16720 HW 5 Write-up

Jagjeet Singh

- **Q1.1.1 In training deep networks, the ReLU activation function is generally preferred to the sigmoid activation function. Why might this be the case?**

The main advantage of ReLU over Sigmoid is that it prevents the problem of vanishing gradient.
$y = max(x, 0)$
Given this function, the gradient remains constant even at very high values of x. This won't be the case in sigmoid as the gradeint approaches zero as the value of x becomes large.

- **Q1.1.2 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**

If there was no hidden layer:
Weights and biases between input and output layer are W and b respectively. Linear Activation function in the output layer is $f(x) = mx + c$.
$y = m(Wx + b) + c$
i.e., $y = mWx + mb + c$

Network with 1 hidden layer:
Lets say we have 1 hidden layer in the network.
Weights and biases between input and hidden layer are W_1 and b_1. Weights and biases between hidden and output layer are W_2 and b_2 respectively. Linear Activation function in the hidden and output layers are $f_1(x) = m_1x + c_1$ and $f_2(x) = m_2x + c_2$ respectively.
Hidden layer: $h = m_1(W_1x + b_1) + c_1$
Output layer before activation: $y_a = W_2h + b_2 = W_2m_1(W_1x + b_1) + W_2c_1 + b_2 = W_1W_2m_1x + W_2b_1m_1 + W_2c_1 + b_2$
Output layer after activation : $y = m_2y_a + c_2 = m_2(W_1W_2m_1x + W_2b_1m_1 + W_2c_1 + b_2) + c_2 = m_2W_1W_2m_1x + m_2W_2b_1m_1 + m_2W_2c_1 + m_2b_2 + c_2$
i.e. $y = m_2W_1W_2m_1x + m_2W_2b_1m_1 + m_2W_2c_1 + m_2b_2 + c_2$

From the two equations it is clear that output is of the form $y = Wx + b$ for both the cases (0 and 1 hidden layer). If we add more hidden layers to the network, the equation will boil down to the same format, i.e., format of a fully connected layer. This proves that adding more layers will not have any impact on the representation capability of the network

- **Q2.1.1 Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value?**

  The problem with initializing the network weights with constant values is that it will not break the symmetry of the network. For example, if we have a 3 layer network and all the weights are initialized to 1, this will result in all hidden nodes being same, irrespective of what is the input. Further all the output nodes will also be the same. When error is back-propagated, all the gradients will have the same value in a layer. This means that we are stuck in a local minima where the nodes have the same value everytime. This is not a desirable property of neural networks and they expect the network to be unsymmetric.

- **Q2.1.3 Describe the initialization you implemented in Q2.1.2 and any reasoning behind why you chose that strategy**

  The initialization used was random for weights and all-zero for biases.
  Weights: Random initialization helps in avoiding any local minima in the Error function. Further, I also analyzed the effect of 2 more initialization.
  1. Uniform: This will give more weight to the pixel values which are toward the end. Hence, rejected.
  2. Gaussian: This will give more weight to pixel values which are around the center. Hence, rejected.
  Bias: I used all-zero initialization as they won't suffer from same gradient value, unlike weights. Random initialization will also work here.

- **Q2.4.1 Give pros and cons for both stochastic and batch gradient descent**

Stochastic Gradient Descent:
Pros:
1. Faster in terms of number of iterations (weight updates)
2. Helps in escaping undesirable local minima
3. Will be faster in terms of epochs when compared to batch GD if data size is large.
Cons:
1. Noisy updates. Each iteration cannot guarantee movement along the steepest decline. This also causes a lot of jerk
Batch Gradient Descent:
Pros:
1. Always moves in the direction of steepest decline. Hence, number of updates required to reach the minima are less
2. Good for convex error functions since we directly move to the optima (global optima in case of convex functions)
Cons:
1. Much slower in terms of number of iterations (weight updates)
2. Much slower in terms of number of epochs if the data size is large

That said, a better tradeoff is to use mini-batch gradient descent. It incorporates pros of both batch and stochastic GD

