16-720B Computer Vision: Homework 5

Vigneshram Krishnamoorthy - Andrew ID: vigneshk

November 2017

1 Overview

1.1 Answer to Q1.1.1 - ReLU

- The ReLU activation function is $\boldsymbol{g}_{ReLU}(a) = max(0, a)$, and the sigmoid activation function is given by $\boldsymbol{g}(a) = 1/(1 + e^{-a})$.
- The ReLU activation function has a constant gradient of 1 if $a > 0$ and the gradient is 0 if $a < 0$. Whereas, the gradient of the sigmoid activation function is $\boldsymbol{g}(a)(1 - \boldsymbol{g}(a))$ whose values are very close to zero when a is either very large or very small, leading to the problem of **vanishing gradients**, slowing down the learning process significantly. ReLU hence supports a consistent learning speed.
- ReLU also speeds up the training process as it creates a sparse representation, since it outputs zeros whenever $a < 0$, making learning faster.

1.2 Answer to Q1.1.2 - Linear Activation Function

We have a neural network with input dimension N and output dimension C, T hidden layers and a linear activation function $\boldsymbol{g}$. The intermediate output of the first hidden layer will be $\boldsymbol{h}^{(1)} = \boldsymbol{g}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$, the second layer will be, $\boldsymbol{h}^{(2)} = \boldsymbol{g}(\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)})$. The final output will be $\boldsymbol{y} = \boldsymbol{o}(\boldsymbol{W}^{(T)}\boldsymbol{h}^{(T-1)} + \boldsymbol{b}^{(T)})$ where $\boldsymbol{h}^{(T-1)} = \boldsymbol{g}(\boldsymbol{W}^{(T-1)}\boldsymbol{h}^{(T-2)} + \boldsymbol{b}^{(T-1)})$. We can consider the biases as a form of weight, with the input being 1. Hence it can be appended to the weights. Let's call the new weights with the appended biases as $\hat{\boldsymbol{W}}$

$$\boldsymbol{y} = \boldsymbol{o}(\hat{\boldsymbol{W}}^{(T)}\boldsymbol{g}(\hat{\boldsymbol{W}}^{(T-1)}...\boldsymbol{g}(\hat{\boldsymbol{W}}^{(1)}[\boldsymbol{x} \quad \boldsymbol{1}])))$$

Since $\boldsymbol{g}$ is a linear activation function, the composition of the linear transformations in all the T hidden layers is also a linear transformation, which can also be obtained by using $T^{'}$ ($T^{'} \neq T$) number of hidden layers (can even be collapsed to a single linear function). For example it can be collapsed to,

$$\boldsymbol{y} = \boldsymbol{o}(\tilde{\boldsymbol{W}})[\boldsymbol{x} \quad \boldsymbol{1}]$$

where $\tilde{\boldsymbol{W}}$ is the composition of the weights.
Hence, the number of hidden layers has no effect on the representation capability of the network.

2 Implement a Fully Connected Network

2.1 Answer to Q2.1.1 - Zero or one initialization

- The activations in the i^{th} layer is given by $\boldsymbol{a}^{(i)} = \boldsymbol{W}^{(i)}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}$.
- Case I: If the weights and biases $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ are initialized to zeros, then the activations $\boldsymbol{a}^{(i)}$'s will also be zeros, leading to zero gradients (in the case of ReLU) or constant gradients for other activation functions. Hence, the update to the weights will keep their values same. Therefore, it is a very bad idea to initialize a network with all zeros.

- Case II: If the weights and biases $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ are initialized to ones or some other constant value, then the weights branching out from each input of the previous layer to the current layer will be updated to the same value in all iterations of gradient descent.

$$\frac{\partial L}{\partial \boldsymbol{W}^{(i)}} = \frac{\partial L}{\partial \boldsymbol{g}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{a}^{(i)}} \frac{\partial \boldsymbol{a}^{(i)}}{\partial \boldsymbol{W}^{(i)}}$$

  In the above chain-rule, the gradients will be a scaled version of the input from the previous layer. This makes all the units in hidden layers learn the same feature. The outputs (say with a softmax activation function for the output layer) will have the same probability for all output nodes. Since we want the units in our network to learn different features/representations, it is a bad idea to initialize the parameters of the network with the same constant value.

2.2 Answer to Q2.1.3 - Describe initialization strategy

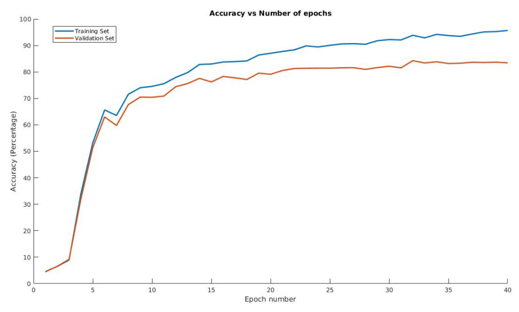
- The weights are initialized as $\boldsymbol{W}^{(i)} = 0.05randn(N_{i-1}, N_i)/\sqrt{(N_{i-1})}$. In order to normalize the variances of the output we divide by the square root of the number of inputs of the previous layer. The multiplicative factor is to initialize the weights close to zero. The randn function (normal distribution of random numbers) creates assymetry in the weights leading the network to learn diverse representations.
- The biases are initialized to zeros since the assymetry in activations is already created by the weights as described above.
- Reference: http://cs231n.github.io/neural-networks-2/#init

