16720-A Computer Vision
Homework 5
Ruixin Liu
4/16/2019

Q1.1: Prove that softmax is invariant to translation.
We are given that

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

Set x_i to $x_i + c$,

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

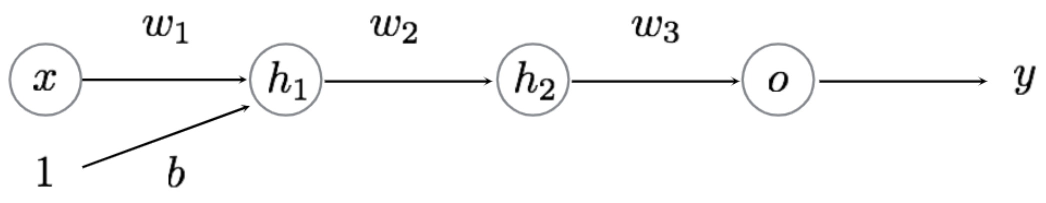
Since e^c is cancelled out,

$$softmax(x_i + c) = softmax(x_i)$$

If we set $c = -\max x_i$, the maximum value in the vector will be one. This makes all the elements in the vector between zero and one, which makes them easier to compare and handle.

Q1.2: Softmax can be written as a three step processes.
The range of Softmax function is from 0 to 1. The sum over all elements is 1.
One could say that "*softmax takes an arbitrary real valued vector x and turns it into a probability distribution.*".
Since we could treat a Softmax function as a probability distribution, s_i will stand for one specific sample, and S represents all the candidates that contribute to the distribution. The softmax function of x_i represents the probability of x_i in S.

Q1.3: Show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.
Assume we have a multi-layer neural network without an activation function as follows:



And we could derive an expression of y:

$$h_1 = w_1 x + b$$
$$h_2 = w_2(w_1 x + b)$$
$$y = o = w_3\big(w_2(w_1 x + b)\big) = w_1 w_2 w_3 x + w_2 w_3 b$$

The resultant y is still a linear function. Now we add a linear activation function to each hidden layer:

$$y = w_3 a_2\big(w_2 a_1(w_1 x + b)\big) = a_1 a_2 w_1 w_2 w_3 x + a_1 a_2 w_2 w_3 b$$

Which is still a linear function. If we assume the activation functions a_1, a_2 are non-linear, y will not be a linear function and the problem will not become a simple linear regression problem.

Q1.4: Given the sigmoid activation function σ(x) = 1 1+e−x, derive the gradient of the sigmoid function and show that it can be written as a function of σ(x).

The sigmoid function can be written as:

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

Using chain rule to take the derivative:

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

This function can be rewritten as:

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

Q1.5: Given y = xTW +b (or yj =Pd i=1 xiWij +bj), and the gradient of some loss J with respect y, show how to get ∂J ∂W , ∂J ∂x and ∂J ∂b . Be sure to do the derivatives with scalars and re-form the matrix form afterwards. Here is some notional suggestions.

Since we know $\frac{dJ}{dy} = \delta$, we just need to find $\frac{dy}{dW}, \frac{dy}{dx}, \frac{dy}{db}$ to solve the problem. We first want to take the derivative with scalars:

$$\frac{dy_j}{dW_{ij}} = x_i, \frac{dy_j}{dx_i} = W_{ij}, \frac{dy_j}{db_j} = 1$$

Expanding the scalar results to matrix forms:

$$\frac{dy}{dW} = [x \quad \cdots \quad x] \in \mathbb{R}^{d*k}$$

$$\frac{dy}{dx} = W \in \mathbb{R}^{d*k}$$

$$\frac{dy}{db} = \begin{bmatrix}1\\ \vdots \\ 1\end{bmatrix} \mathbb{R}^{k*1}$$

Q1.6.1: Why might it lead to a ”vanishing gradient” problem if it is used for many layers?

As shown in the figure below, the derivative of sigmoid function only contains a small value when g(x) is around 0, and is very close to 0 anywhere else. If one takes the derivative of a sigmoid function for multiple times, the output value will become very close to 0, causing a vanishing gradient.