- **Q3.1.2 Plots and Test Accuracy**

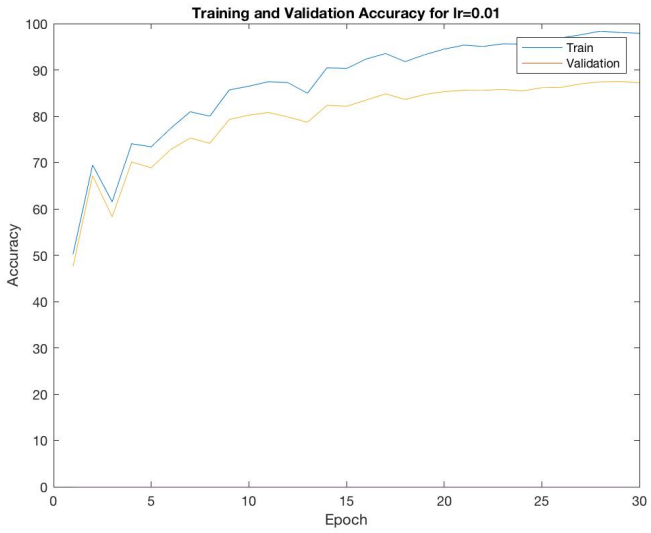


Figure 1: Training and Validation Accuracy for learning rate 0.01 after 30 epochs

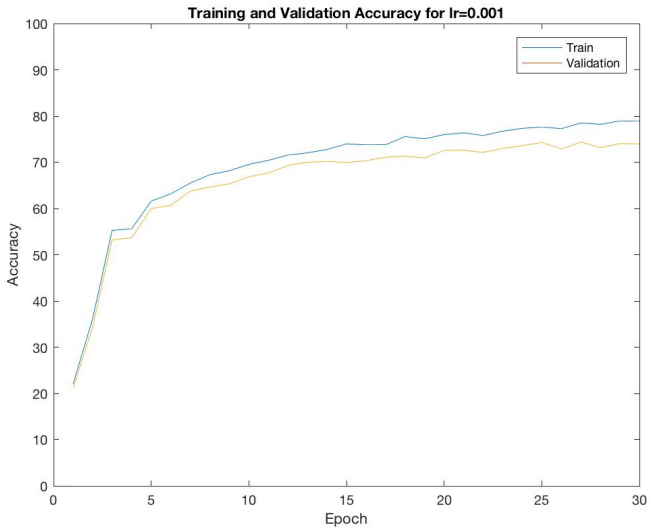


Figure 2: Training and Validation Accuracy for learning rate 0.001 after 30 epochs