2.3 Answer to Q2.4.1 - Batch vs. stochastic gradient descent

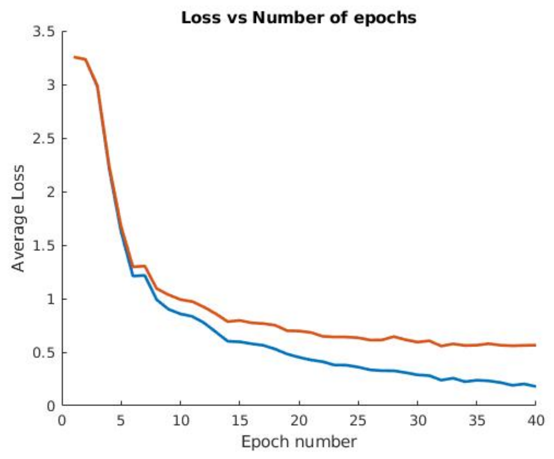
- **Stochastic Gradient Descent**: This technique uses a single random data sample to update the parameters in each iteration. Hence this process takes more number of iterations. However it takes lesser number of epochs, as the parameters are tuned more often, for every single sample in the training set. SGD is computationally more tractable since even for a large dataset, the whole of the dataset doesn't have to be held in RAM as we load sample by sample. However, the result may be a close approximation as the loss may not be minimized as much as in Batch Gradient Descent.
- **Batch Gradient Descent**: This technique uses the whole training set to compute the gradients. Hence it is computationally more expensive. The batch size is very big and hence it takes less number of iterations to converge, but relatively more epochs. This method is computationally not feasible for big datasets. Theoretically, this method gives us a optimum which is deterministic in nature.

3 Training Models

3.1 Answer to Q3.1.2 - Training plots and comments on nist26 data

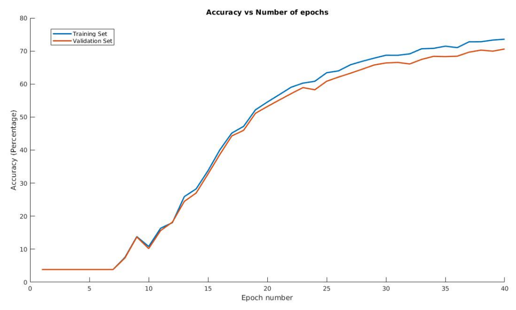


(a) Accuracy vs Number of Epochs

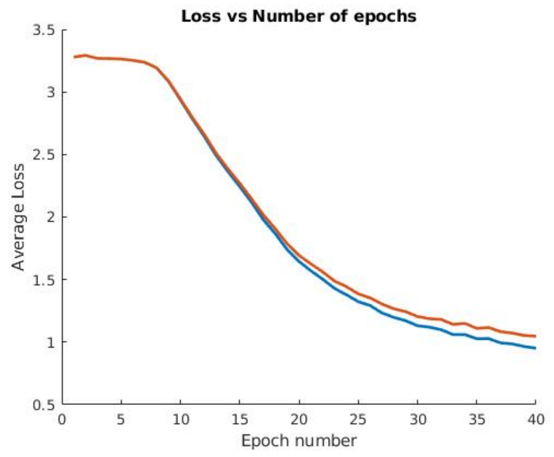


(b) Loss vs Number of Epochs

Figure 1: Learning Rate = 0.01



(a) Accuracy vs Number of Epochs



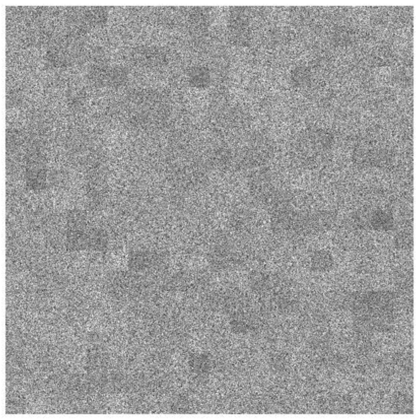
(b) Loss vs Number of Epochs

Figure 2: Learning Rate = 0.001

- From the above 4 plots, we can clearly see the difference in the rate of convergence to the final value. The learning rate of 0.01 is seen to converge much faster than that of 0.001, since the latter proceeds towards the optimum point at a much slower rate.
- A faster learning rate means that you take bigger steps towards the goal, thus requiring lesser number of iterations to converge to the optimal solution. The opposite happens when we have a slower learning rate. However, we should be careful not to set the learning rate too large, since it can skip the global optimum, jumping back and forth around it.
- Sometimes, people prefer to vary the learning rates through the course of the learning process. The initialize with a faster learning rate, and once the solution reaches close to the global optimum, the learning rate is lowered.

The final accuracy of the best network (learning rate = 0.01) on the test set is **86.31 %**.

