16-720B Computer Vision: Homework 5
Neural Networks for Recognition

Heethesh Vhavle
Andrew ID: hvhavlen

22 November 2018
(Using **THREE LATE DAYS**)

1 Theory

1.1 Softmax Translation Invariance

The proof is as follows. The term e^c can be factored out in both the numerator and the denominator.

$$\begin{aligned}
\text{softmax}\,(x_i) &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\
\text{softmax}\,(x_i + c) &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\
\text{softmax}\,(x_i + c) &= \frac{e^{x_i}e^{c}}{e^c\sum_j e^{x_j}} \\
\text{softmax}\,(x_i + c) &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\
\therefore \text{softmax}\,(x_i) &= \text{softmax}\,(x_i + c)
\end{aligned} \tag{1}$$

By setting $c = -\max x_i$ or subtracting the maximum value exponent term, we get $e^0 = 1$ and all the other terms will be scaled between 0 and 1, thus avoiding large exponent terms which improves numerical stability and avoids overflows.

1.2 Softmax Properties

1.2.1

Every element, for $d > 2$, will be in the interval (0, 1) and the sum over all elements will be equal to 1.

1.2.2

One could say that *"Softmax takes an arbitrary real valued vector x and turns it into a vector of size (d, 1) with each element of the vector in the interval (0, 1) and sum over all elements equal to 1."*

1.2.3

The first step is the exponential which increases the probability of the biggest score and decreases the probability of the lowest score exponentially, which is much closer to the argmax compared just taking the mean. The second step is to sum up all the probabilities and the final step is to divide all the probabilities by the sum to ensure that the output is in the interval (0, 1) and the sum of all probabilities is 1.

1.3 Multi-Layer Neural Networks

Let us consider this network without any activation functions. Let H_i be the output at every layer and let n be total number of layers.

$$\begin{aligned}
H_i &= W_i H_{i-1} + b_i \\
H_n &= W_n H_{n-1} + b_n \\
H_{n-1} &= W_{n-1} H_{n-2} + b_{n-1} \\
H_n &= W_n (W_{n-1} H_{n-2} + b_{n-1}) + b_n \\
H_n &= W_n W_{n-1} H_{n-2} + (W_n b_{n-1} + b_n)
\end{aligned} \tag{2}$$

Similarly, we can express this network for all layers till the input layer as a linear combination of product of the weights added with some bias term, which is essentially the same as linear regression.

$$H_n = W_n W_{n-1} W_{n-2} \dots W_1 X + bias \tag{3}$$

Thus, we can see that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

1.4 Sigmoid Derivative

Differentiating using chain rule, we get the following.

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} = \left(1+e^{-x}\right)^{-1} \\
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\left(1+e^{-x}\right)^{-1}\right) \\
\frac{d}{dx}\sigma(x) &= -1\left(\left(1+e^{-x}\right)^{(-2)}\right)\left(e^{-x}\right)(-1) \\
\frac{d}{dx}\sigma(x) &= \frac{\left(e^{-x}\right)}{\left(1+e^{-x}\right)^2}
\end{aligned} \tag{4}$$

We can simplify the above, by adding and subtracting 1 in the numerator and rewrite it in terms of $\sigma(x)$.

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{(1+e^{-x}-1)}{(1+e^{-x})^2} \\
\frac{d}{dx}\sigma(x) &= \frac{(1+e^{-x})}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
\frac{d}{dx}\sigma(x) &= \frac{1}{(1+e^{-x})}\left(1 - \frac{1}{(1+e^{-x})}\right) \\
\therefore \frac{d}{dx}\sigma(x) &= \sigma(x)(1-\sigma(x))
\end{aligned} \tag{5}$$

1.5 Gradient of Loss Function

We have the following.

$$\begin{aligned}
y &= x^T W + b \\
y_j &= x_1 W_{1j} + b_j + x_2 W_{2j} + b_j + \dots + x_m W_{mj} + b_j
\end{aligned} \tag{6}$$

We can represent the derivatives with respect to W as follows.

$$\begin{aligned} \frac{\partial J}{\partial W} &= \frac{\partial J}{\partial y}\frac{\partial y}{\partial W} \\ where, \frac{\partial J}{\partial y} &= \delta \in \mathbb{R}^{k\times 1} \\ \frac{\partial J}{\partial W} &= \delta\frac{\partial y}{\partial W} \end{aligned} \tag{7}$$

Similarly, we can represent the derivatives with respect to x as follows.