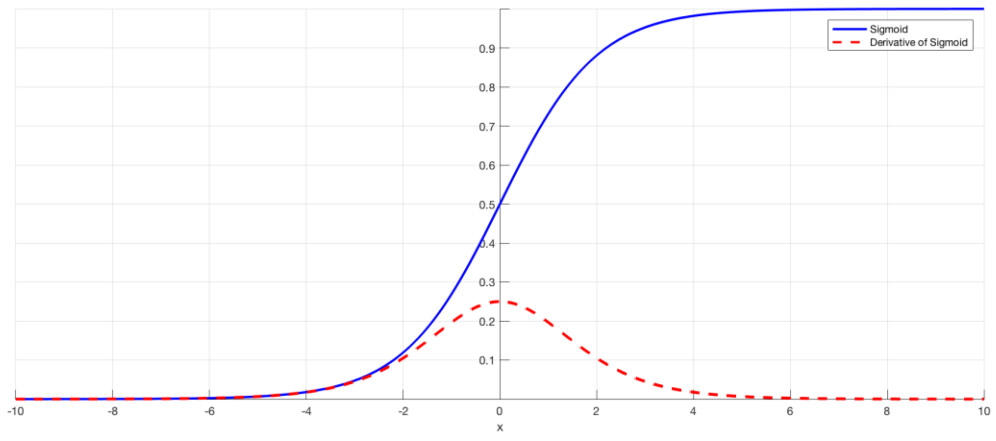


Figure 1 – Plot of Sigmoid function and its derivative. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

Q1.6.2: What are the output ranges of both tanh and sigmoid? Why might we prefer tanh ?

The range of tanh is from -1 to 1, and the range of sigmoid is from 0 to 1. As shown in the figure below, the amplitude of the derivative of tanh is much larger than that of the sigmoid. This may reduce the effect of vanishing gradient.

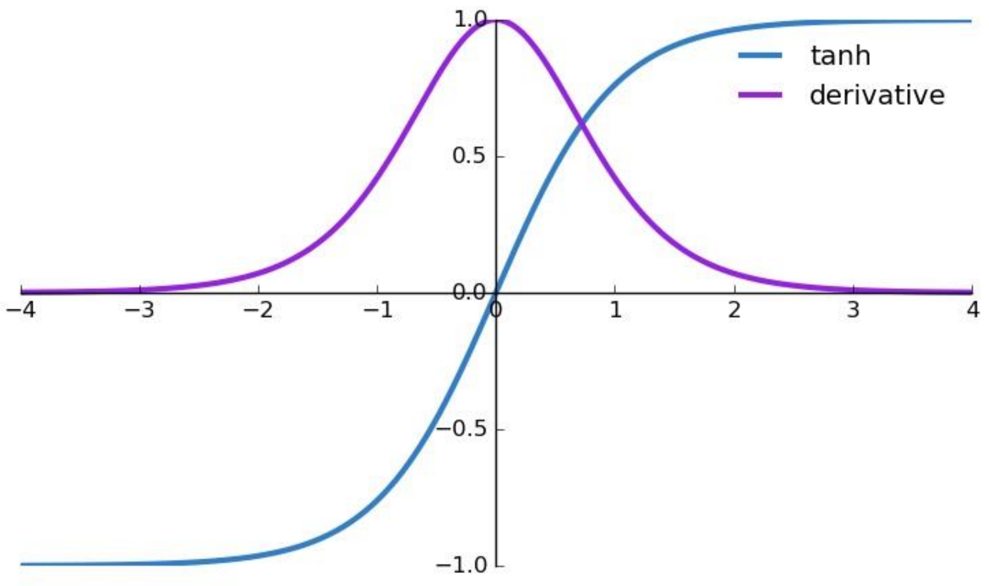


Figure 2 - Plot of Sigmoid function and its derivative. http://ronny.rest/blog/post_2017_08_16_tanh/

Q1.6.3: Why does tanh(x) have less of a vanishing gradient problem?

The amplitude of the derivative of tanh is much larger than that of the sigmoid. This may reduce the effect of vanishing gradient.

Q1.6.4: Show how tanh(x) can be written in terms of σ(x).

$$\tanh(x) = 2 * \sigma(x) - 0.5$$

Q2.1.1: Why is it not a good idea to initialize a network with all zeros?
If we initialize weights with all zeros, there will be no differences between the neurons – which means that the system ends up to be a linear regression model.

