization of the first layer, and the right one is the best weights initialization of the first layer. It is not surprised to find that there is no big difference between them, because the initial weights are transplanted from an already well trained network. Thus the initial weights have already acquired some of the recognition features.

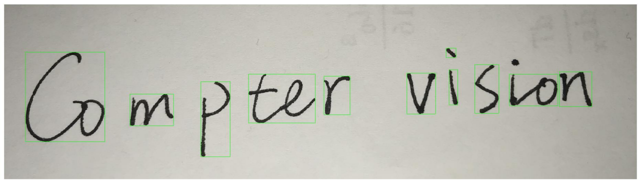
Q3.2.3 Writeup [5 points] Visualize the confusion matrix for your best model as a 36 $\times$ 36 image (upscale the image so we can actually see it). Comment on the top two pairs of classes that are most commonly confused. How has introducing more classes affected the network?



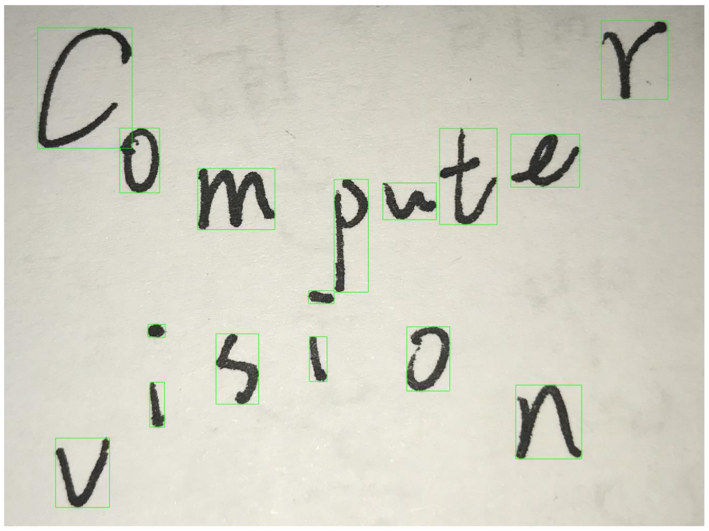
The top two classes that are confused are B, 0 and O, as we discussed above. These letters have a similar property, having a circle as a part. As a result, the network might classify them into a wrong class. When introduce digit number classes into original network, the total accuracy decreases a little bit, while the loss increases.

Q4.1 Theory [5 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).

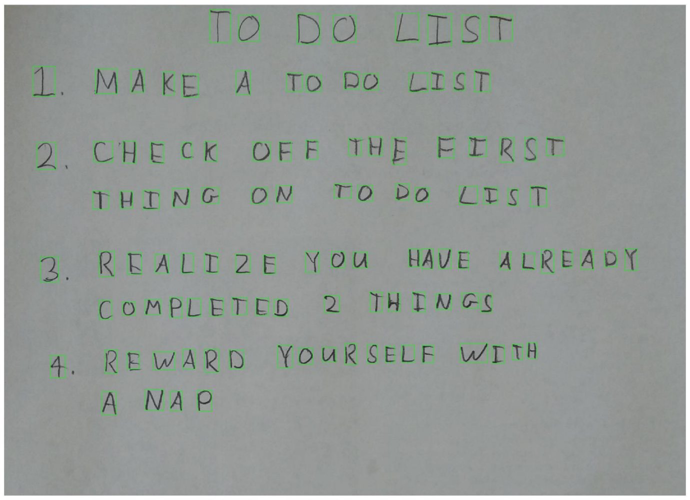
There are two basic assumptions. The first one is that letters are required to be separately hand-written with each other. If they have connection between letters, bounding box can hardly extract the text.



The second one is to assume the letters are obviously arranged in a line. Otherwise, it’ s hard to order them according lines neither the letter number along that line.



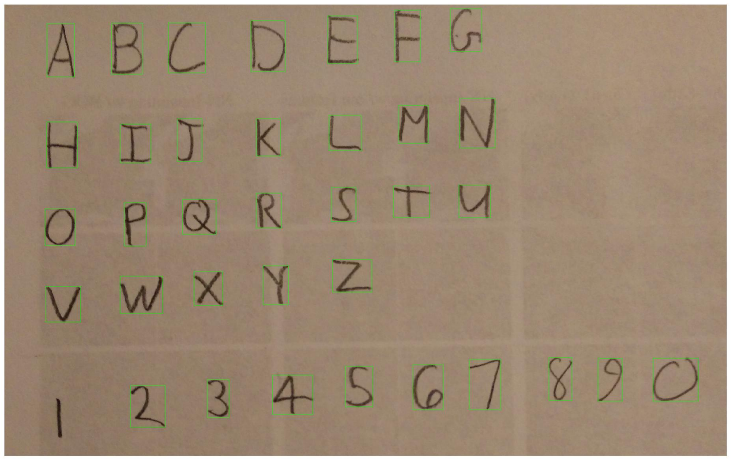
Q4.3 Writeup [5 points] Run findLetters(..) on all of the provided sample images in images/. Plot all of the located boxes on top of the image to show the accuracy of your findLetters(..) function. Include all the result images in your writeup.