3.2 Answer to Q3.1.3 - Network test set performance and weights visualization


(a) First layer weights before training

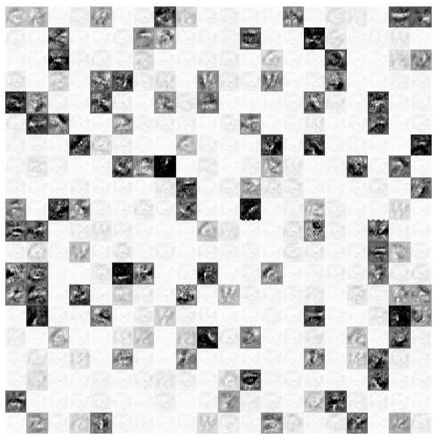

(b) First layer weights after training

Figure 3: Visualization of weights for the best network with learning rate = 0.01

The final accuracy of the best network (learning rate = 0.01) on the test set is **86.31 %** and the average cross-entropy loss is **0.5322**.

The initial weights are randomly initialized and hence the visualization looks very noisy. The learned weights however, seem to have some patterns representing the input data. We can see the edge information of different letters in some of the weights.

3.3 Answer to Q3.1.4 - Confusion matrix image visualization and comments

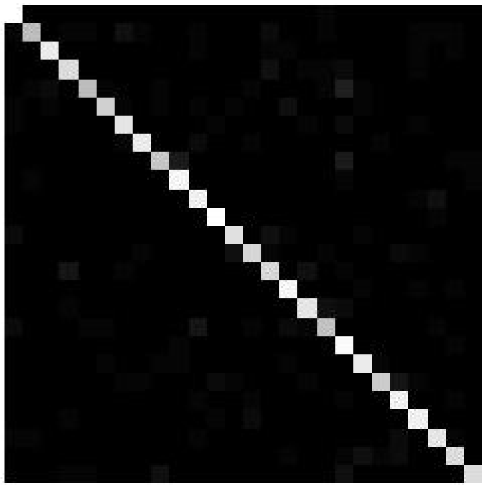
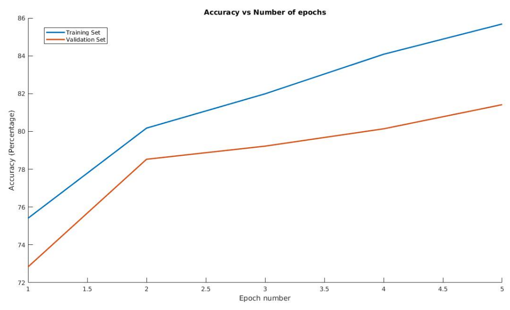


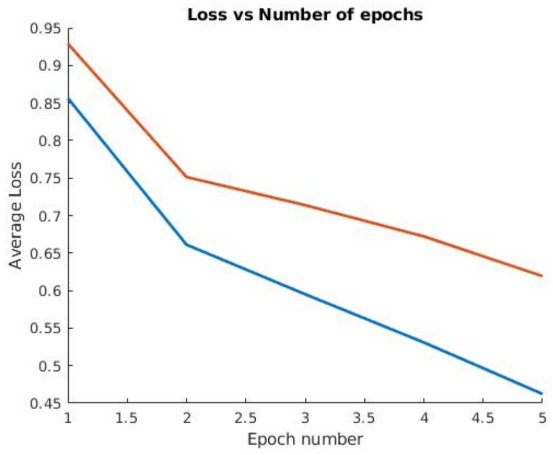
Figure 4: Visualization of the confusion matrix for the best model as a 26 × 26

1. **'M'** is confused with **'N'**,
2. **'O'** gets confused with **'D'**,
3. **'R'** gets confused with **'K'**
4. This is due to fact that these pairs of letters when write have very close structural representation and can even confuse human beings, when the writing is sloppy. If the distinguishing features in between the words are not well learnt, the network can easily confuse between the words.

3.4 Answer to Q3.2.1 - Training plot for fine tuning



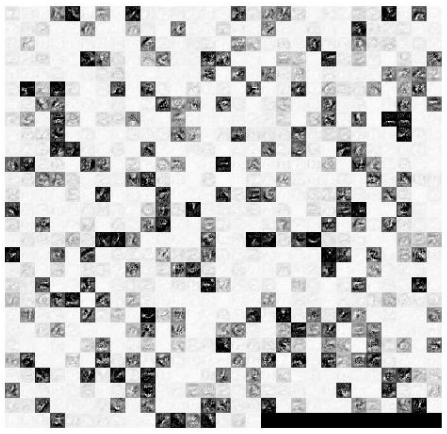
(a) Accuracy vs Number of Epochs



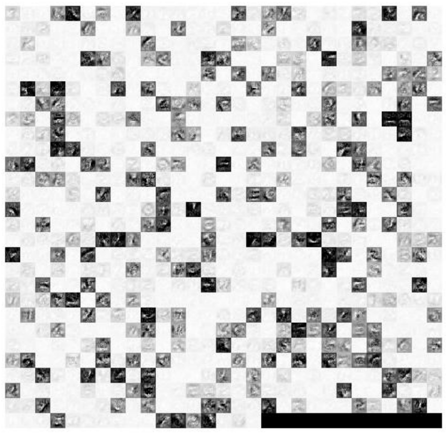
(b) Loss vs Number of Epochs

Figure 5: Learning Rate = 0.01

3.5 Answer to Q3.2.2 - Fine tuned weights visualization



(a) First layer weights before training



(b) First layer weights after training

Figure 6: Visualization of weights for the best network with learning rate = 0.01

We see from the weight visualization that there is not much difference before and after training. This is because the initial weights are already obtained from a trained network for 26 out of the 36 classes. On close observation however, we can notice some changes in between the two, mainly as weights update themselves to accommodate the other 10 classes. Since the initial weights are very close to the optimal weights, we need very less number of epochs to converge.