Q2.1.3: Why do we initialize with random numbers? Why do we scale the initialization depending on layer size (see near Fig 6 in the paper)?
Initializing weights with random numbers could eliminate the effect of symmetry, which means that each neurons in the hidden layers become unique, since they are derived from random weights. The advantage of this normalized initialization is that it prevents ill-conditioning and slower training due to a large difference in the magnitude of weights.

Q3.1.1: Modify the script to plot generate two plots: one showing the accuracy on both the training and validation set over the epochs, and the other showing the cross-entropy loss averaged over the data.
The two figures below show the average loss and accuracy with both training set and validation set over epoch. The batch size used is 48, learning rate 0.002 and the maximum iteration is 80. As expected, the training loss decreases fast at the beginning and slows down as the weights converge. On the other hand, training accuracy increases fast at the beginning and slows down as weights converge. The final accuracy on the validation set is 75.1%.

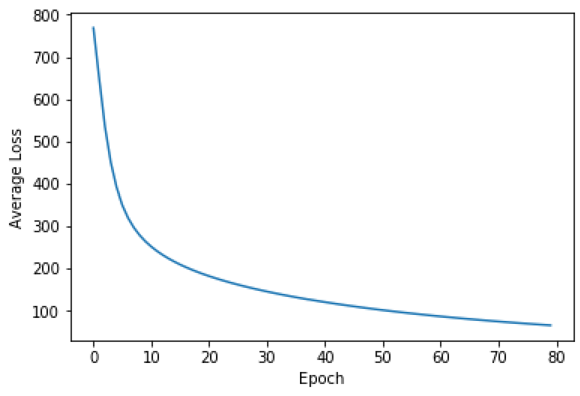


Figure 3 – Average training loss over epochs.

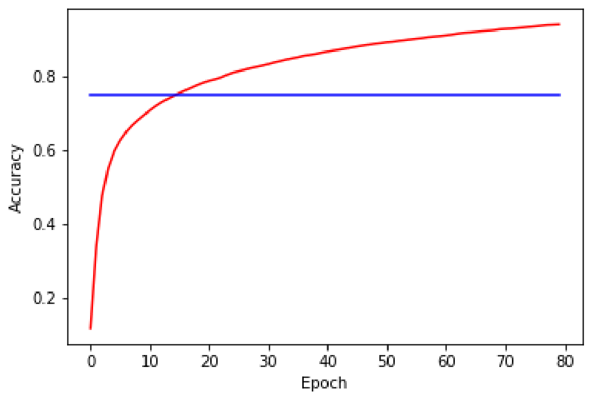


Figure 4 – Training and validation accuracy over epochs. The blue line stands for validation accuracy and the red line is training accuracy.

Q3.1.2: 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.

The following four plots show training and validation results with learning rate 0.02 and 0.0002. With 0.02 learning rate, the loss decreases really fast and the accuracy increases really fast. However, since the learning rate is too big, the weights fluctuate back and forth over the local minimum, as one could observe according to loss and accuracy on the plots. As a result, this issue significantly affect validation accuracy, which ends up with 62.9%.

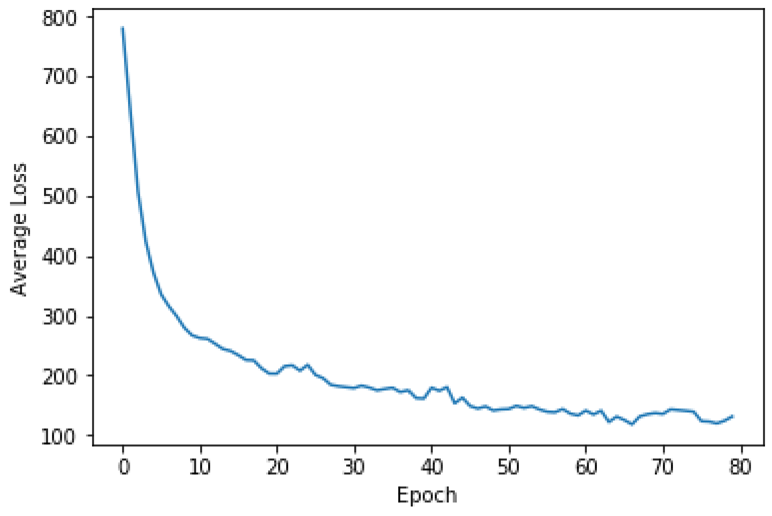


Figure 5 – Average loss over epoch with 0.02 learning rate.