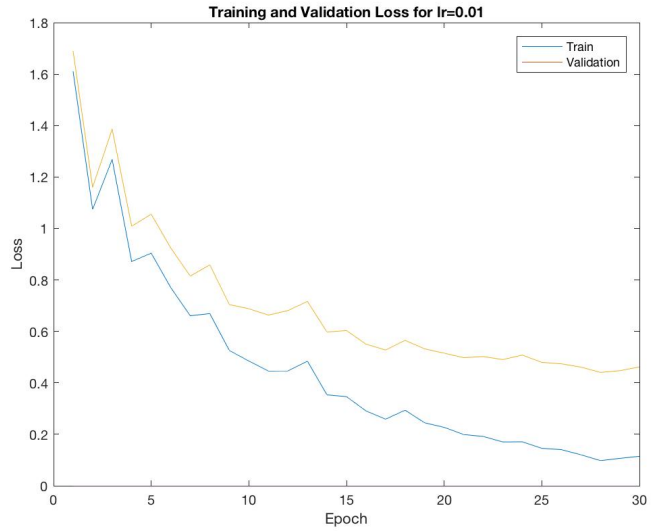


Figure 3: Training and Validation Loss for learning rate 0.01 after 30 epochs

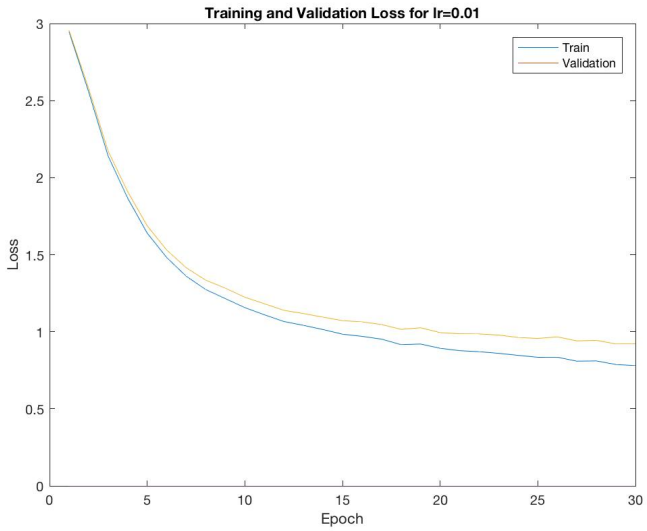


Figure 4: Training and Validation Loss for learning rate 0.001 after 30 epochs

Effect of learning rate:
It is evident from the plots that 0.01 is a better learning rate than 0.001 in this case. Low learning rate causes the network to learn slowly. As such, the accuracy attained even after 30 epochs was not much. Higher learning rate trains faster but there is a lot of noise which comes with it. As such, there are many spikes noticed in the plots corresponding to lr 0.01.

The best accuracy obtained after 30 epochs on test set is **88.69 %**

- **Q3.1.3 Accuracy,loss and visualization**

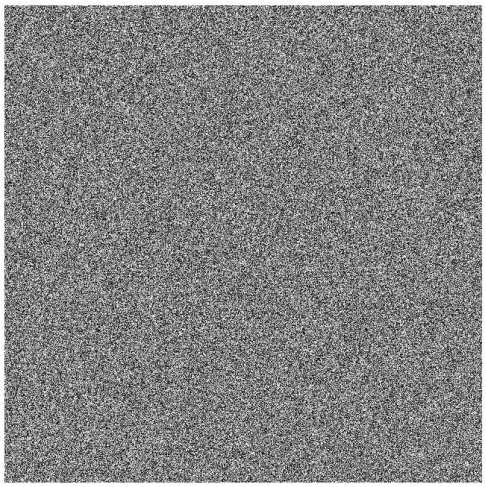


Figure 5: First layer weights after initialization

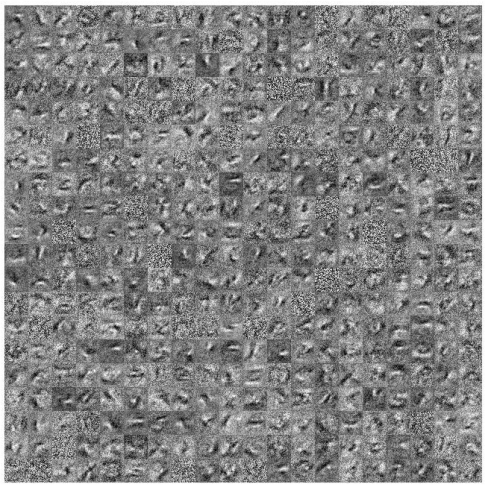


Figure 6: First layer weights after training for 30 epochs

From the above visualizations, it is clear that the weights are learning to detect features in the image. Initial weights when visualized do no show any pattern. Learned weights when visualized show show that they have started to learn features like edges on the image.
The best accuracy obtained after 30 epochs on test set is **88.69 %**
The loss obtained after 30 epochs on the test set is **0.4701**

