Homework 5

Kai Li from MRSD

Q 1.1.1

Q 1.1.1 : The benefits of ReLU:

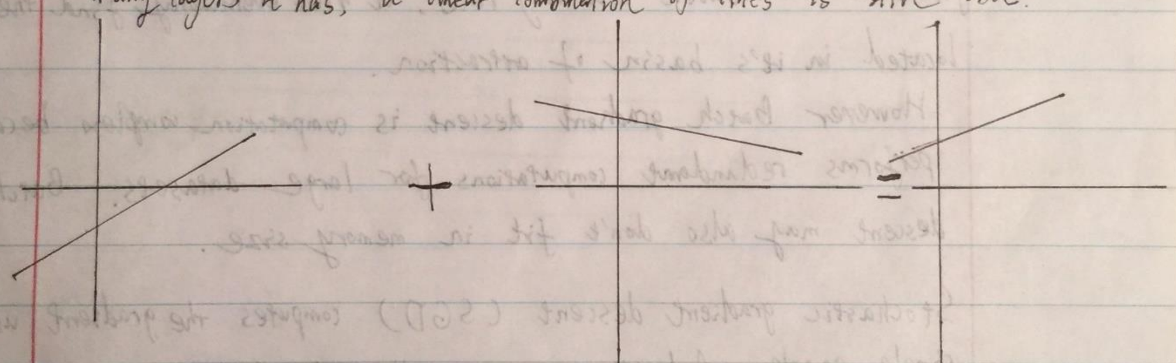
It induces the sparsity in the hidden units when $a \le 0$.

ReLU doesn't face gradient vanishing problem as sigmoid function. The gradient of the sigmoid function vanishes as we increase or decrease α. However, the gradient of the ReL function doesn't vanish as we increase x.

The constant gradient of ReLUs result in faster learning,

Q 1.1.2

Q 1.1.2: Without a nonlinear activation function, the neural network is calculating linear combinations of linear functions. And no matter how many layers it has, a linear combination of lines is still line.



If all the activation functions are lines, we can get the specific line using arbitrary number of layers. That is to say the number of hidden layers has no effect on the actual network.

Q 2.1.1

Q 2.1.1 : First neural networks tend to get stuck in local minima.
Second if the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another.
Third, if every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. There is no source of asymmetry between neurons if their weights are initialized to be the same.

Q 2.1.2

Please refer to InitializeNetwork.m for detailed implementation.

Q 2.1.3

Q 2.1.3 The way I implemented in Q 2.1.2 is make initial W follow Uniform (-0.5, 0.5) distribution and make initial b to be all zero.

The reason is that I want the initial parameters all to be near zero but without bias and symmetry, in this way, sigmoid function will be significant and the training will be good and relatively fast.

Q 2.2.1

Please refer to Forward.m for detailed implementation. Note that X is a vector of dimensions N × 1 as required.

Q 2.2.2

Please refer to Classify.m for detailed implementation.

Q 2.2.3

Please refer to ComputeAccuracyAndLoss.m for detailed implementation.

Q 2.3.1

Please refer to Backward.m for detailed implementation.

Q 2.3.2

Please refer to UpdateParameters.m for detailed implementation.

Q 2.4.1

Q2.4.1 Batch gradient descent is great for convex, or relatively smooth error manifolds, it moves directly towards an optimum solution, also given an annealed learning rate, it will eventually find the minimum located in it's basin of attraction.

However Batch gradient descent is computation complex because it performs redundant computations for large datasets. Batch gradient descent may also don't fit in memory size.

Stochastic gradient descent (SGD) computes the gradient using a single sample. And it's computationally faster, this computational advantage is leveraged by performing many more iterations of SGD.

However, SGD on the other hand, chooses a random point and takes the steepest route towards this point. In this case, the somewhat noisier gradient calculated using the reduced number of samples tends to jerk the model out of local minima.

SGD is faster to train in terms of number of epoches.

Batch Gradient Descent is faster to train in terms of number of iterations.

Q 2.4.2

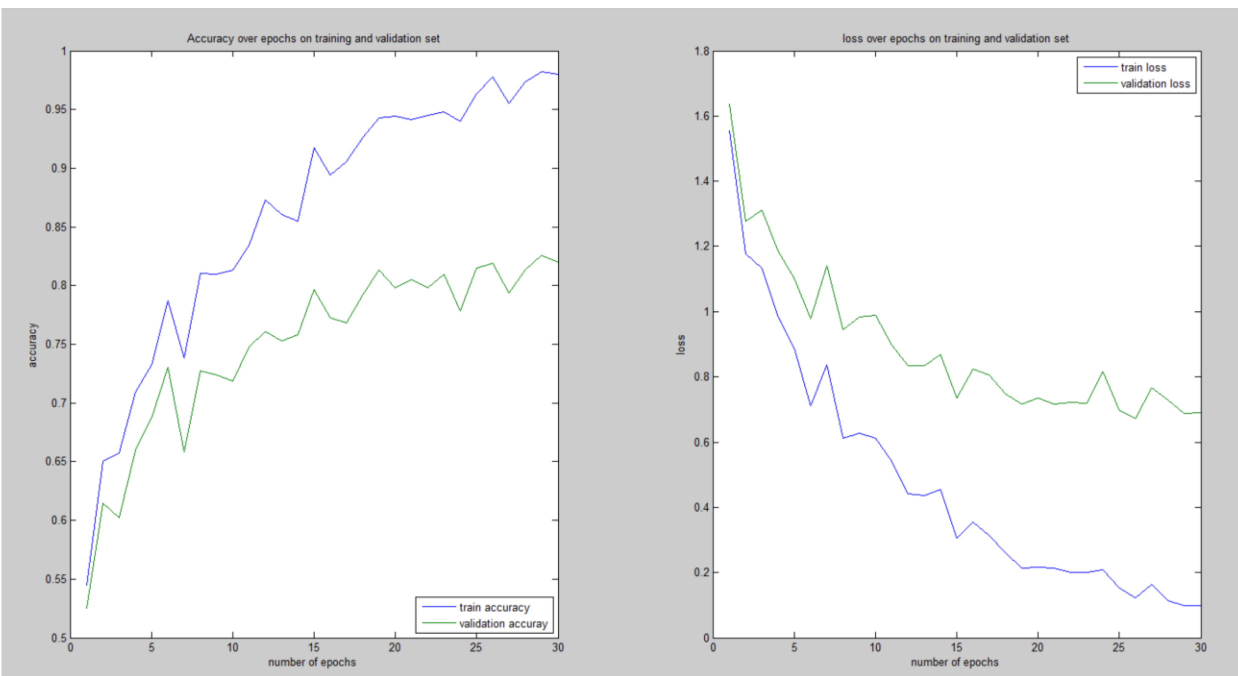
Please refer to Train.m for detailed implementation.

Q 2.5

Please refer to the script named “checkGradient”. Here I randomly picked one W(i,j) in each layer. The outputs are the error terms which all should near 0.

Q 3.1.1

Please refer to train26.m for more details. And the following plots are the accuracy and loss on training validation set over epochs.