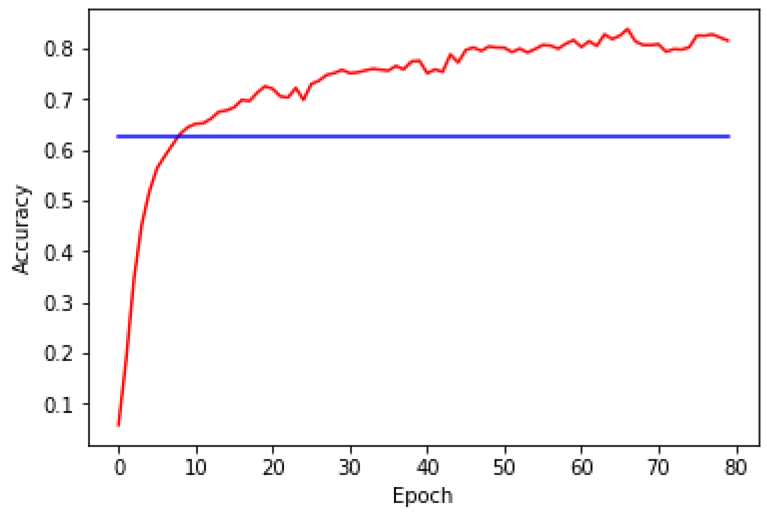


Figure 6 – Accuracies over epoch 0.02 learning rate. The blue line is validation accuracy and the red line is training accuracy.

Figure 7 and Figure 8 shows the result with 0.0002 learning rate. It can be seen that the gradients of loss and accuracy are still non-trivial, meaning the weights have not converged yet. This is because the small learning rate causes the weights change slowly. This learning rate ends up giving a validation accuracy of 65.8%.

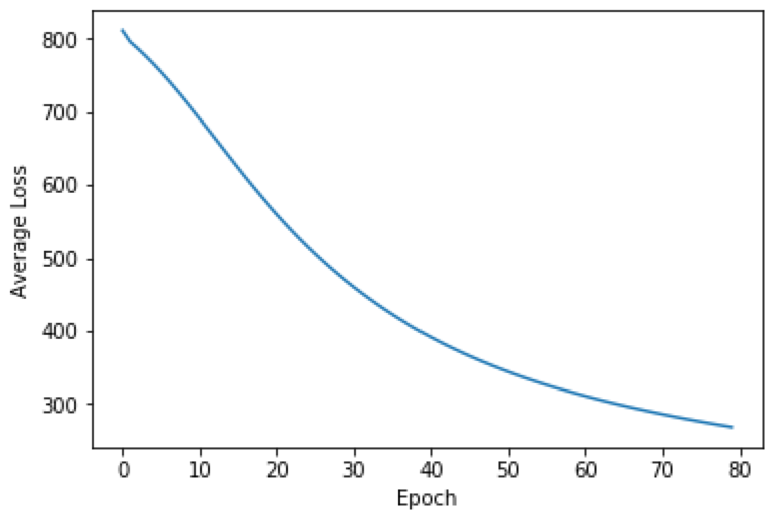


Figure 7 – Average loss over epoch with learning rate of 0.0002.