$$\begin{aligned} \frac{\partial J}{\partial x} &= \frac{\partial J}{\partial y}\frac{\partial y}{\partial x} \\ \frac{\partial J}{\partial x} &= \delta\frac{\partial y}{\partial x} \end{aligned} \tag{8}$$

The last term above, $\frac{\partial y}{\partial W}$ and $\frac{\partial y}{\partial x}$ can be written in the matrix form as below.

$$\begin{aligned} \frac{\partial y}{\partial vec(W)} &= \begin{bmatrix} \frac{\partial y_1}{\partial W_{11}} & \frac{\partial y_1}{\partial W_{21}} & \cdots & \frac{\partial y_1}{\partial W_{m1}} \\ \frac{\partial y_1}{\partial W_{12}} & \frac{\partial y_1}{\partial W_{22}} & \cdots & \frac{\partial y_1}{\partial W_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial W_{1n}} & \frac{\partial y_1}{\partial W_{2n}} & \cdots & \frac{\partial y_1}{\partial W_{mn}} \end{bmatrix} \begin{bmatrix} \frac{\partial y_2}{\partial W_{11}} & \frac{\partial y_2}{\partial W_{21}} & \cdots & \frac{\partial y_2}{\partial W_{m1}} \\ \frac{\partial y_2}{\partial W_{12}} & \frac{\partial y_2}{\partial W_{22}} & \cdots & \frac{\partial y_2}{\partial W_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_2}{\partial W_{1n}} & \frac{\partial y_2}{\partial W_{2n}} & \cdots & \frac{\partial y_2}{\partial W_{mn}} \end{bmatrix} \cdots \\ \frac{\partial y}{\partial x} &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \end{aligned} \tag{9}$$

Let us take equation 6 and do the partial derivative for just the first elements with respect to W_{1j} and x_1. We get the following.

$$\begin{aligned} \frac{\partial y_1}{\partial W_{11}} &= x_1 \\ \frac{\partial y_1}{\partial x_1} &= W_{11} \end{aligned} \tag{10}$$

Now, doing the same for all the elements and putting them in the matrix form, we get the following.

$$\frac{\partial y}{\partial vec(W)} = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & \cdots & 0 \\ x_1 & x_2 & \cdots & x_m \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \cdots \tag{11}$$

Upon simplification and rearrangement of this sparse matrix, we get the following.

$$\therefore \frac{\partial J}{\partial W} = x^T\delta \tag{12}$$

$$\begin{aligned} \frac{\partial y}{\partial x} &= \begin{bmatrix} W_{11} & W_{21} & \cdots & W_{m1} \\ W_{12} & W_{22} & \cdots & W_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ W_{1n} & W_{2n} & \cdots & W_{mn} \end{bmatrix} = W^T \\ \therefore \frac{\partial J}{\partial x} &= \delta W^T \end{aligned} \tag{13}$$

1.6 Activation Functions

1.6.1

The sigmoid activation function maps all values, including large values to the interval [0, 1]. When we are training the network using gradient descent, if large changes lead to such small changes in the output, the gradient will be very small (for sigmoid, the interval of its derivative is [0, 0.25]). If sigmoid is used for successive layers and the layer outputs keep getting mapped to smaller and smaller regions successively, we end up having the vanishing gradient problem and the training might stall.

1.6.2

The output range for tanh is the interval [-1, 1] and output range for sigmoid is the interval [0, 1]. As per Yan LeCun's *Efficient BackProp* paper, convergence is usually faster if the average of each input variable over the training set is close to zero which is the case with tanh. The sigmoid activation function has the problem of saturating at 0 and 1 while tanh saturates at 1 and -1. So if the activity in the network during training is close to 0 then the gradient for the sigmoid activation function may go to 0. Furthermore, the derivative of the activations matter, which is explained below.

1.6.3

The maximum value of derivative of tanh is up to 1 compared to 0.25 with sigmoid, which leads to greater updates in the parameters during gradient descent and thus tanh is better than sigmoid considering the vanishing gradient problem, if parameters are not updated properly in multiple layers.