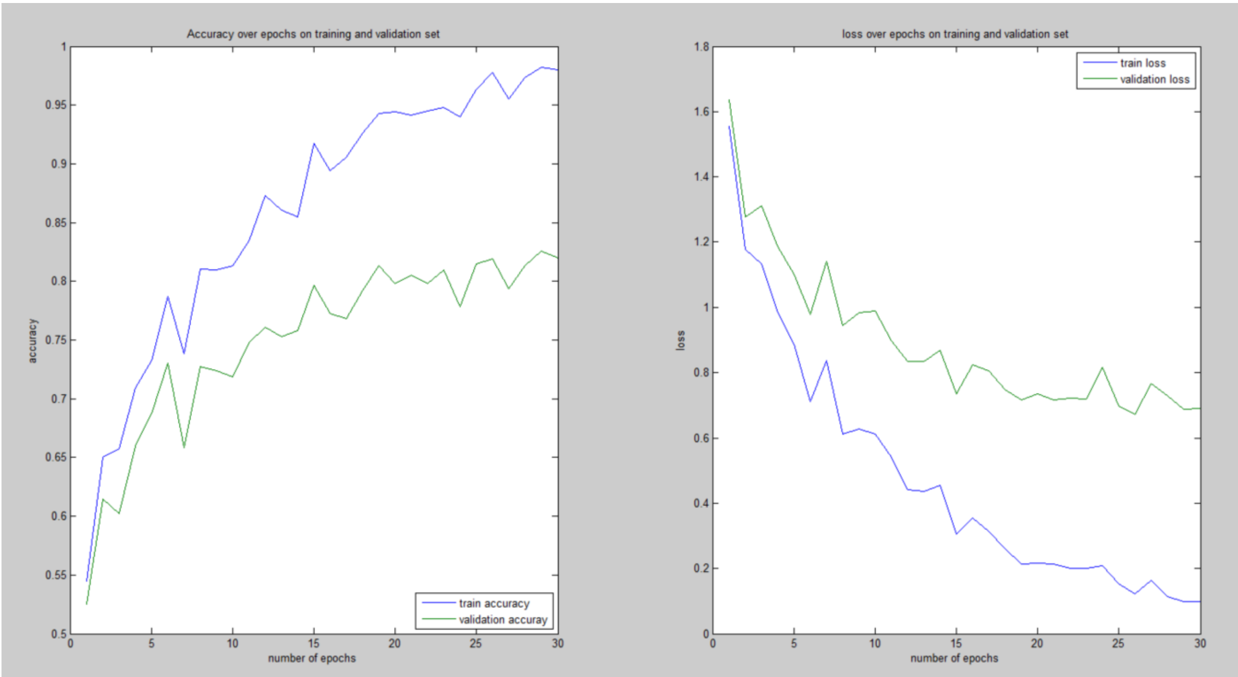


The last epoch’s accuracy and loss on training and validation set are:

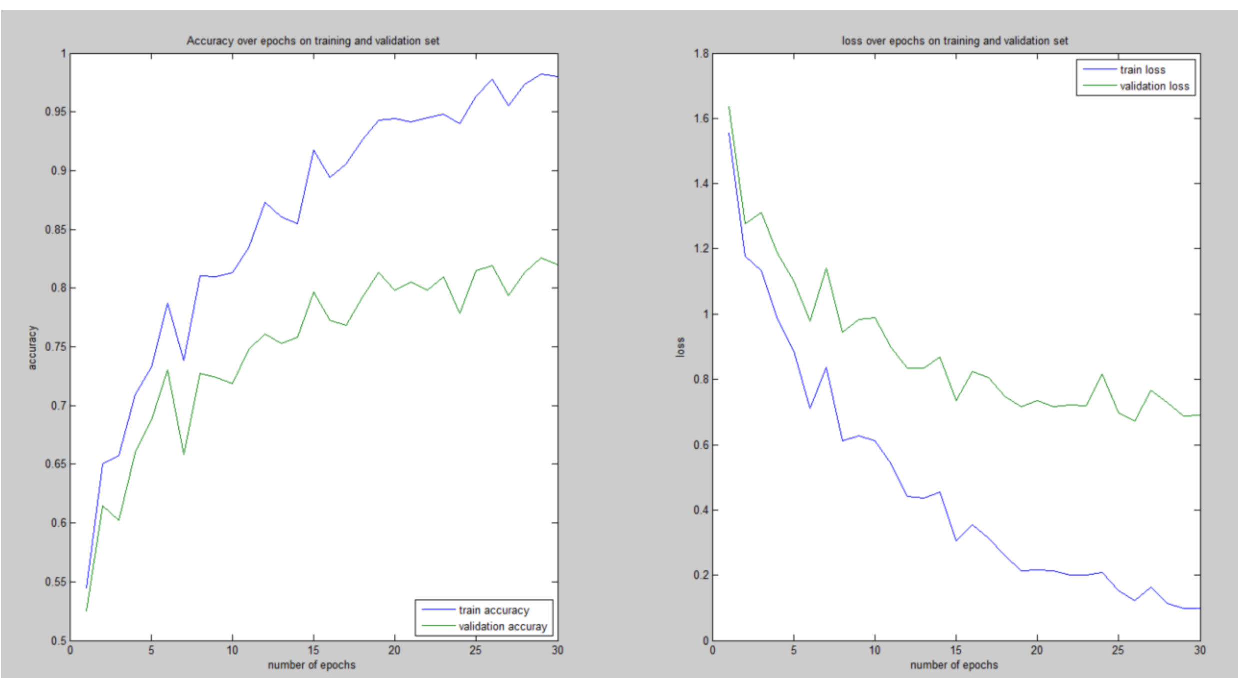
Epoch 30 - accuracy: 0.98006, 0.82000 loss: 0.09945, 0.69127

Q 3.1.2

The following four plots are generated from learning rate =0.01 and learning rate =0.001:



Epoch 30 - accuracy: 0.98006, 0.82000 loss: 0.09945, 0.69127



Epoch 30 - accuracy: 0.84152, 0.74000 loss: 0.61790, 0.92602

Therefore, learning rate indeed affects the learning process:

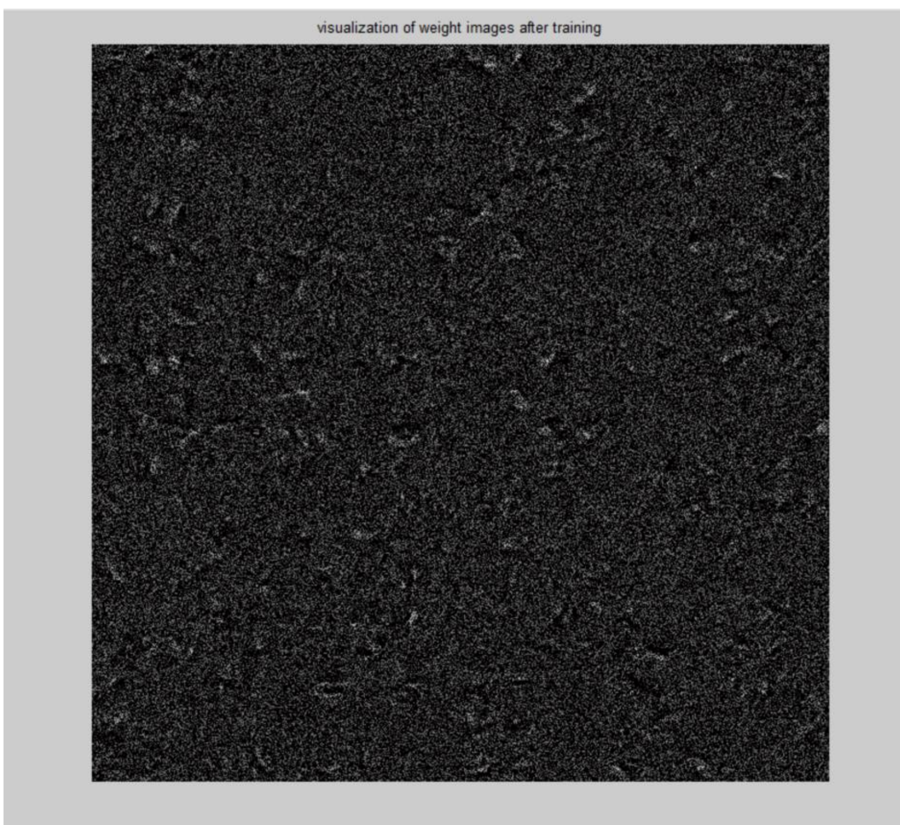
This parameter determines how fast or slow we will move towards the optimal weights. If the learning rate is very large we will skip the optimal solution. If it is too small, we will need too many iterations to converge to the best values. So, in our case, we will use the learning rate =0.01, because the learning rate 0.001 is too small to converge to the good weight values under 30 epochs.

The final accuracy of best network on test set is 82.19%.