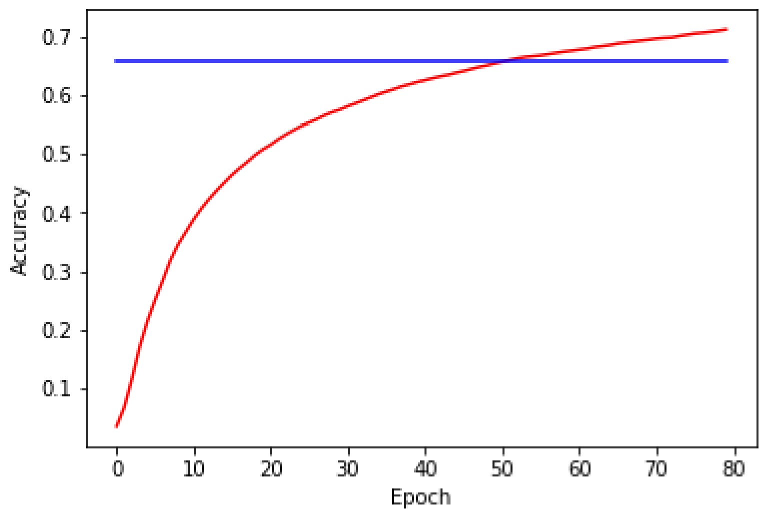


Figure 8 – Accuracies over epoch 0.02 learning rate. The blue line is validation accuracy and the red line is training accuracy.

Q3.1.3: Visualize the first layer weights that your network learned (using reshape and ImageGrid).

In Figure 9, the plot on the left shows the weights right after initialization. It looks like white noise since there is no correlation between neighboring pixels. The plot on the right shows the first layer weights. It can be easily observed that there are correlation between neighboring pixels, as one could estimate their gradients (assuming the darker regions stand for smaller values and brighter regions stand for larger values, the change between them is gradual).

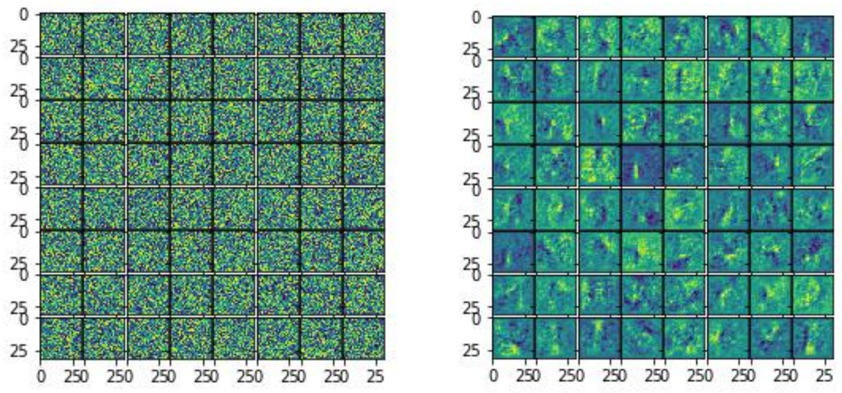
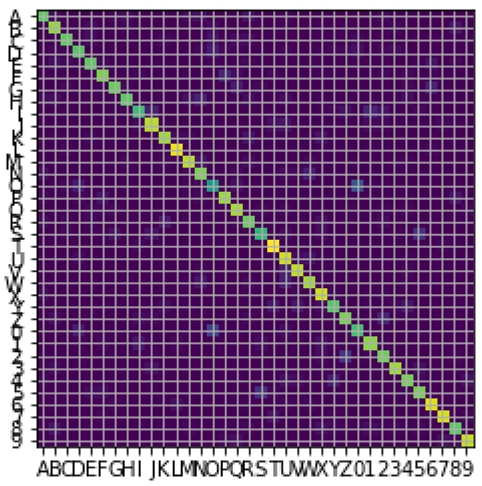


Figure 9 – Visualization of weights from initialization and first layer.

Q3.1.4: Visualize the confusion matrix for your best model.

Figure 10 shows the confusion matrix of the classification results. Most of the samples are classified correctly. However, one could see that there is a noticeable confusion between 0 and O, z and 2, and 5 and S. This is reasonable because these letters do look similar, meaning that some of their features are the same.



Q4.1: What are two big assumptions that the sample method makes?

All the letters are included in the training; All text is oriented in the same direction. I think the two images below would fail, since they do not satisfy one of the assumptions.