1.6.4

The derivation is as follows.

$$\begin{aligned}
\sigma(x) &= \frac{e^x}{1+e^x} \\
\sigma(2x) &= \frac{e^{2x}}{1+e^{2x}} \\
e^{2x} &= \frac{\sigma(2x)}{1-\sigma(2x)} \\
\tanh(x) &= \frac{1-e^{-2x}}{1+e^{-2x}} \\
\tanh(x) &= \frac{1-\frac{1}{e^{2x}}}{1+\frac{1}{e^{2x}}} \\
\tanh(x) &= \frac{1-\frac{1-\sigma(2x)}{\sigma(2x)}}{1+\frac{1-\sigma(2x)}{\sigma(2x)}} \\
\tanh(x) &= \frac{\frac{\sigma(2x)-1+\sigma(2x)}{\sigma(2x)}}{\frac{\sigma(2x)+1-\sigma(2x)}{\sigma(2x)}} \\
\therefore \tanh(x) &= 2\sigma(2x)-1
\end{aligned} \tag{14}$$

Thus, it can be shown that tanh is a scaled and shifted version of the sigmoid.

2 Implement a Fully Connected Network

2.1 Network Initialization

2.1.1

Firstly, in the back-propagation algorithm, when we multiply the zero weights by delta, there will be no change and our training will have no effect on the weights. Secondly, if all the neurons are initiated with the same weight, they will follow the same gradient and could end up learning the same function.

2.1.2

Code implemented.

2.1.3

Random initialization increase the entropy of the system and can increase the chances of finding the local minima faster. Furthermore, if all the neurons are initiated with the same weight, they will follow the same gradient and could end up learning the same function.

If we do not scale the weights properly, it could lead to vanishing or exploding gradient problems over multiple layers and this could lead to slow training due to gradient descent stuck on flat or saturated regions of the activation function curves.

2.2 Forward Propagation

Code implemented.

2.3 Backwards Propagation

Code implemented.

2.4 Training Loop

Code implemented.

2.5 Numerical Gradient Checker

Code implemented.

3 Training Models

3.1 Train a Network from Scratch

Code implemented.

3.2 Learning Rate Selection

The test and validation accuracy was found to be the best for training with **learning rate = 0.001 or 1e-3**. A lower learning rate will slow down the training or take more time to converge as the parameters and updated by a smaller magnitude and a higher learning rate might overshoot the minimum or even start to diverge from the minimum. The plots of the models trained with different learning rates along with their accuracies for 100 epochs are shown below.

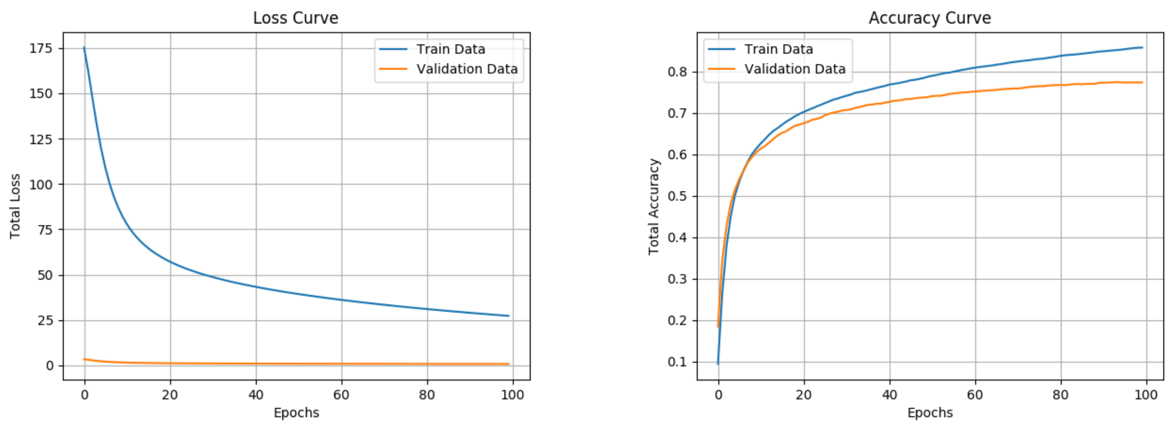

Figure 1: Loss and accuracy curves for learning rate = 1e-3

```
Train Accuracy: 0.8575
Validation Accuracy: 0.7733
Test Accuracy: 0.7794
```