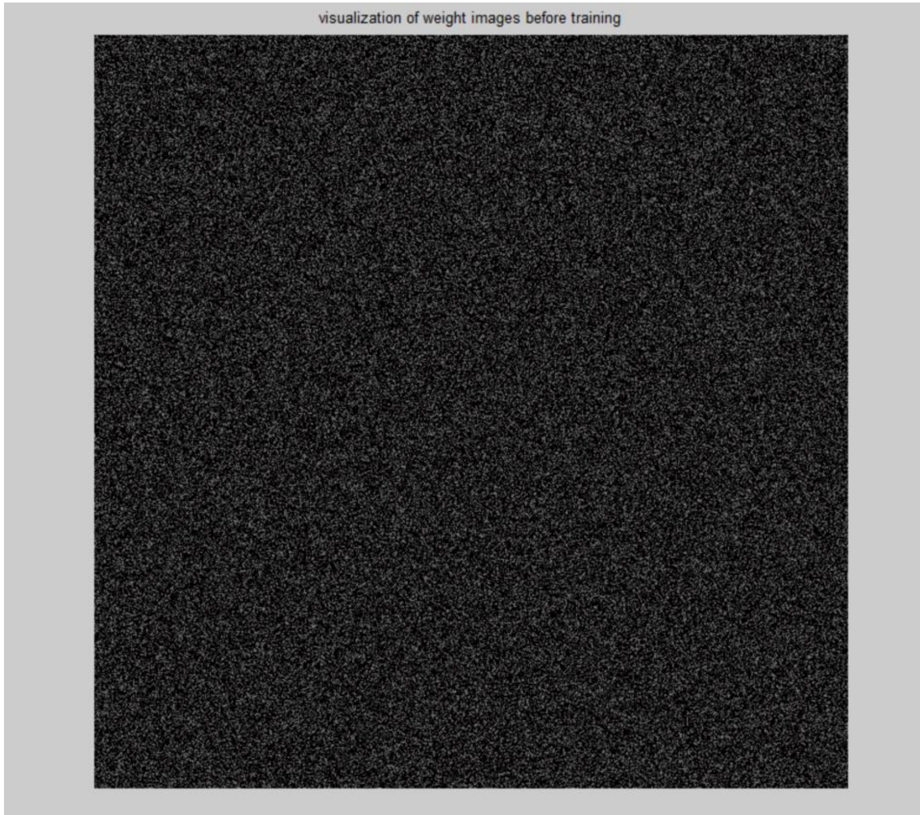
Q 3.1.3

The accuracy and cross-entropy loss on the test set are:

Test_acc: 82.19%   Test_loss: 0.6625



Visualization of first layer's weights after training



Visualization of first layer's weights before training

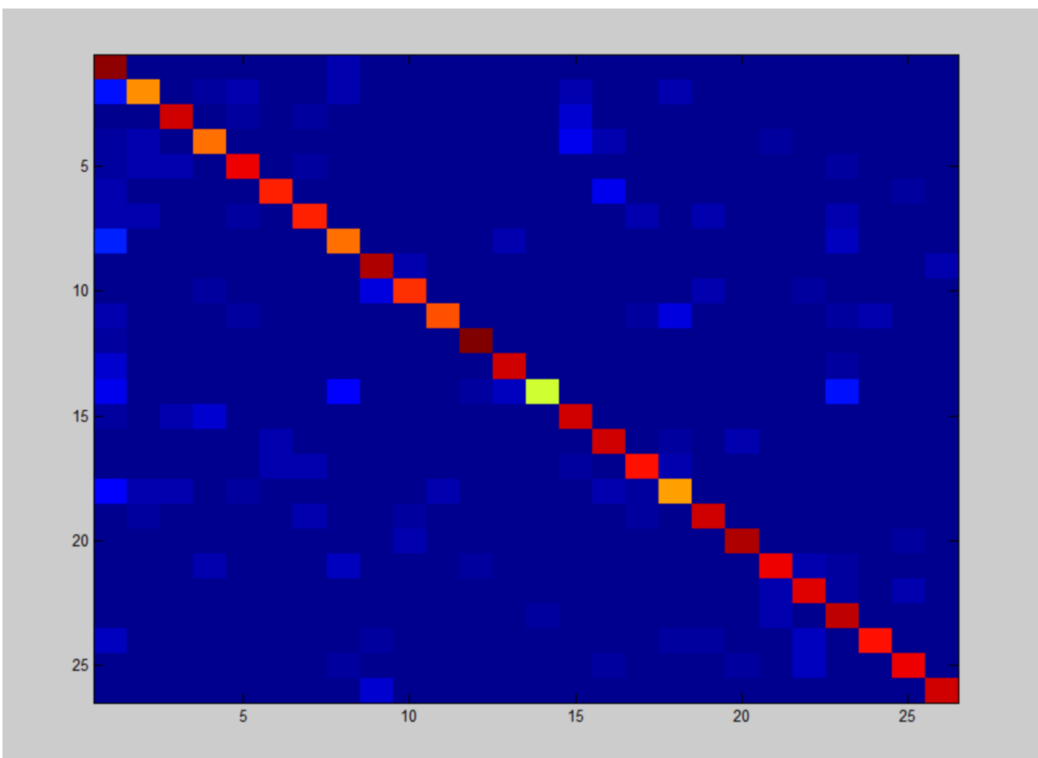
We can see the learned weights in the first layer already have significant patterns compared to the ones that right after initialization. Inside each neuron, weights have different position and orientation of emphasized area (white pixels), so that based on the combination of 400 neurons' weights, different characters will be recognized.

Q 3.1.4

confusion_matrix =

94	0	0	1	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
13	69	0	2	3	0	0	3	0	0	0	0	0	0	4	0	1	3	1	0	0	0	0	0	0	1
0	1	88	0	2	0	2	0	0	0	1	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0
2	3	1	73	1	0	1	0	0	0	0	0	0	0	10	4	1	0	1	0	2	0	0	1	0	0
2	3	3	0	84	0	2	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	2	0	0	1
3	0	0	0	1	80	0	1	0	0	0	1	0	0	0	10	0	0	1	0	0	1	0	0	2	0
3	3	1	0	2	1	80	0	0	0	0	0	0	0	0	0	3	0	3	1	0	0	3	0	0	0
14	1	0	0	0	0	0	73	0	0	1	0	3	0	0	0	0	1	0	0	1	0	5	1	0	0
0	0	0	1	0	1	0	0	90	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	3
1	0	0	2	0	0	0	0	8	78	0	0	1	0	1	0	0	0	4	1	1	2	0	0	0	1
3	1	1	0	2	1	0	1	0	0	75	1	0	0	0	0	2	8	0	0	0	0	2	3	0	0
2	0	0	0	0	0	0	0	1	0	1	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	88	0	0	1	1	0	0	0	0	0	2	0	0	0
9	0	1	1	0	0	0	11	0	0	1	2	5	54	0	0	0	1	0	0	1	1	13	0	0	0
2	1	3	6	0	0	0	0	0	1	0	0	0	0	87	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	0	0	0	0	0	0	0	0	0	87	1	2	0	3	0	1	0	0	1	0
1	0	1	0	0	3	4	0	1	0	0	0	0	0	2	0	81	3	1	0	1	0	1	0	0	1
11	4	3	0	2	0	0	0	0	0	3	0	0	0	0	4	2	68	0	1	0	1	0	0	0	1
0	2	1	0	1	0	3	0	0	2	0	0	0	0	0	0	2	1	88	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	4	0	0	0	0	0	0	0	0	0	91	0	0	0	0	2	1
0	0	0	4	0	0	0	5	0	0	0	2	0	0	0	0	0	0	0	0	84	3	2	0	0	0
0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	4	86	2	0	4	0
1	0	0	0	0	0	0	0	0	0	0	0	1	2	0	1	0	1	0	0	4	0	89	0	1	0
5	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	2	2	0	0	5	0	82	1	0
0	0	0	1	0	1	0	2	0	1	0	0	0	0	1	2	0	0	0	2	0	5	0	1	84	0
1	1	0	1	0	0	0	0	6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	88