- **Q3.1.4 Confusion Matrix**

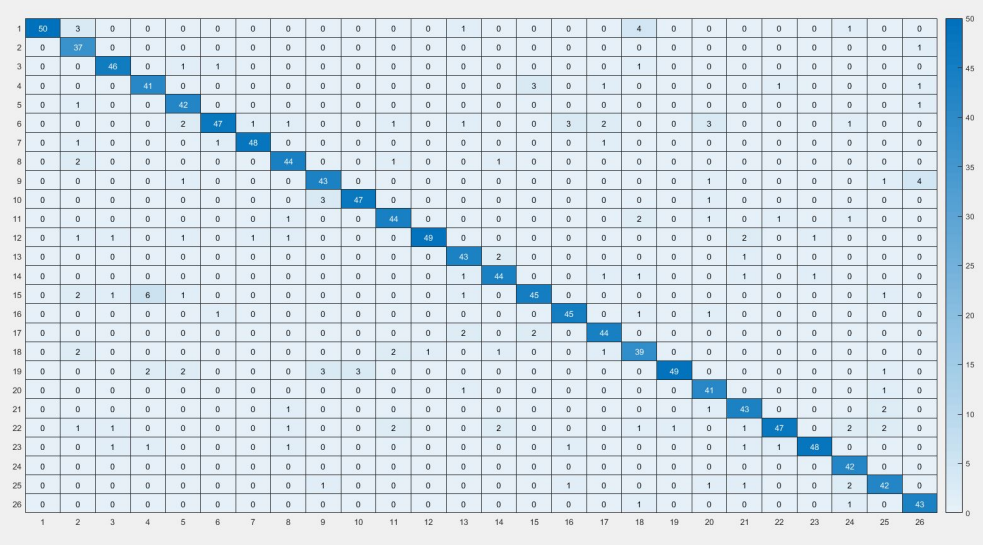


Figure 7: Confusion Matrix after 30 epochs

The top two pairs which are commonly confused are:
1. **D** and **O** : It is clear that these letters resemble each other and will often be confused by the network.
2. **R** and **A** : These two pairs also resemble each other. Hence are often confused.

- **Q3.2.1 Accuracy and loss plots**

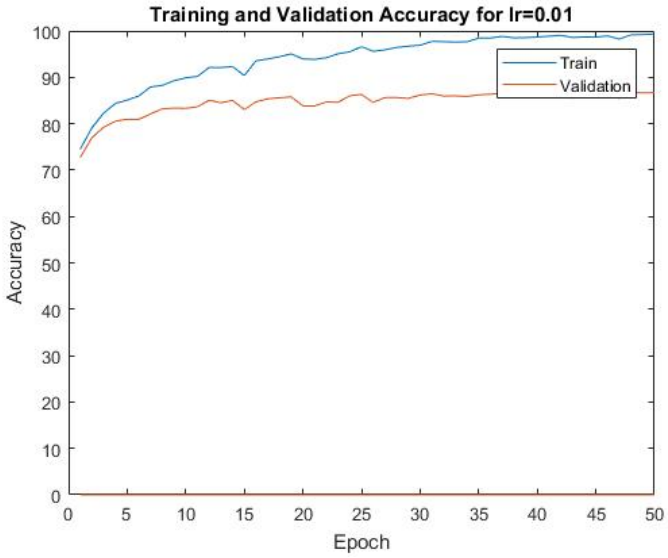


Figure 8: Training and Validation Accuracy for learning rate 0.01 after 50 epochs