- Accuracy - **82.28 %**
- Average Cross-entropy loss - **0.6119**

3.6 Answer to Q3.2.3 - Confusion matrix image for fine tuned network

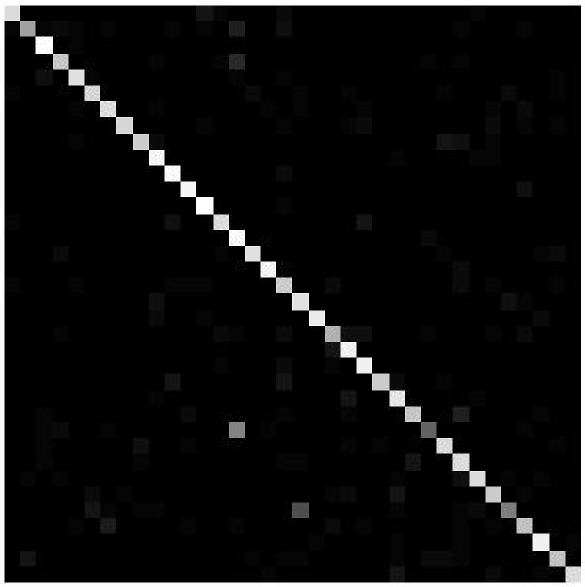
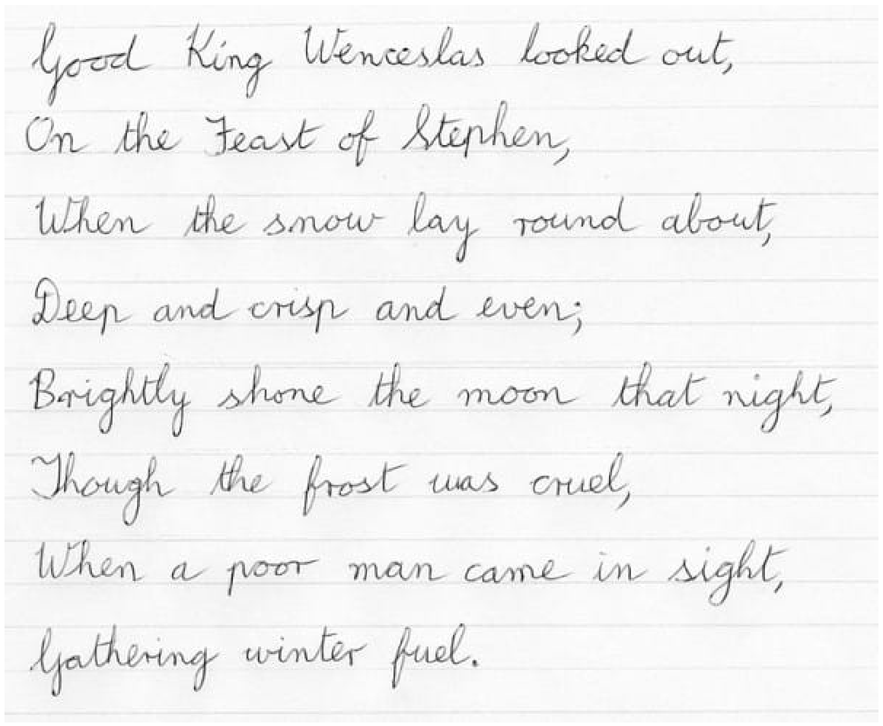


Figure 7: Visualization of the confusion matrix for the best model as a 36×36

1. **'0'** is confused with **'O'** the most.
2. **'5'** and **'S'**, **'2'** and **'Z'** also get confused with a high frequency.
3. The introduction of the extra classes have brought about increased confusion leading to reduced accuracy of the system. This is especially the case, since most of the newly added classes of digits have close representations in the existing English alphabets.

4 Extract Text from Images

4.1 Answer to Q4.1 - Failure cases for sample character detection algorithm



Good King Wenceslas looked out,
On the Feast of Stephen,
When the snow lay round about,
Deep and crisp and even;
Brightly shone the moon that night,
Though the frost was cruel,
When a poor man came in sight,
Gathering winter fuel.

Figure 8: Words connected in a sentence when a person writes cursive writing.

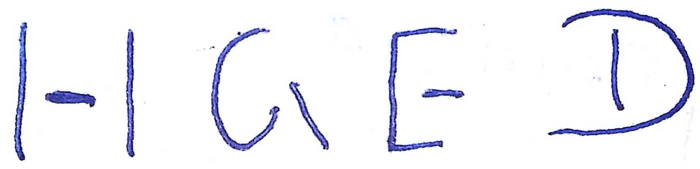


Figure 9: Disconnected representation of letters.

1. The sample method assumes that each character is fully-connected. However, from Figure 9 we can see that characters can be written without fully connecting. Incidentally this happens in 03_haiku.jpg when the 'E' is detected as two objects instead of one. In this case, it can respond to non-letters.