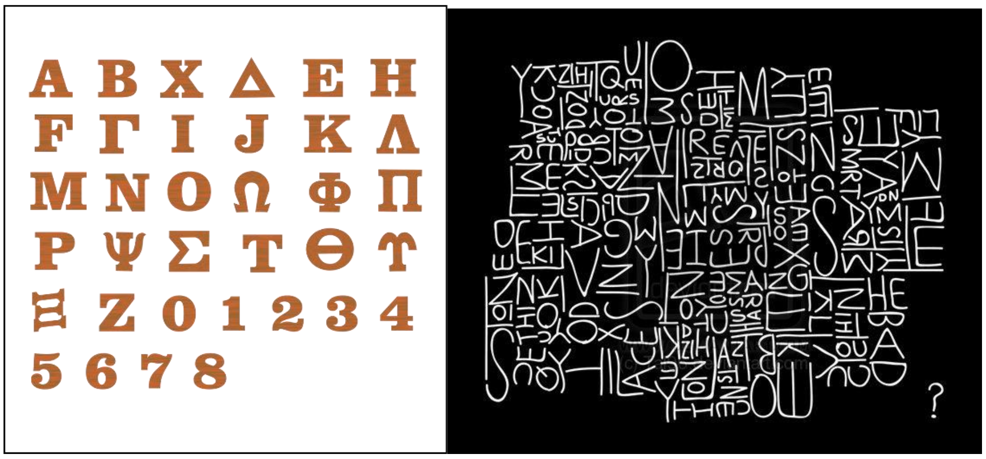


Figure 11 – Images that would fail in character detection.

Q4.3: Include all the result images in your writeup.

The following four figures show the bounding boxes of the four sample images. All letters are located along with few noises. However, there is a flaw in thresholding the images – the letters are composed of dash-line-ish lines, which may affect the accuracy in classification.

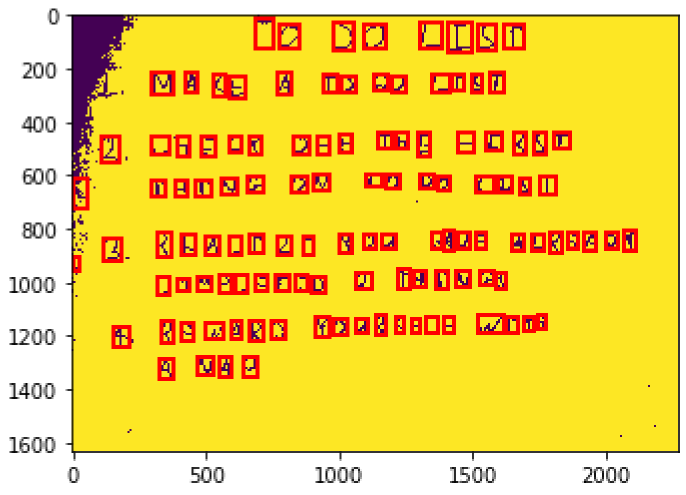


Figure 12 – Image extraction of 01_list.

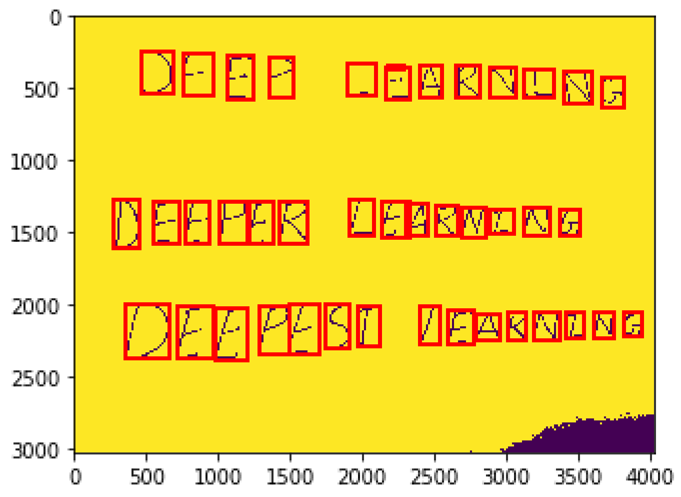


Figure 13 – Image extraction of 02_letters.

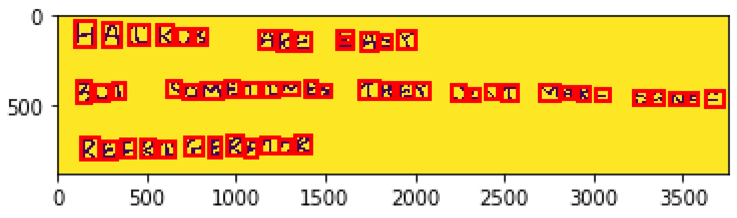


Figure 14 – Image extraction of 03_haiku.