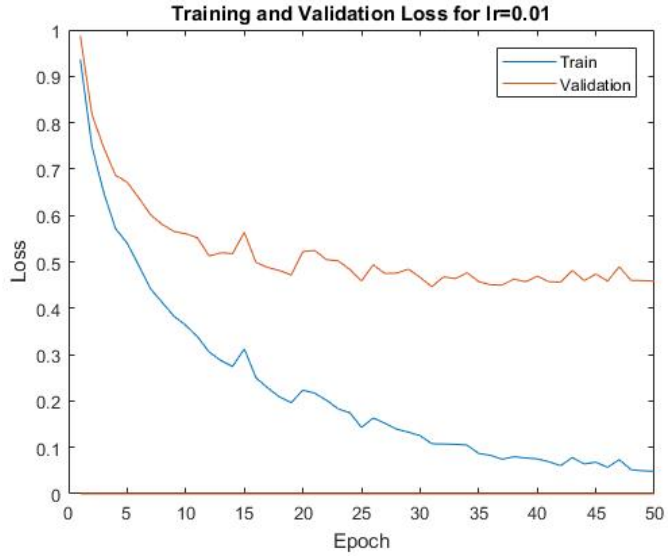


Figure 9: Training and Validation Loss for learning rate 0.01 after 50 epochs

- **Q3.2.2 Accuracy,loss and visualization**

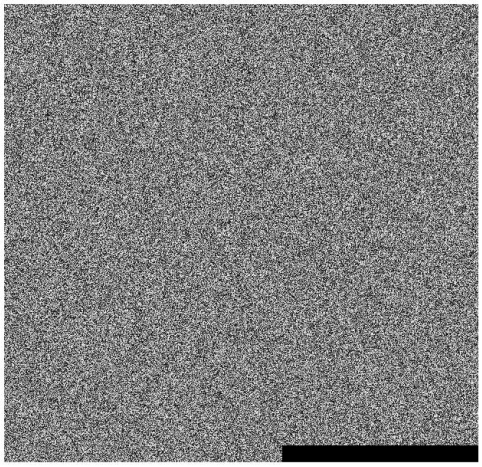


Figure 10: First layer weights after initialization

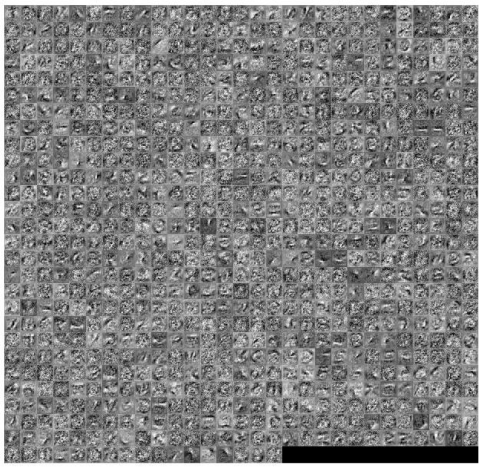


Figure 11: First layer weights after training for 50 epochs

From the above visualizations, it is clear that the weights are learning to detect features in the image. Initial weights when visualized do no show any pattern. Learned weights when visualized show that they have started to learn features like edges on the image.
The best accuracy obtained after 50 epochs on test set is **85.611 %**
The loss obtained after 50 epochs on the test set is **0.4843**

- **Q3.2.3 Confusion Matrix**

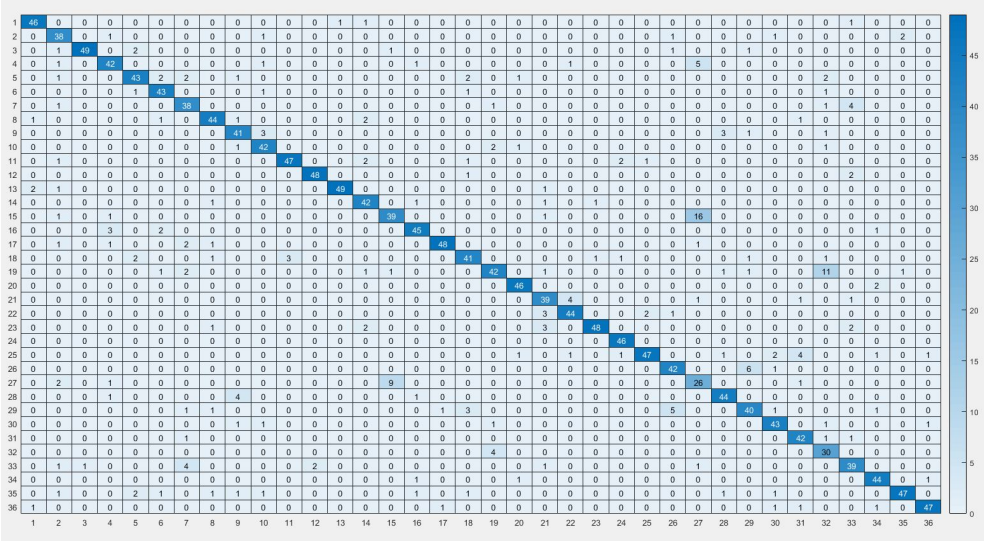


Figure 12: Confusion Matrix after 50 epochs

The top two pairs which are commonly confused are:
1. **O** and **0** : It is clear that letter O and digit 0 letters resemble each other and will often be confused by the network.
2. **S** and **5** : Letter S and digit 5 also resemble each other and are often confused

Introducing more classes has caused the confusion matrix to be spread out. More classes are now mis-classified.