2. This method also assumes that none of the characters in a sentence are connected together with one another. However, most texts in the real-world have one or more letters in a word connected together. Figure 8 illustrates such an example. Our connected components approaches would fail badly in such cases where each letter is connected to its neighbour. In this case, it may miss valid letters (might put a bounding box over the entire word considering it as a single character).

4.2 Answer to Q4.3 - Character detection results on each of the images in images/

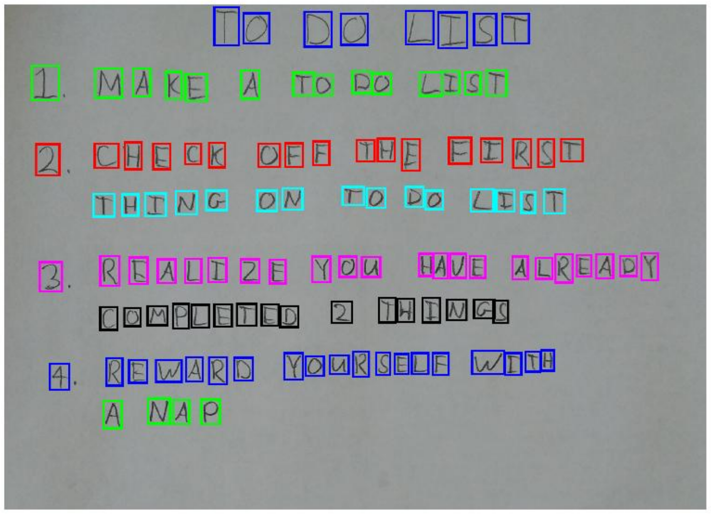


Figure 10: Bounding box output for 01_list.jpg

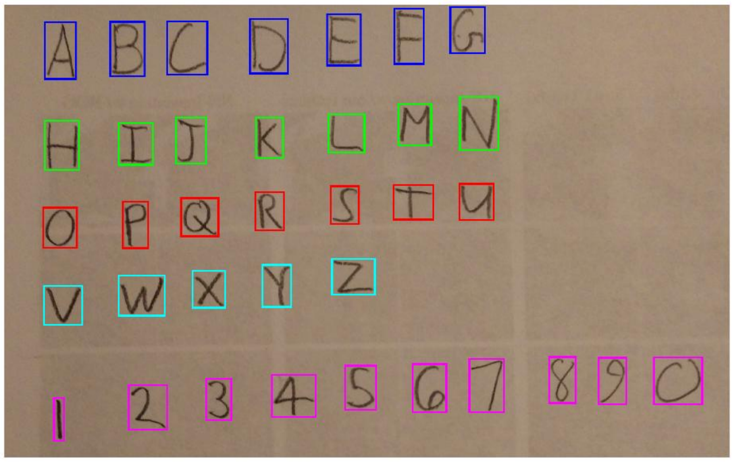


Figure 11: Bounding box output for 02_letters.jpg

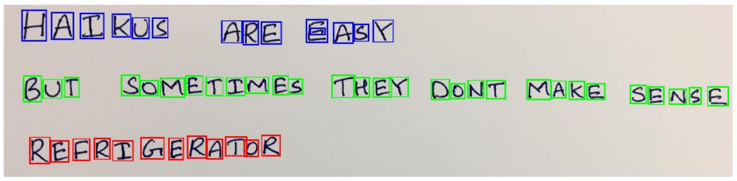


Figure 12: Bounding box output for 03_haiku.jpg

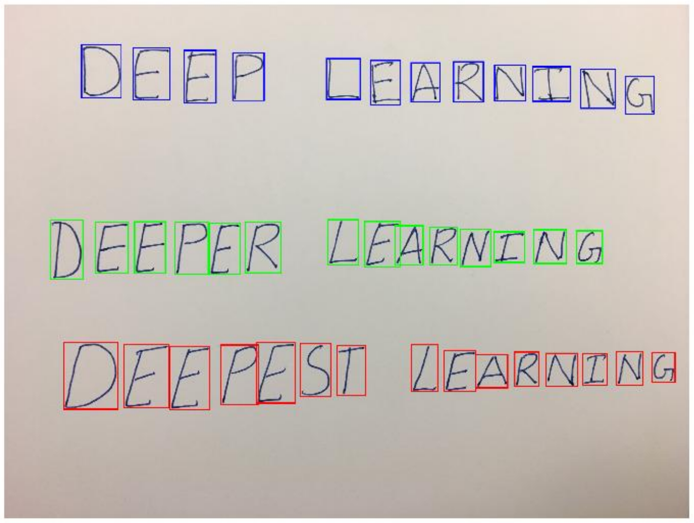


Figure 13: Bounding box output for 04_deep.jpg

4.3 Answer to Q4.5 - Extracted text from each of the images in images/

Text output for 01_list.jpg

```
TO DO LIST
L MAKE A TODO LIST
2 CHECK OFF THE FIRST
THING ON TODO LIST
3 R2ALIZE YOU HRVE ALREADY
COMPLET2D 2 THINGS
4 REWARD YOURSELF WITH
A NAP
```

Text output for 02_letters.jpg

```
A BC D E FG
H IJ K L MN
Q P Q R S TU
V WX Y Z
8 Z 3 G S G7 Y7Q
```