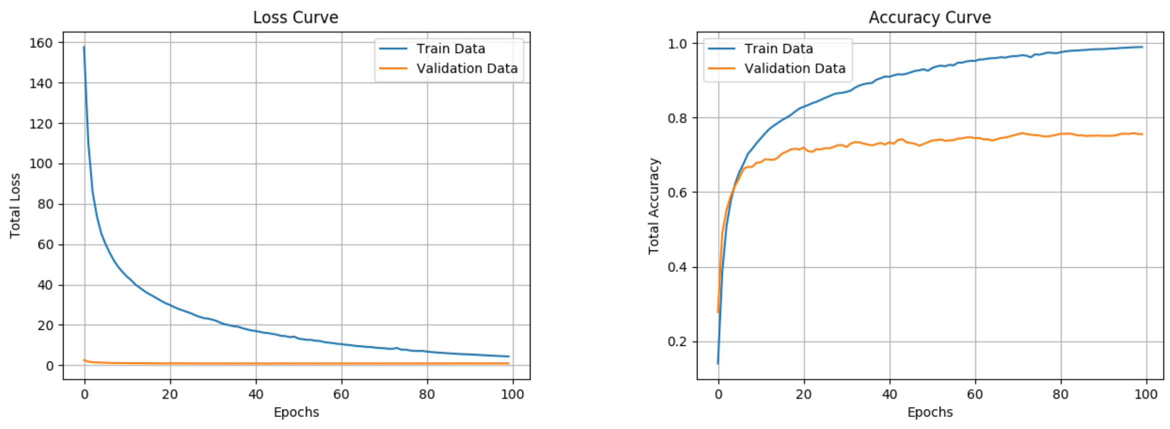

Figure 2: Loss and accuracy curves for learning rate = 1e-2

```
Train Accuracy: 0.9893
Validation Accuracy: 0.7555
Test Accuracy: 0.7644
```

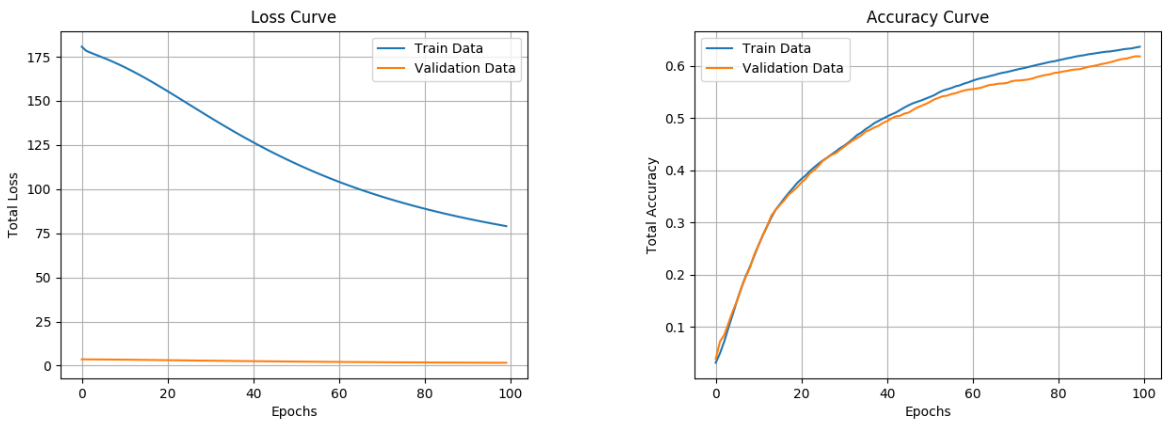

Figure 3: Loss and accuracy curves for learning rate = 1e-4

```
Train Accuracy: 0.6371
Validation Accuracy: 0.6186
Test Accuracy: 0.6183
```

3.3 Weights Visualization

Certain patterns can be observed in the weights after training. The corners and the regions near the edges seem to be low frequency or flat with less features learned and in the center region, we can see some visible strokes of the letters and numbers which look somewhat averaged over the dataset. The initial weights have no patterns in them as they are randomly initialized.