- **Q4.1 Failure Cases**

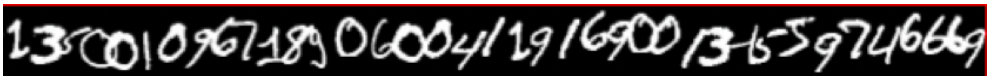


Figure 13: Failure case 1

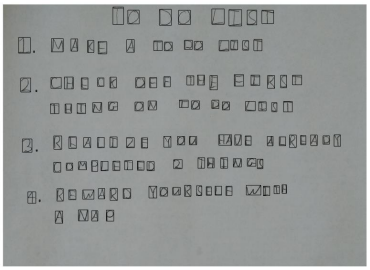
Figure 14: Failure case 2

Two big assumptions that the method makes are that bounding boxes will not overlap and each character is fully connected. As such, 2 failure cases that can arise out of them are:

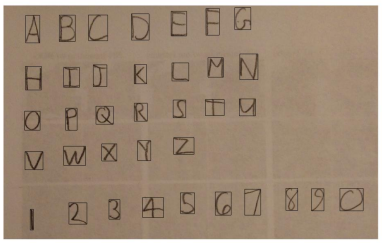
Case 1: When the characters cannot be separated by bounding boxes, i.e., there is some overlap between the characters. Fig above shows this case. It can be encountered if the characters are handwritten.

Case 2: When a single character is not fully connected. For example, lower case **i** has a dot on the top. Given approach will end up having 2 bounding boxes for the same character