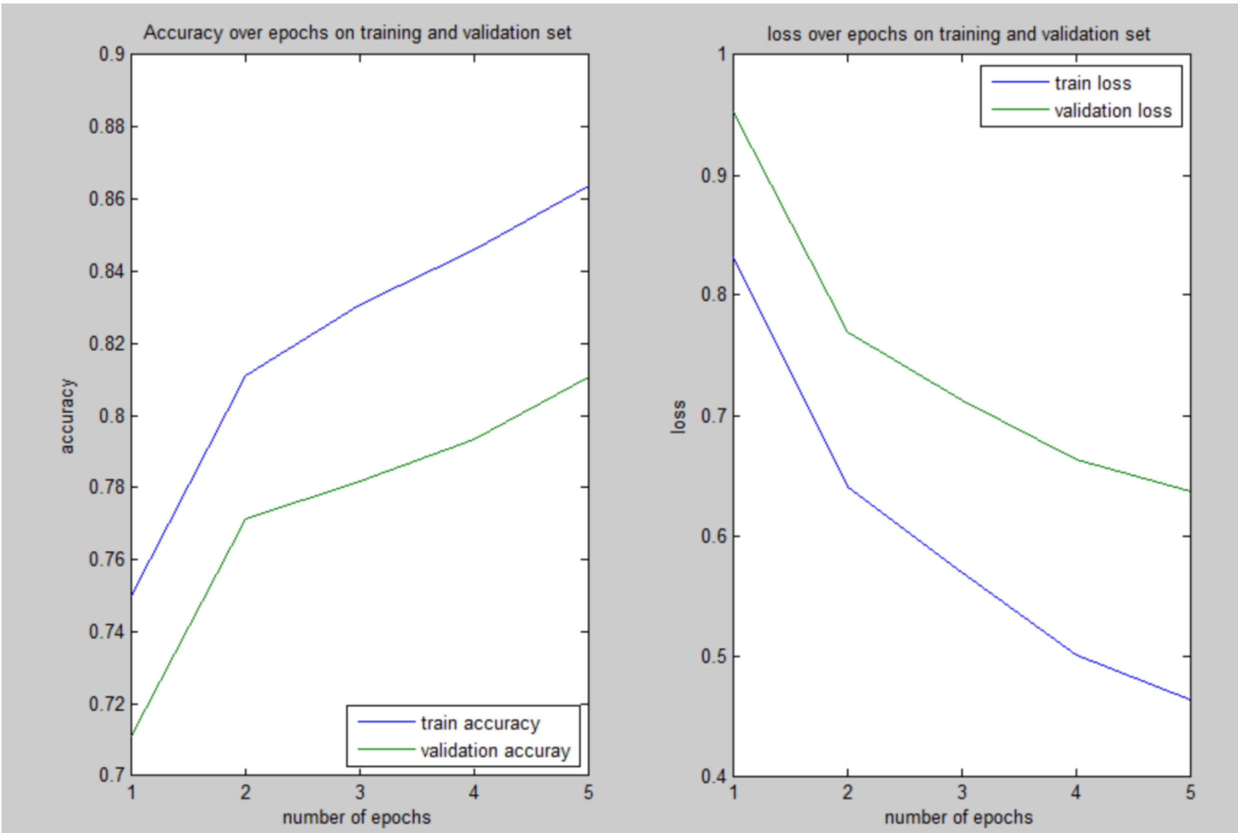
The confusion matrix



The visualization of confusion matrix

From the confusion matrix above, we can see the top two pairs of classes that are most commonly confused are: H and A; B and A (same as N and W). That's because those upper-case character if hand written, could be very similar so that even human eyes may get confused.

Q 3.2.1



Please refer to finetune36.m for details.

Using the fine tune method, with only 5 epochs, the accuracy and loss on training and validation set already reach:

Epoch 5 - accuracy: 0.86381, 0.81056 loss: 0.46332, 0.63647

Compared to the result without fine tuning, we can see within several epochs, the accuracy increases very quick.

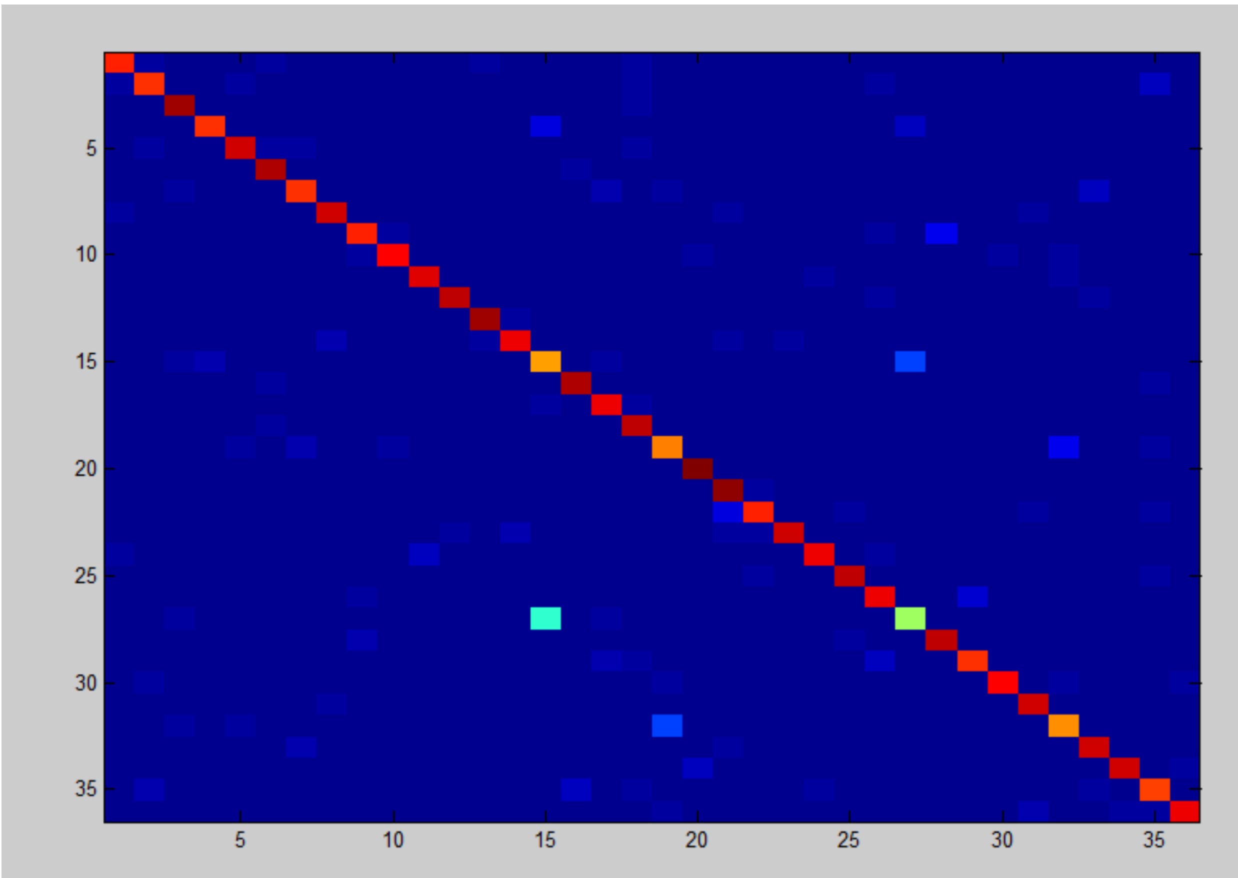
Q3.2.2

We can see the following plots have the similar patterns while the weights after training has brighter white pixels (higher weights value) at similar position. Also after training there are slightly more patterns.

The accuracy on test set is 80.02%, and the loss on test set is 0.6683

Q 3.2.3

In order to gain the best network model, in this part, I followed the fine tuning method but ran for 30 epochs instead of just 5 epochs.