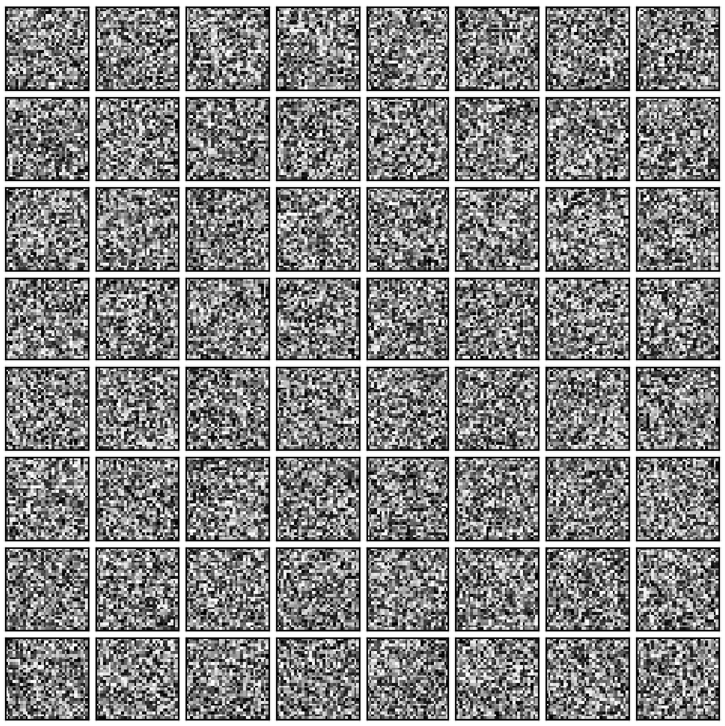

Figure 4: First layer weights immediately after initialization

Figure 5: First layer weights after training for 100 epochs with learning rate 1e-3

3.4 Confusion Matrix

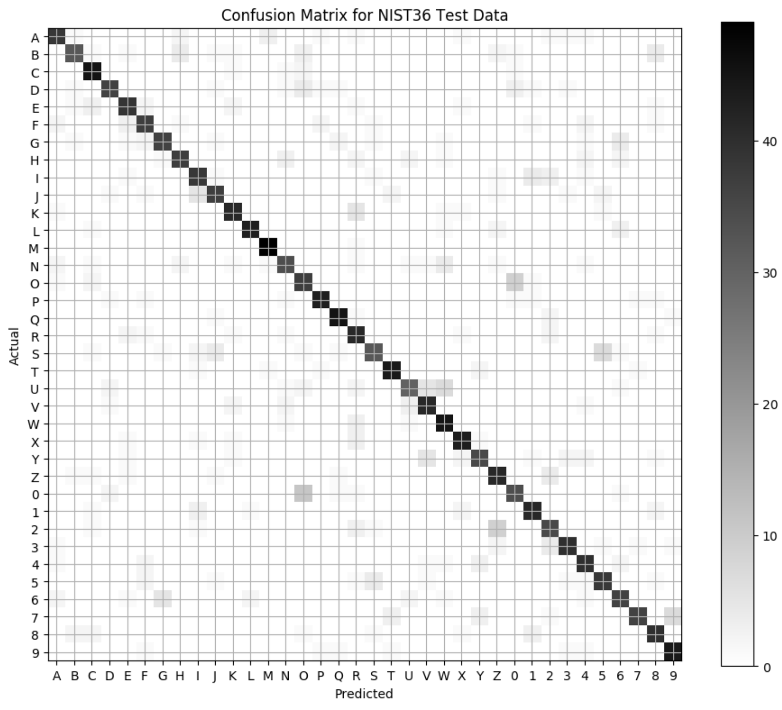

Figure 6: Confusion matrix on the NIST36 test dataset after training 100 epochs with learning rate 1e-3. The top few pairs that are commonly confused are letter ’O’ and number ’0’, ’2’ and ’Z’, ’5’ and ’S’, ’6’ and ’G’, ’K’ and ’R’ which were expected.

4 Extract Text from Images

4.1 Assumptions

1. Firstly, we assume that the characters extracted are very similar in structure to characters from the NIST36 dataset that we trained. This includes the stroke width, overall size and trivially, the style of writing. Another assumption is that the pixel values of the extracted characters are similar to the ones in the dataset, however the dataset does not have perfect binary black-and-white images upon inspection.
2. The second assumption would be that, the text extraction or the cropped image is perfect without any noises or parts of neighbouring characters. This is true in most cases, except a few extractions.

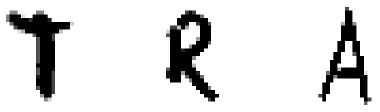

Figure 7: Example images where character detection might fail

In the first two images above, the characters look very ambiguous and in the first and last image, the stroke widths are slightly different than in the dataset and these images might fail to be detected correctly.

4.2 Text Extraction

Code implemented.