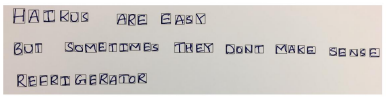
- **Q4.3 Bounding boxes on images**



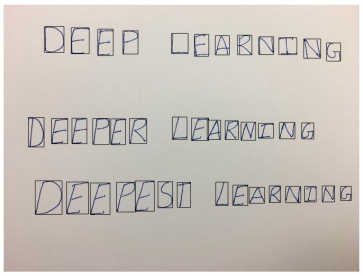
(a) Image1



(b) Image 2



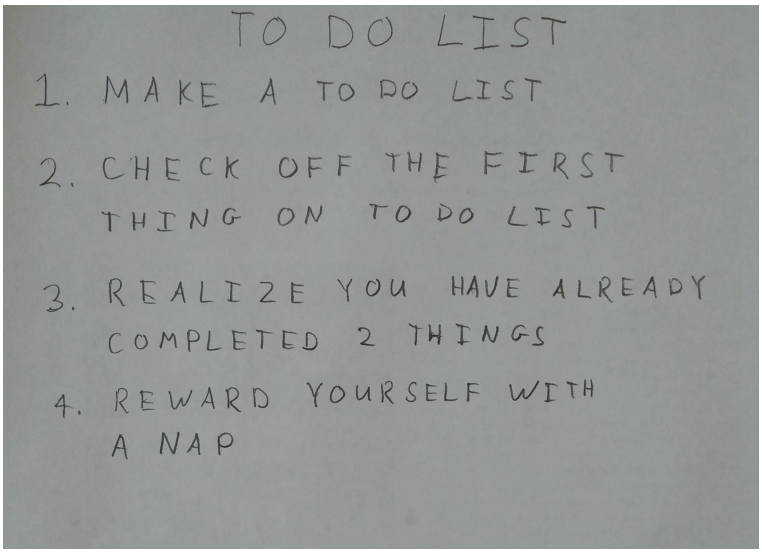
(c) Image3



(d) Image 4

Figure 15: Result of finding letters on different images

- **Q4.5 Extracting image Text Results**

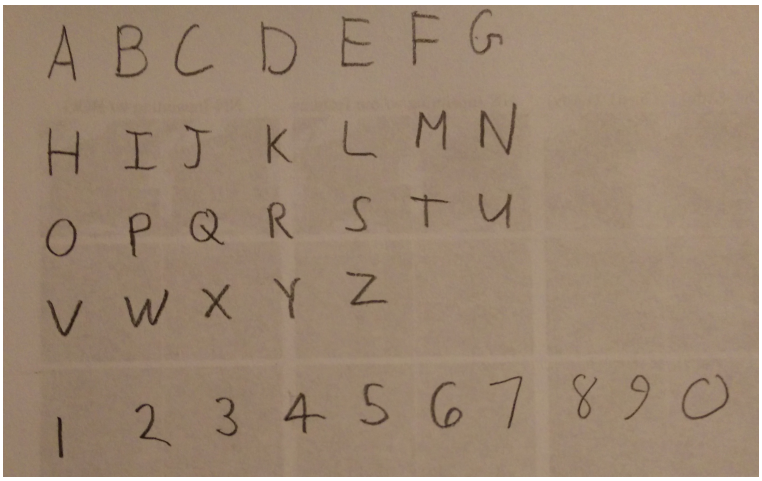


(a) Image

```
TC D0 LIST
L NAKE A T0FD LIST
2 CHECK 0FF YHE FIRST
TPING 0N TODO LIST
3 RFALIZE YDU HVE ALKRADY
C0MPLEIKD 2 YHINGS
9 R8WARD YOU8SELF WIYH
A NAP
```

(b) Extracted Text

Figure 16: Image 1 and the extracted text

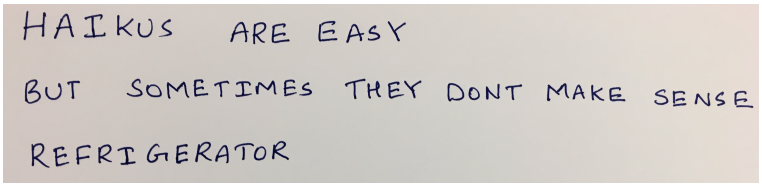


(a) Image

```
A B C D E F G
H I J K L M N
O P Q K S T V
V W X Y Z
1 Z 3 G S G Y 8 9 D
```

(b) Extracted Text

Figure 17: Image 2 and the extracted text

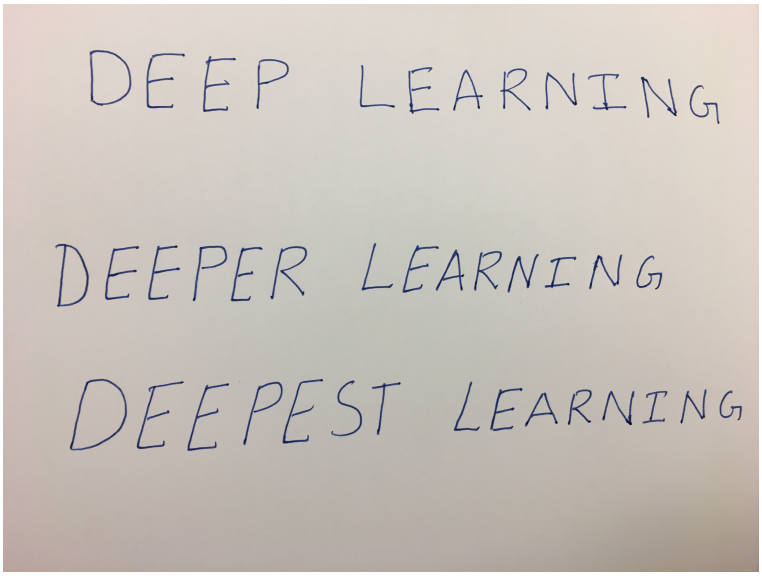


(a) Image

```
HAIKUS ARE EASY
3UT SCMETIMES THEY DONT MAKE SENGE
REFRIGERATOR
```

(b) Extracted Text

Figure 18: Image 3 and the extracted text



(a) Image

```
CE E F CEA RMINQ
DFETF9 LEAKNING
DEEFFST LEARNING
```

(b) Extracted Text

Figure 19: Image 4 and the extracted text