82	3	0	1	0	3	1	1	0	0	1	0	3	0	0	1	0	2	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
2	81	1	1	2	0	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	6	0
0	1	94	0	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	81	1	0	0	0	1	0	0	1	0	0	9	1	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
0	2	0	0	89	2	2	0	0	0	1	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	1	92	1	0	0	0	0	0	1	0	0	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	3	1	0	0	80	0	0	0	0	0	1	1	0	0	4	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	6	0	0	1
3	0	0	0	0	0	0	90	0	0	1	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
0	0	0	0	1	0	0	0	82	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	10	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	3	85	0	0	0	0	0	0	0	0	1	3	0	0	0	0	0	0	0	0	1	2	0	2	0	1	0	0
0	1	1	0	0	1	0	0	0	0	88	0	1	0	0	0	0	1	0	0	1	0	0	2	0	0	0	0	0	0	1	2	0	1	0	0
0	1	0	0	0	0	0	1	0	0	0	91	0	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	1	0	0	0	3	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	94	2	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	4	0	0	1	0	2	86	0	0	0	1	0	0	2	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	4	1	0	0	0	0	0	0	0	0	0	70	0	3	0	0	0	1	0	0	0	0	0	18	0	0	0	1	0	0	0	0	0
1	0	0	0	0	2	0	0	0	0	1	0	0	0	0	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	1
0	1	1	0	0	0	0	0	0	0	0	0	1	0	2	1	87	2	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0
1	0	1	0	1	2	0	0	0	0	1	0	1	0	0	0	0	91	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	2	1	4	0	0	2	0	0	0	0	1	0	0	1	73	1	0	1	0	0	0	0	0	0	0	1	0	10	0	0	2	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	96	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	8	82	0	1	2	0	0	0	0	0	2	0	0	0	2	0
0	0	0	0	0	0	0	0	0	0	0	2	0	4	0	0	0	0	0	0	2	2	89	0	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	1	1	0	6	0	0	0	0	0	0	0	0	0	0	0	0	86	0	2	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	3	0	0	91	0	0	0	0	0	1	0	0	1	2	0
0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	87	0	0	7	0	0	0	0	1	0	0
0	0	3	0	0	0	0	0	0	0	0	0	0	0	41	0	2	0	0	0	0	0	0	0	0	0	51	0	0	0	0	0	1	1	1	0
0	0	0	0	0	1	0	0	4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	91	0	0	0	0	0	0	1	0
0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0	4	3	0	0	0	0	0	0	0	6	0	1	80	0	0	0	1	0	0	0
0	2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	3	1	0	0	0	0	1	1	0	0	1	85	0	2	0	0	0	2
1	0	0	0	0	1	0	3	0	0	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	89	0	0	0	0	1
0	1	2	1	3	1	0	0	1	1	0	0	0	0	1	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	71	0	0	0	1
0	0	0	1	1	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	0	0	0	0	0	90	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	1	0	0	0	0	90	0	3
0	4	0	0	0	1	0	0	0	0	0	0	0	0	0	6	0	3	0	0	0	0	0	2	1	0	0	0	1	0	0	1	2	0	79	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	2	0	0	0	0	0	0	0	0	0	0	0	4	0	0	2	2	87

From the two plots above, we can tell that the top two pairs that are most commonly confused are ‘Q’ and ‘O’; ‘O’ and ‘0’. This make sense because Q, O and number 0 are all in similar shapes, and even human eyes may get confused.

Compared this with the result in Q 3.1.4, we can see introducing more classes indeed affects the network. It will eliminate some confused pairs while introducing new confused pairs. Overall, it makes classification more challenging!

Q 4.1

The method outlined in the text are based on several assumptions:

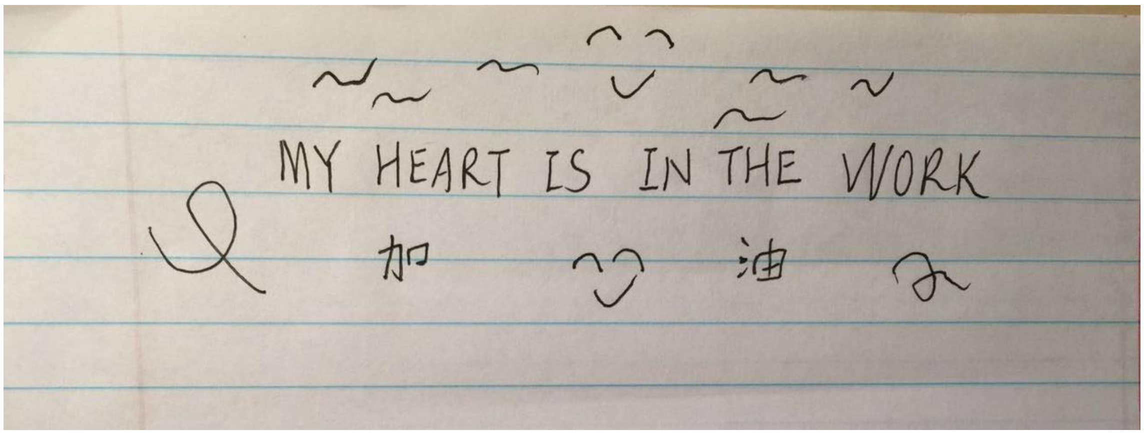
First, the background is clean, all the pixels except background belongs to certain classes (characters or numbers).

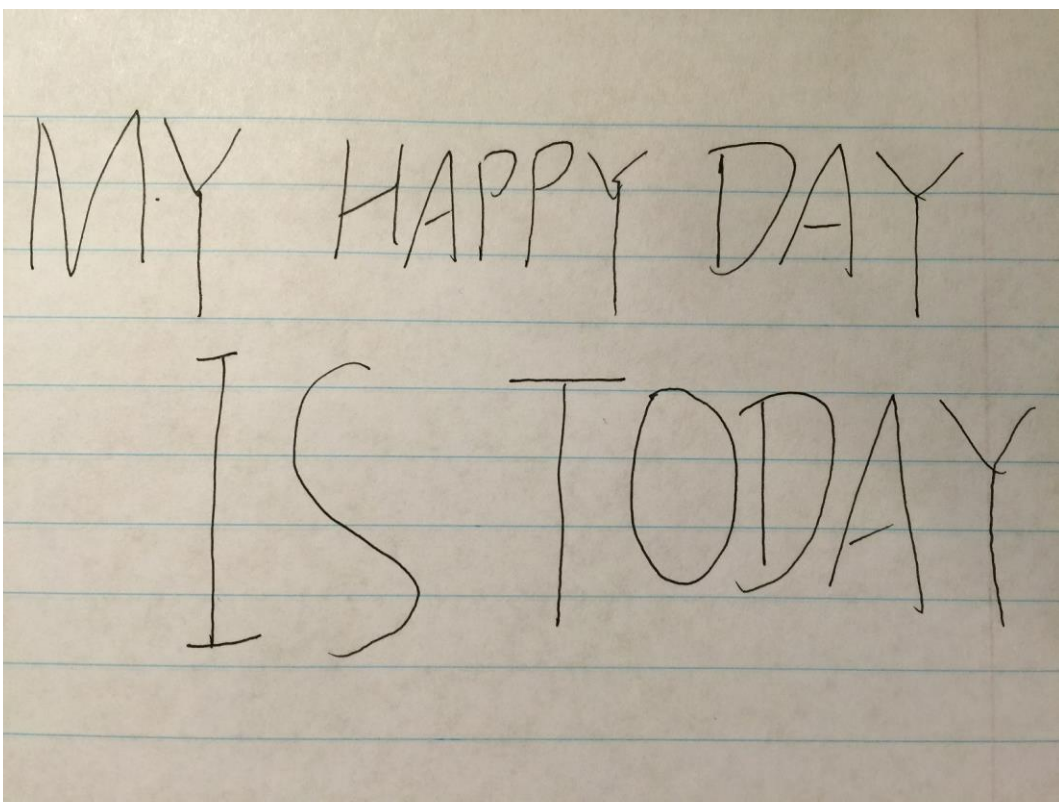
Second, the text contents are within reasonable size (32×32 $pixel^2$).

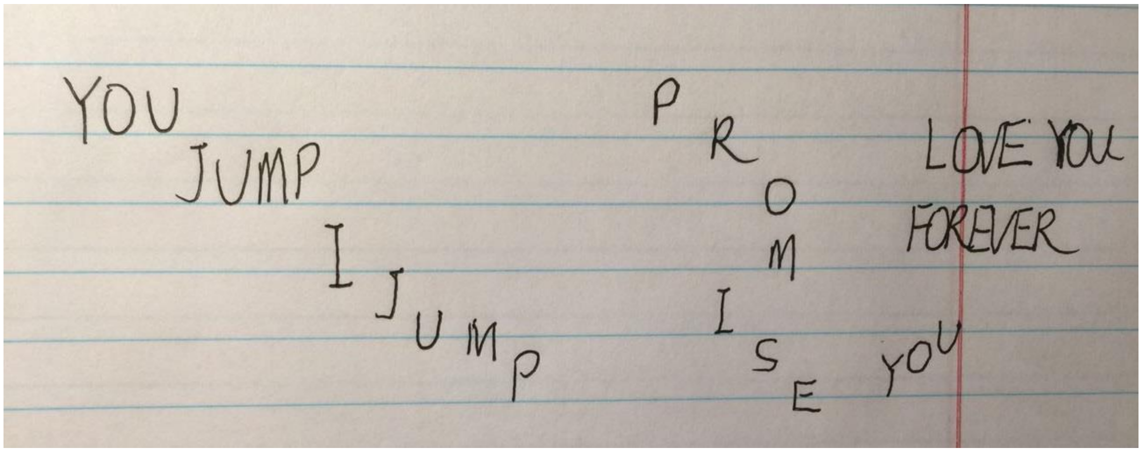
Thirdly, the text can be easily grouped line by line; that is to say they're in good order on the paper.