Text output for 03_haiku.jpg

```
HAIKUS ARE EMASY
BUT SOMETIMES TREX DONT MAK2 SENGE
REFRIGERATOR
```

Text output for 04_deep.jpg

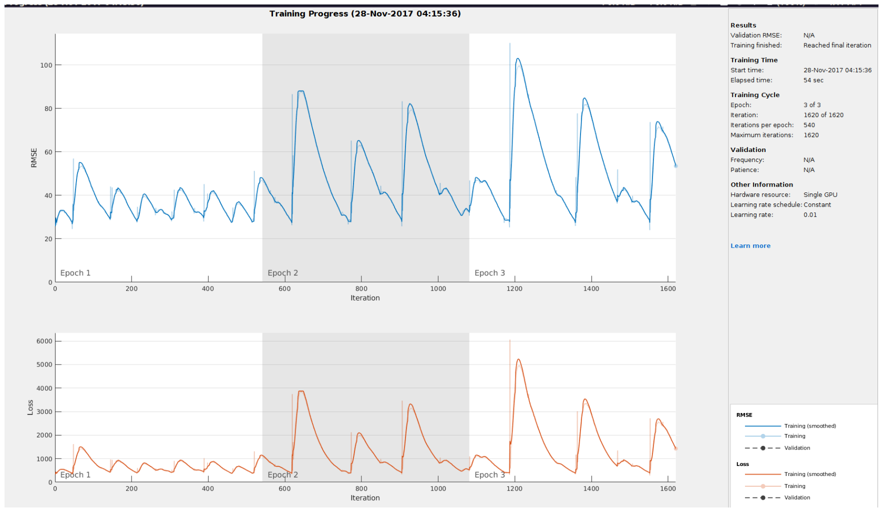
```
DEEP LEARMINQ
DEETER LEAKNING
SFEFFST LEARNING
```

5 Image Compression with Autoencoders

5.1 Answer to Q5.1 - Fixing define autoencoder.m

define_autoencoder.m does not have any non-linear activations. This can significantly affect performance since the whole network can only learn linear represntations. Hence, I added a Leaky ReLU activation function after each convolution and transposed convolution layer. Batch normalization, which is a weak form of regularization, is also done after every layer.

5.2 Answer to Q5.2.1 - Train Autoencoder



Results
Validation RMSE: N/A
Training finished: Reached final iteration

Training Time
Start time: 28-Nov-2017 04:15:36
Elapsed time: 54 sec

Training Cycle
Epoch: 3 of 3
Iteration: 1620 of 1620
Iterations per epoch: 540
Maximum iterations: 1620

Validation
Frequency: N/A
Patience: N/A

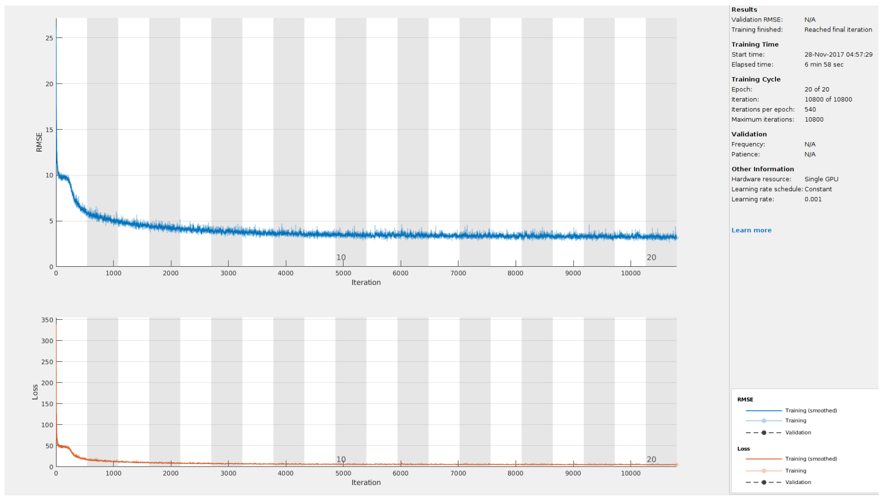
Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.01

Learn more

Figure 14: Training loss curve with the provided default settings.

The training loss curve for 3 epochs with the given default settings is seen to be oscillating. This can be attributed to the high initial learning rate of 10−2.

5.3 Answer to Q5.2.2 - Improve Training Performance



Results	
Validation RMSE:	N/A
Training finished:	Reached final iteration
Training Time	
Start time:	28-Nov-2017 04:57:29
Elapsed time:	6 min 58 sec
Training Cycle	
Epoch:	20 of 20
Iteration:	10800 of 10800
Iterations per epoch:	540
Maximum iterations:	10800
Validation	
Frequency:	N/A
Patience:	N/A
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more

Figure 15: Training loss curve after the adjustments were made.

The adjustments made to the hyper-parameters are as follows.

1. The learning rate is decreased to 10^{-3}
2. The weight initialization was done using the method suggested in the paper 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification' by He et. al.