TO DO LIST

1. MAKE A TO DO LIST
2. CHECK OFF THE FIRST THING ON TO DO LIST
3. REALIZE YOU HAVE ALREADY COMPLETED 2 THINGS
4. REWARD YOURSELF WITH A NAP

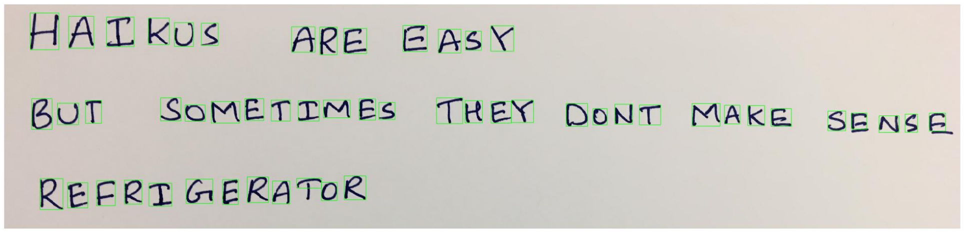
total number: 115 letters

correct extraction: 115 letters



total number: 36 characters

correct number: 35 characters

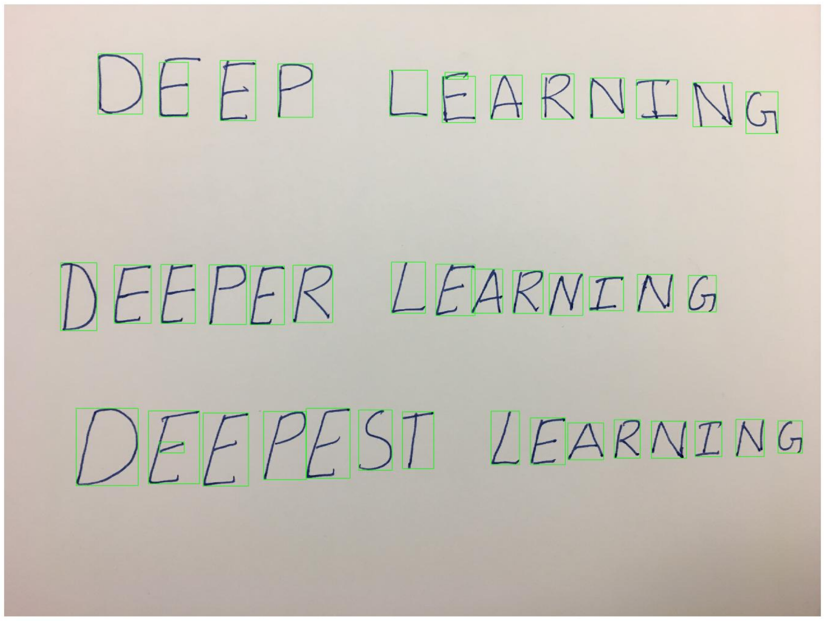

HAIKUS ARE EASY

BUT SOMETIMES THEY DONT MAKE SENSE

REFRIGERATOR

total number: 54 characters

correct number: 53 characters



total number: 41 characters

correct number: 39 characters

Conclusion, the accuracy of extraction is 98.37%.

Q4.5 Writeup [5 points] Run your extractImageText(..) on all of the provided sample images in images/. Include the extracted text in your writeup.

Img1:

line1='FQ JQ LIFT';

line2='I NAKE A TQ QQ LIST';

line3='L LHLEK QFE THE FIR5T'

line4='THING QN TQ 6Q LI5T'

line5='3 RFALILF YQU HAVE RLRLA6T'

line6='LQMPLFILD J THINGQ'

line7='9 RFWQRD YQUR SELF WITH'

line8='A NAP'

02_letters.jpg:

line1='HAIKU QAR EEAGY';

line2='EUT SQME TIMES TREY DDNT MAKG SGNGG';

line3='KBFRIGERAMQK'

03_haiku.jpg

line1='F E L J E F G';

line2='H I F K L K N';

line3='Q P Q K 5 T U';

line4='V W X Y Z';

line5='I 3 G 5 G 7 5 Y 5';

04_deep.jpg

line1='LEFF LLMAKN';

line2='0EFFFK LEIKNING';

line3='5FHFFF5F FEARNING';