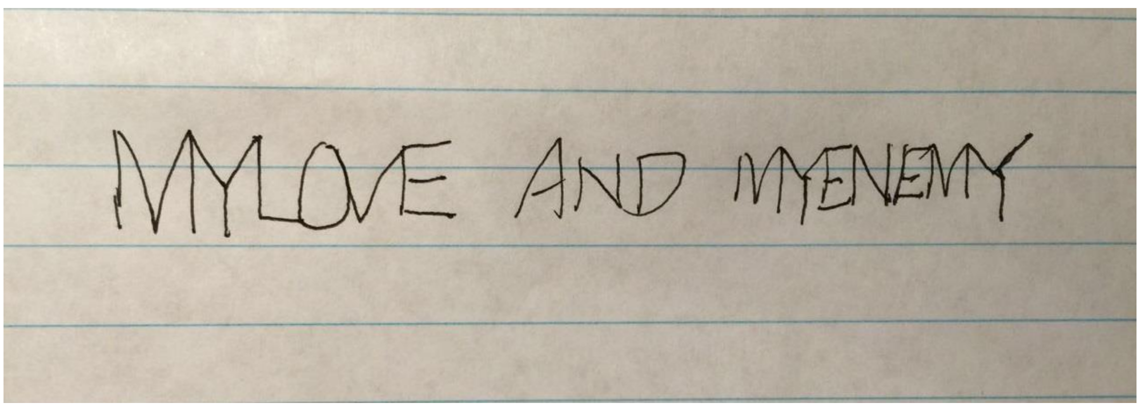
Also I think another assumption is each two elements(characters or numbers) have certain space in between so that they are not connected and can be distinguished as two separate elements.

See the following two examples:





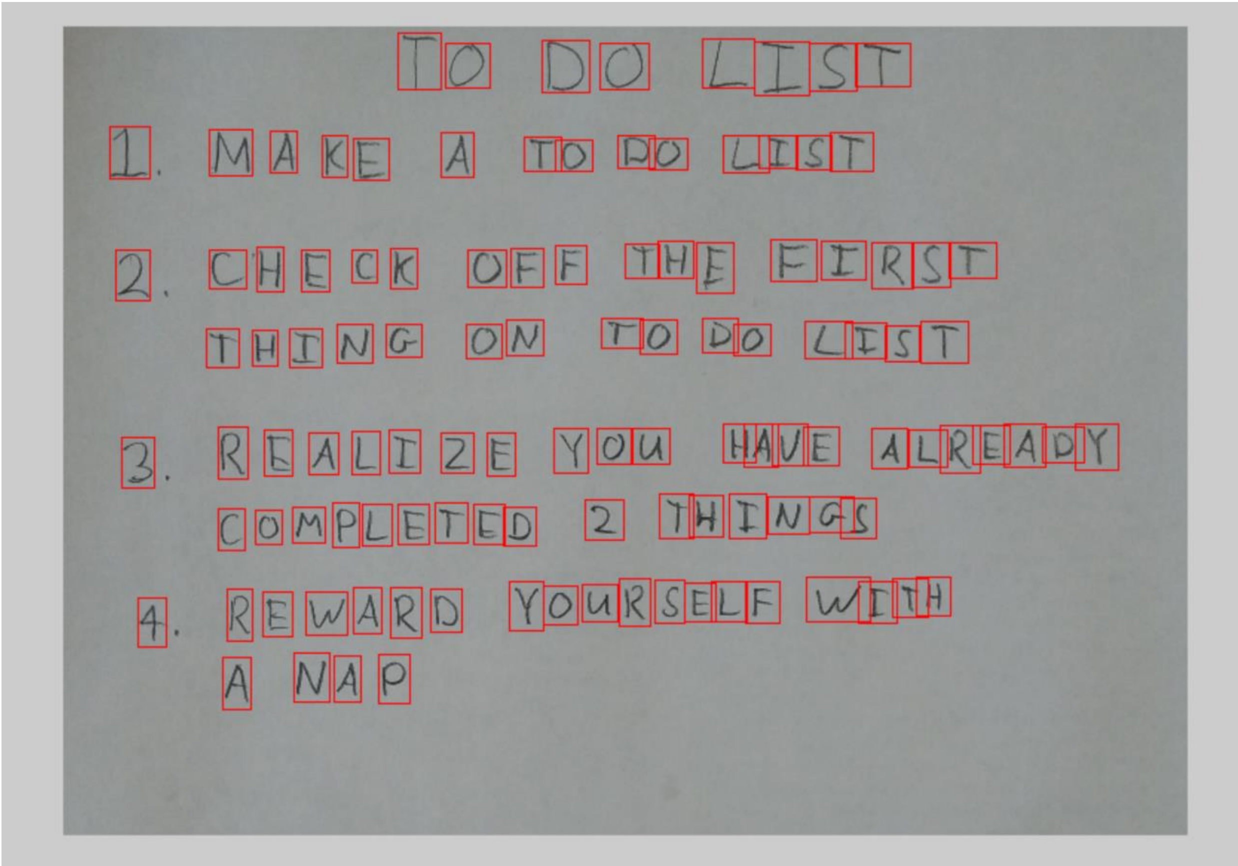


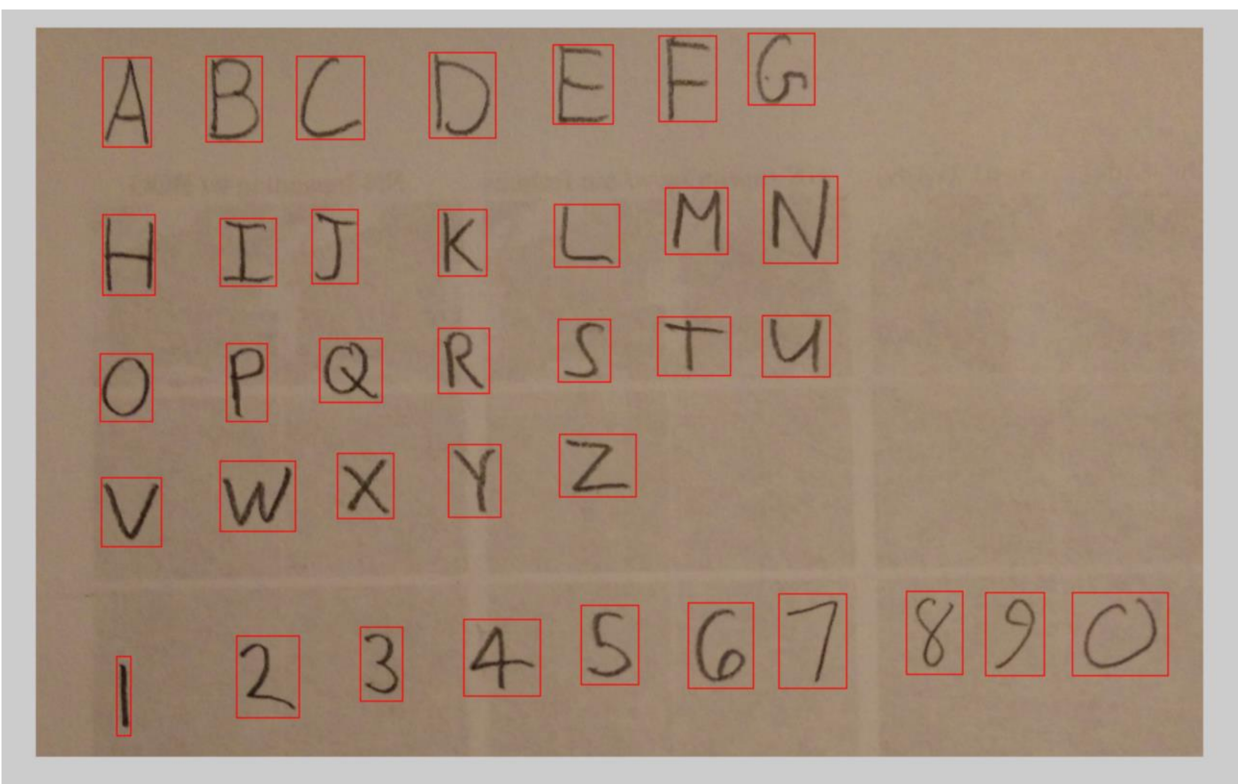


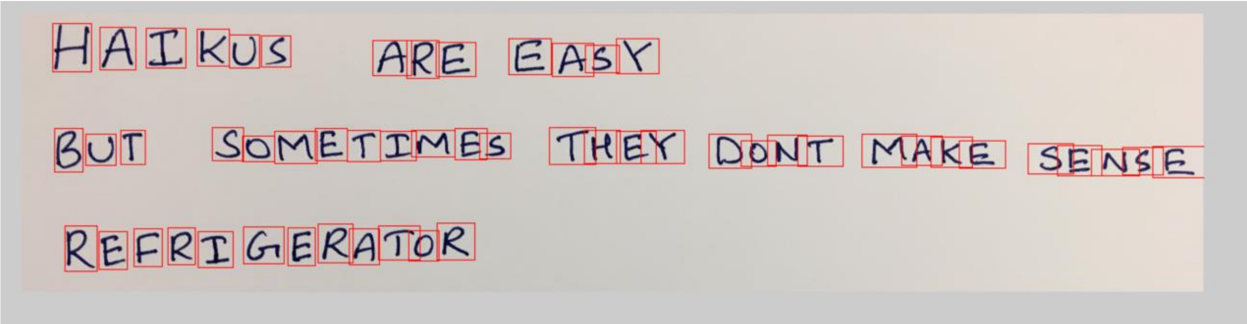
Q 4.2

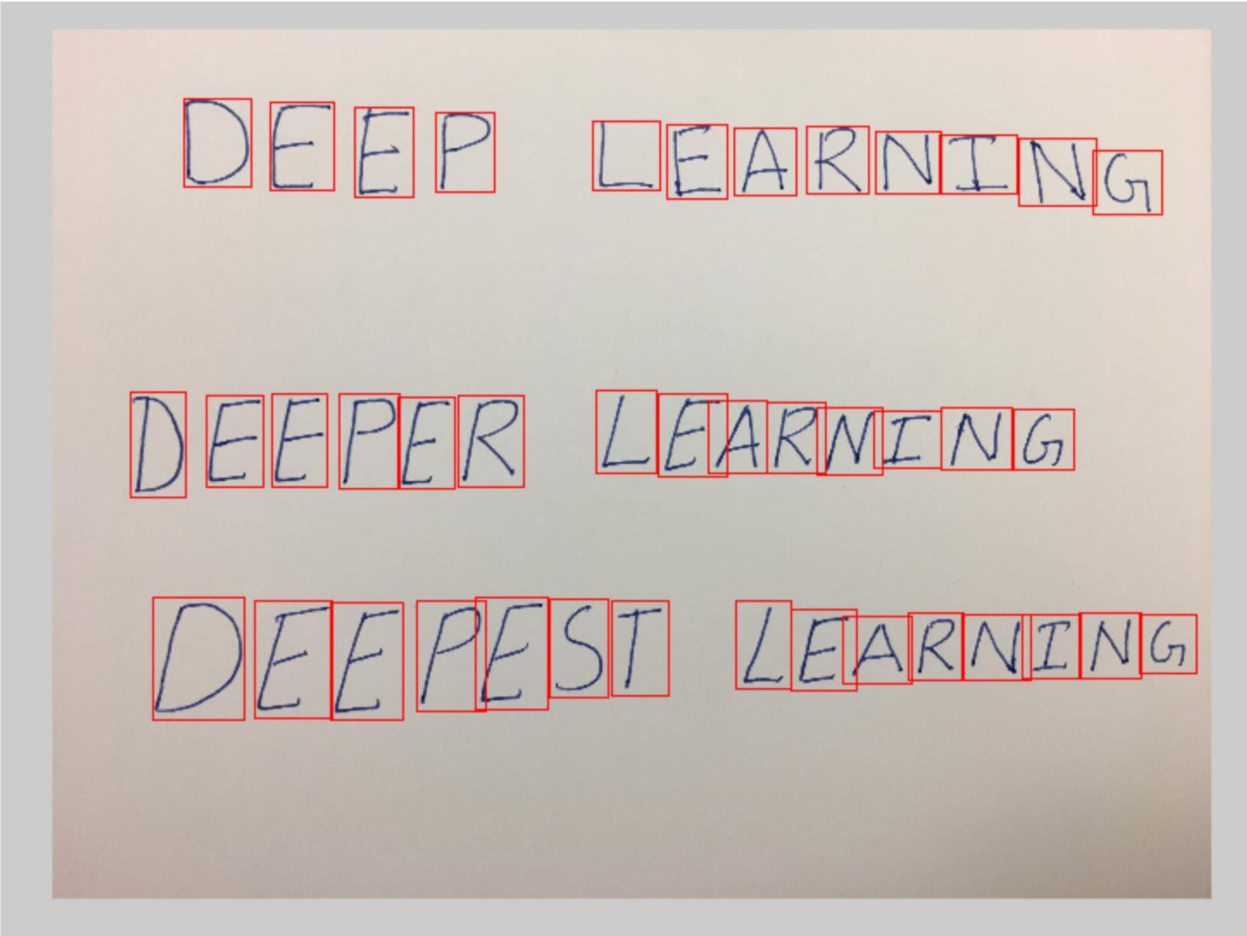
Please refer to findLetters.m and testFindLetters.m for more detailed implementation.

Q 4.3


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


A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
1 2 3 4 5 6 7 8 9 0





Q 4.4

After running extractImageText() function using testExtractImageText script, I got the following outputs:

TQ DQ LI5T

I MAKE A TDDQ LI5T

Z CHEQK QF8THE FIR5T

TMING QH T0D0 LI5T

3 REALZZEYOU BUVEALR6ADT

COMPLETKD 2THINGI

4 RBWARD Y0URSELF WITB

A NAP

A B C D E F G

M N

M I I K L

Q P Q R S T U

V W X Y Z

B Z 3 4 5 G 7 8 7 O

MA I KU5 ARE EAGY

BUT SBMETIMES TREX DQNT MAKE SGN6E

REGRI GERATOR

C C E P L E A R N I N G

D E E R E R L E A R N I N G

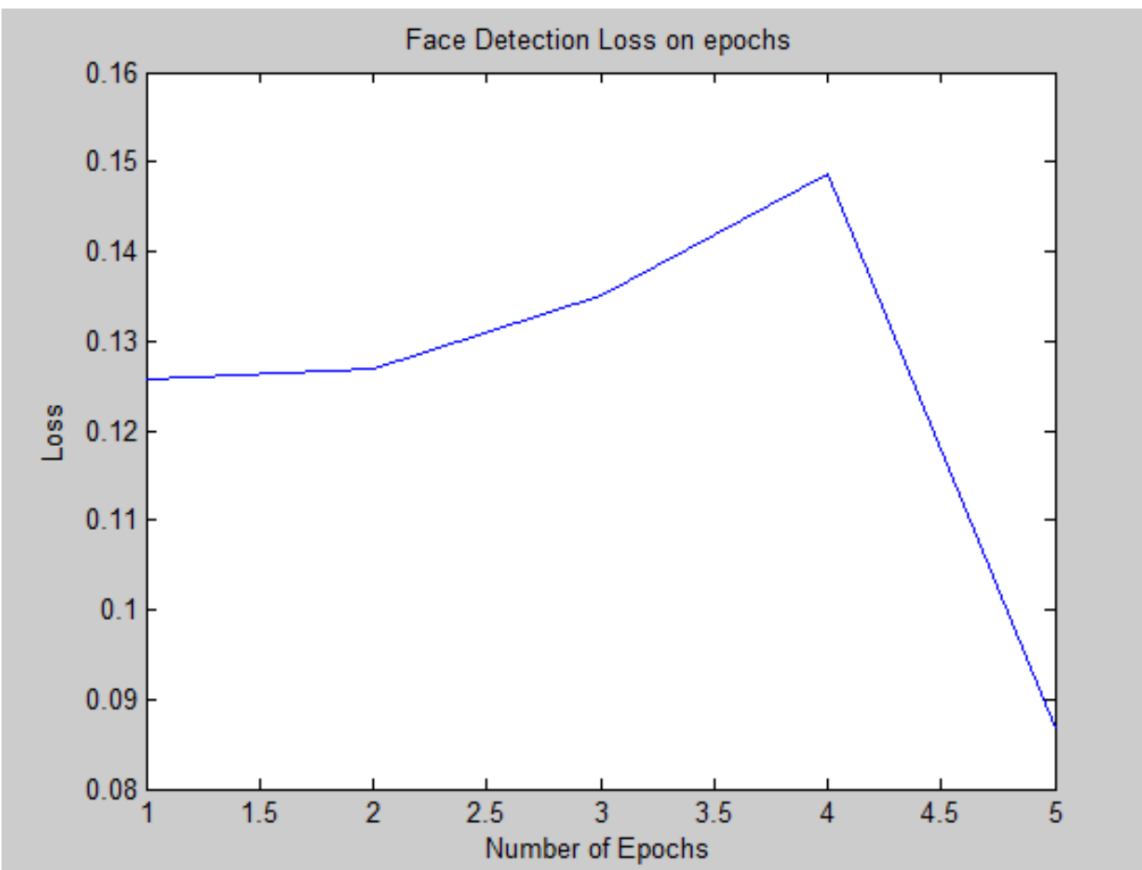
D E E F E S T L E A R N I N G

Q 5.1 Face Detection using Neural Networks (Extra)

Please refer to Face_Detection.m; FaceForward.m; FaceClassify.m; FaceComputeLoss.m; FaceBackward.m; FaceTrain.m and TestFace.m for implementation details.

The logic in this extra is similar to previous BP neuron network. But instead of using the loss function as cross-entropy loss, we use minimal square errors. And instead of softmax function in the output layer, I used sigmoid activation function also for the last output layer. The label data stores the position (x,y) and width, height data of face relative to the whole image.

Here I used layers=[64*64,400,200,100,4] networks and by several different epochs, I found the loss function has some periodical property and I manually tuned the epoch's number as 5 to have the relatively small loss value.

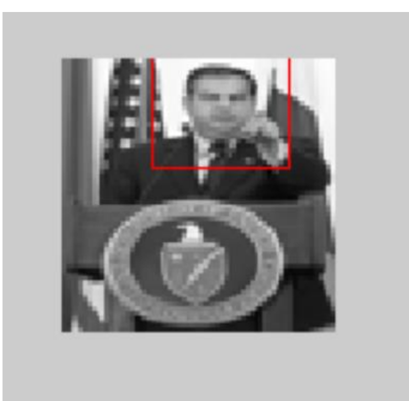


And the examples of face detection are shown below:

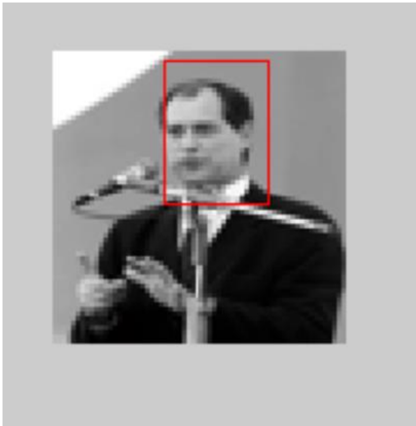
Frame 10:

Frame 55:

Frame 600:



Frame 820:

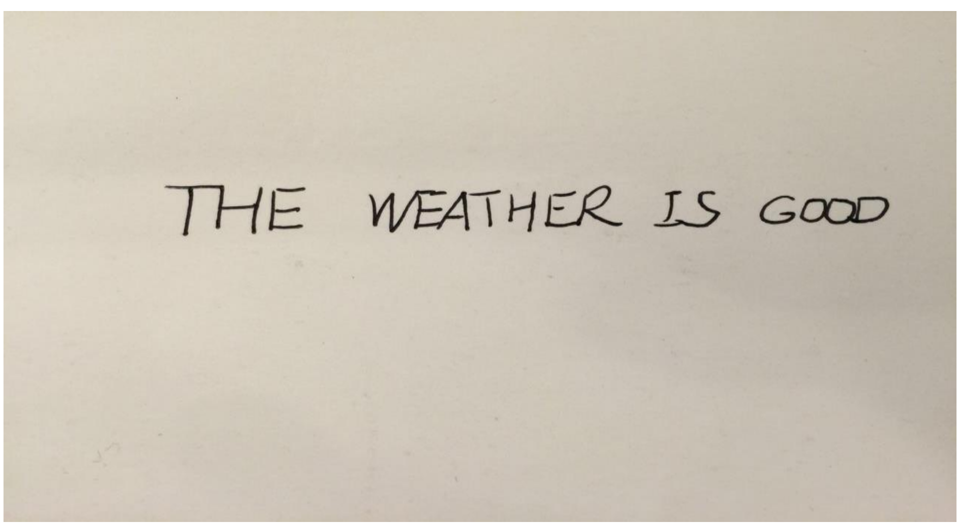


Usage: The W and b used for face detection is already stored as "Face_Detection.mat". Simply change the parameter "image_selected" in TestFace script, and run that script. Then we will see the image with detected face.

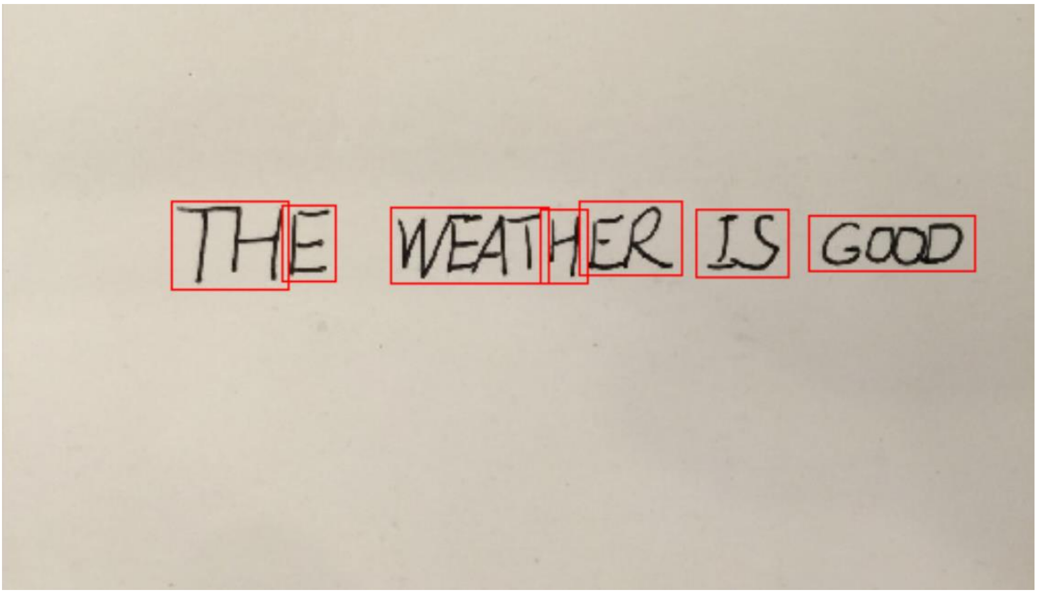
Q 5.3 Better OCR (extra)

For this part, I came up with the ideas to deal with challenging situations for having elements connected together and having too big characters.

Like the following image:



If we directly run original findLetters, we will get this:



My approach is to limit each connectivity's pixel size. Assuming each character has a maximum threshold pixel size, we can regenerate the number of characters based on breaking bigger connected pixels into several small ones. For example, "TH" are regarded as one character due to close positioning, but when we divide this into first half and next half based on average character's pixel size, we will fix this problem.