5.4 Answer to Q5.3.1 - Evaluate Autoencoder

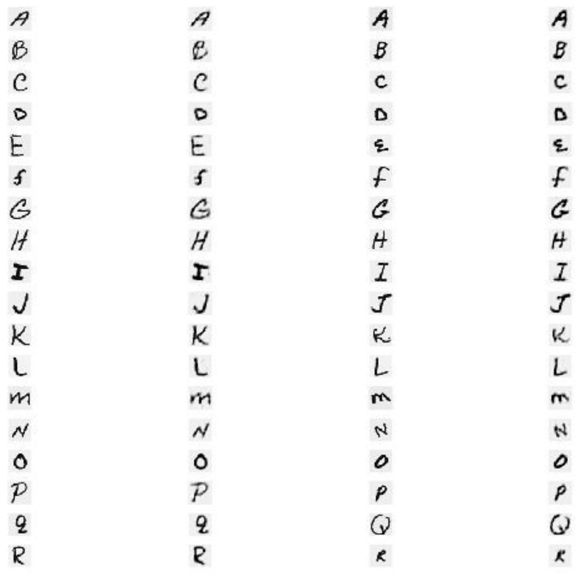


Figure 16: The images and their reconstruction of the first 18 classes using autoencoders. The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

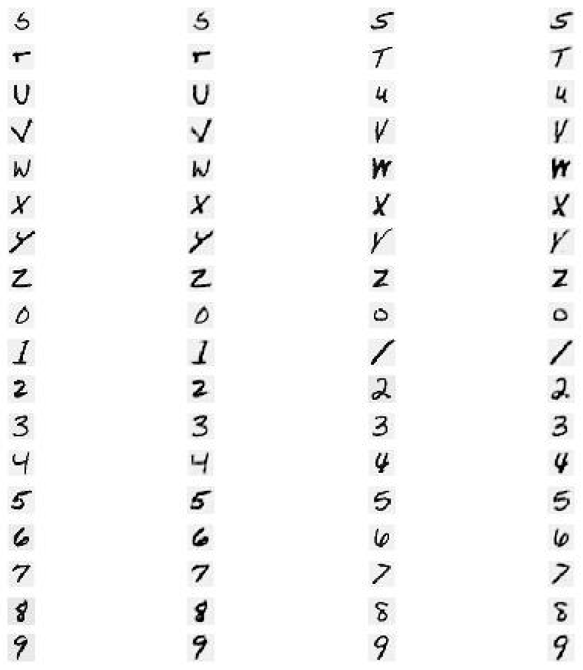


Figure 17: The images and their reconstruction of the next 18 classes using autoencoders. The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

We observe that high frequency features have been partially lost in the compression process. We can hence notice some blurring effect on the reconstructed image.

5.5 Answer to Q5.3.2 - Evaluate Autoencoder with PSNR

The average PSNR obtained using the improved autoencoder is **21.8765** dB.

5.6 Answer to Q5.4.1 - Projection Matrix

The size of the projection matrix is 1024×64. The rank of the projection matrix is 64.

5.7 Answer to Q5.4.2 - Evaluate PCA

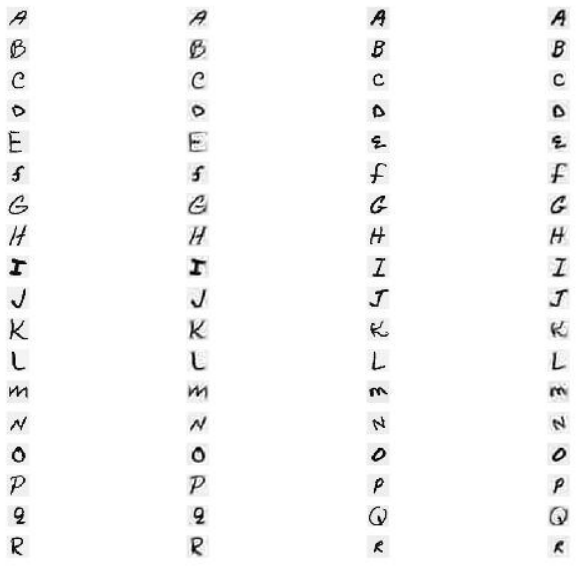


Figure 18: The images and their reconstruction of the first 18 classes using PCA. The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

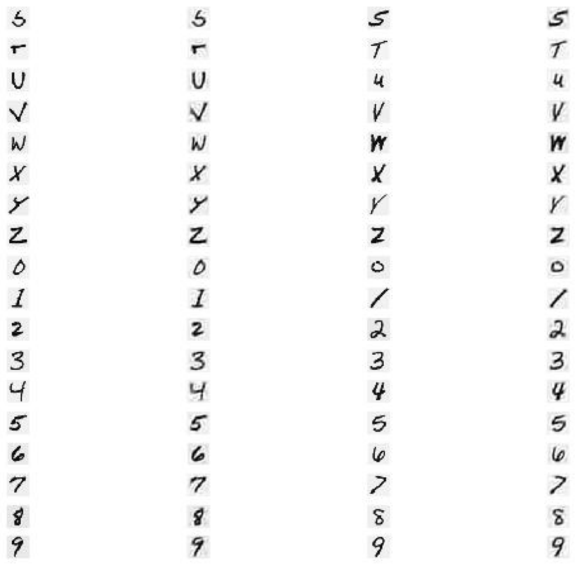


Figure 19: The images and their reconstruction of the next 18 classes using PCA. The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

We observe that high frequency features have been partially lost (more than that by using autoencoders) in the compression process. We can hence notice more blurring effect on the reconstructed image. We also see some slight changes in the background of the letters in the sense that it is not as consistent as given in the original image, which can tend to occur in linear transformations. Overall, we see that the autoencoder gives better reconstruction.