\

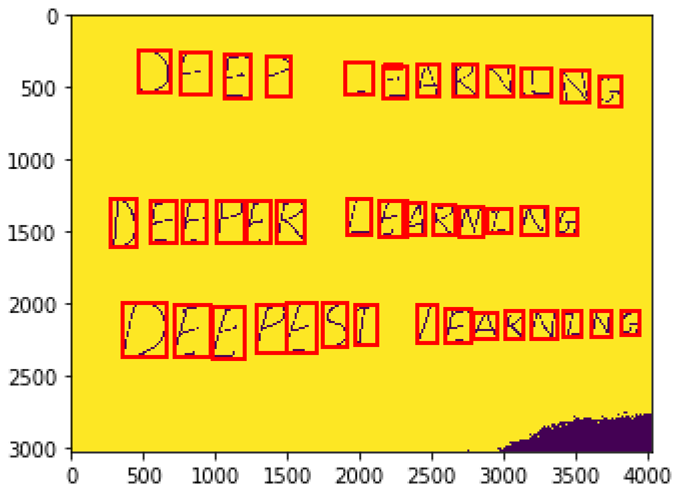


Figure 15 - Image extraction of 04_deep.

Q4.4: Run your run q4 on all of the provided sample images in images/. Include the extracted text in your writeup.

The four figures below show the detection results of the four sample images. I did not finish the sorting part, so the letters are not categorized by rows. However, by comparing the total numbers of each letters, it is found that the detection result is not very accurate. This could be because the issue of the image extraction mentioned in Q4.3.

```
The prediction result is:  ['S' 'T' 'L' 'T' 'I' 'J' 'J' 'T' 'X' 'N' 'A' 'T' 'A' 'L' 'I' 'K' 'L' 'X'
 'T' '6' 'J' 'J' 'F' 'T' 'W' 'K' 'X' 'T' 'E' 'T' 'F' 'W' 'K' 'J' 'F' 'L'
 'L' 'J' 'J' 'D' 'T' 'J' 'N' 'L' 'T' 'L' 'G' 'J' 'Q' 'W' 'X' 'M' 'I' 'T'
 'W' 'Y' 'A' 'W' 'U' 'A' 'K' '6' 'D' 'K' 'E' 'Y' 'J' 'A' 'U' 'L' 'L' 'F'
 'L' 'A' 'X' 'F' '7' 'X' 'I' 'I' 'N' 'G' 'W' 'S' 'J' 'P' 'T' 'L' 'L' 'F'
 'Y' 'L' 'L' 'J' 'H' 'W' 'K' 'T' 'X' 'Y' 'L' 'L' 'F' 'F' 'U' 'J' '0' 'K'
 'C' 'K' 'F' 'W' '4' 'W' 'W' 'A' 'P']
```

```
The prediction result is:  ['A' 'G' 'F' 'F' 'J' 'C' 'M' '0' 'W' 'Y' 'L' 'K' 'I' 'M' '8' 'W' 'T' 'X'
 'K' 'Q' 'V' 'C' 'Z' 'Y' 'X' 'W' 'U' 'X' 'J' 'D' 'J' 'U' 'Y' 'F' '3' 'Z'
 'B']
```

```
The prediction result is:  ['L' 'M' 'F' 'I' 'W' 'L' 'X' 'H' 'Y' 'P' 'K' 'G' 'A' 'B' 'P' 'S' 'T' 'F'
 'E' 'H' 'T' 'M' 'Y' 'T' 'H' 'I' 'Y' 'E' 'S' 'L' 'C' 'Y' 'W' 'C' 'T' 'A'
 'K' 'C' '6' 'Y' '6' '6' 'G' 'X' 'K' 'R' 'T' 'K' 'G' 'E' 'H' 'G' 'F' 'C'
 'B' 'I']
```

```
The prediction result is:  ['A' 'J' 'Y' 'Y' 'Y' 'C' 'F' 'K' 'Q' 'C' 'Y' 'L' 'X' 'U' 'C' 'T' 'Y' 'Y'
 'Y' 'F' 'K' 'Y' 'D' 'K' 'Q' 'X' 'G' 'L' 'C' 'F' 'Y' 'F' 'F' 'Y' 'P' 'C'
 'F' 'M' 'K' 'G' 'X' 'I' 'Q']
```