5.8 Answer to Q5.4.3 - Evaluate PCA with PSNR

The average PSNR from PCA is **19.0317** dB. This is lesser than the average PSNR obtained from autoencoder. The autoencoder performs much better due to the non-linear representations of the image it can capture in the given image, making the reconstruction more similar to the original image.

5.9 Answer to Q5.4.4 - Evaluate Autoencoder without correction

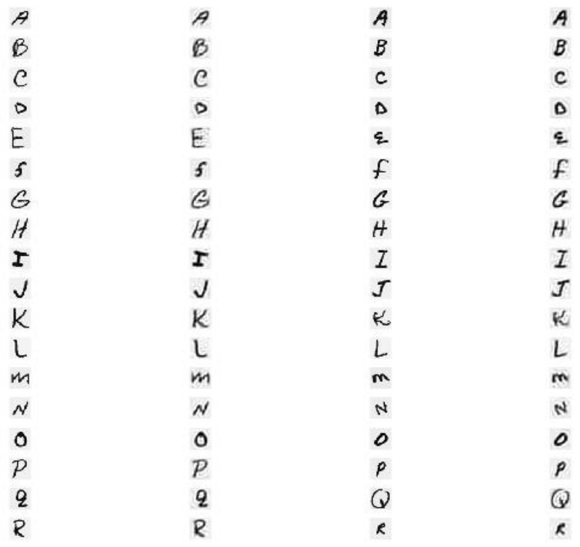


Figure 20: The images and their reconstruction of the first 18 classes using autoencoders without non-linear activation functions. The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

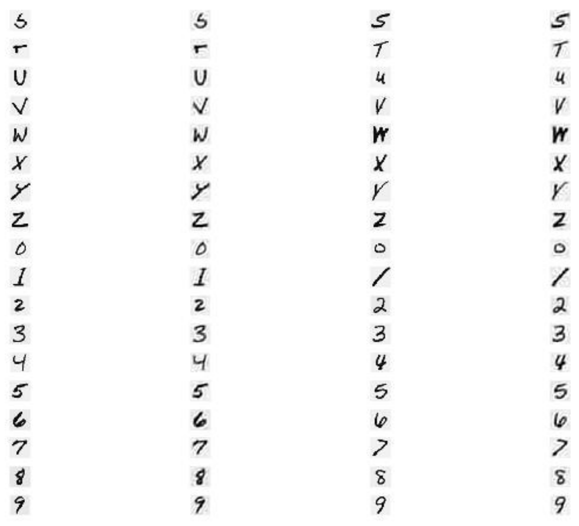


Figure 21: The images and their reconstruction of the next 18 classes using using autoencoders without non-linear activation functions.The first and third columns are the original test images and the second and fourth columns are their corresponding reconstructions.

The average PSNR from autoencoder with current settings, but without the corrections made in Q5.1 is **18.6774** dB. It is almost similar to the result obtained from PCA. This is since both methods are linear representations of the input data and cannot capture non-linearity during the compression. Hence, their performance are very similar.

For the autoencoder output without the non-linear activations, we observe that high frequency features have been partially lost (more than that by using autoencoders with leaky ReLU) in the compression process. We can hence notice more blurring effect on the reconstructed image. We also see some slight changes in the background of the letters in the sense that it is not as consistent as given in the original image, which can tend to occur in linear transformations.

5.10 Answer to Q5.4.5 - Number of Parameters

1. The number of learned parameters in PCA is **65536**
2. The number of learned paramters in autoencoders is **135537**.
3. The number of learned parameters in autoencoders is approximately twice the number of learned parameters in PCA.
4. This is because PCA uses the same parameters for both encoding and decoding whereas autoencoders have different weights for the encoding and decoding blocks, hence multiplying the number of learned parameters by two.
5. The difference in performance (autoencoder performing much better than PCA) is because of the non-linear representations that the network learns which can help represent the data better, also leading to better data compression.

5.11 Answer to Q5.5 - Comparison with JPEG-2000

The average PSNR from JPEG-2000 with compression ratio is **7.4558** dB. We can see that this is worse compared to the autoencoder. When visualized, we see that the compression has taken out a lot of the information from the images. This is since we have a high compression ratio of 16.

Hand-crafted algorithms like JPEG-2000 is used in real-life instead of autoencoders for standard image compression since

- Autoencoders are computationally more expensive (during training) and require lots of training data.
- The output of JPEG-2000 doesn't depend on the prior data. It has a standard algorithm which doesn't depend on the data distribution.
- In case of autoencoders, the compression is achieved by learning on a training set of data. Therefore, autoencoder outputs depend heavily on the training data distribution and cannot generalize across datasets. For example, when we train the encoder with English letters and test it with Chinese letters, it won't work. Biases in the training dataset can also affect the performance of the autoencoders. Hence auto-encoders are poor general purpose image compressors.
- Also, when we compress the image using autoencoders, we need the weights and biases to decompress the image. So one cannot retrieve the image without the availability of weights. This is not the case with JPEG since the parameters used for compression are constant across all systems.