4.3 Text Extraction Results

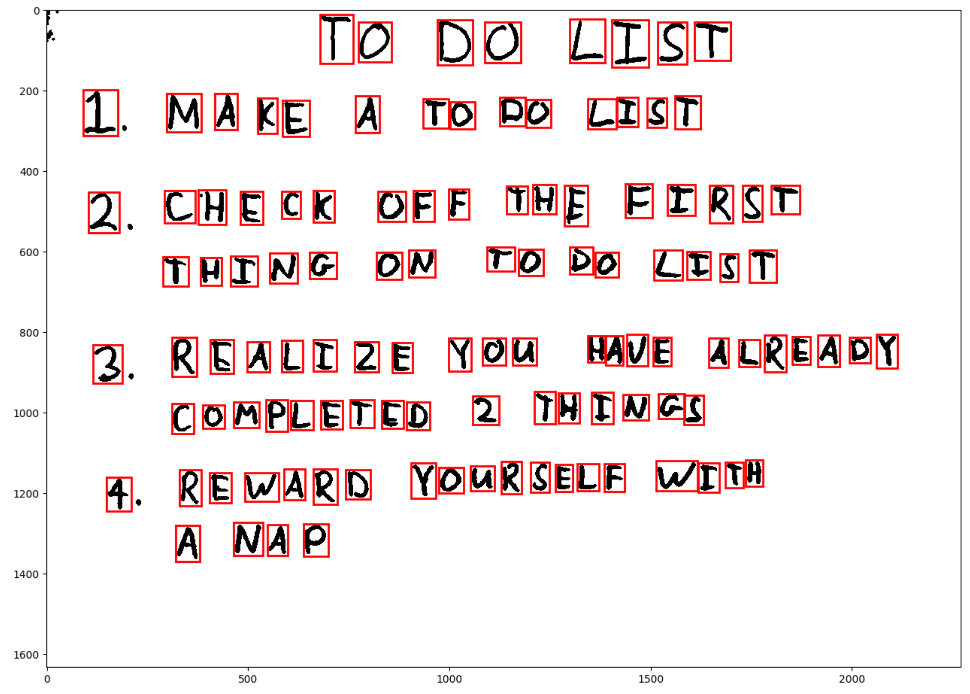

Figure 8: Text extraction results on 01_list.jpg

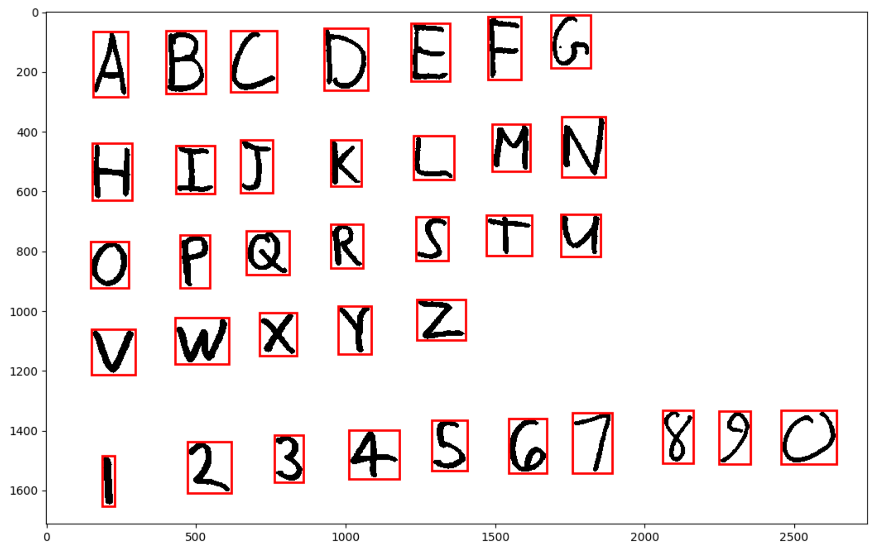
Figure 9: Text extraction results on 02_letters.jpg

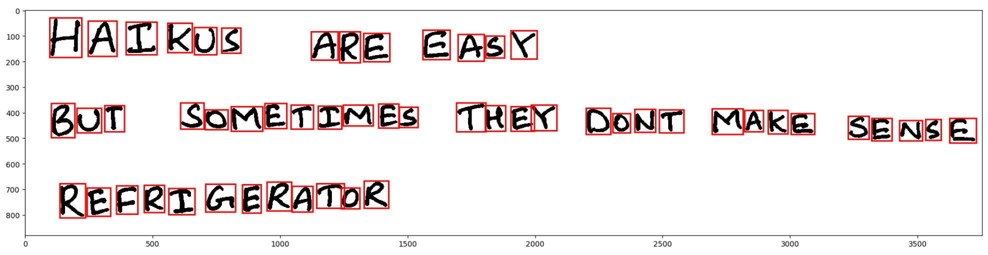
Figure 10: Text extraction results on 03_haiku.jpg

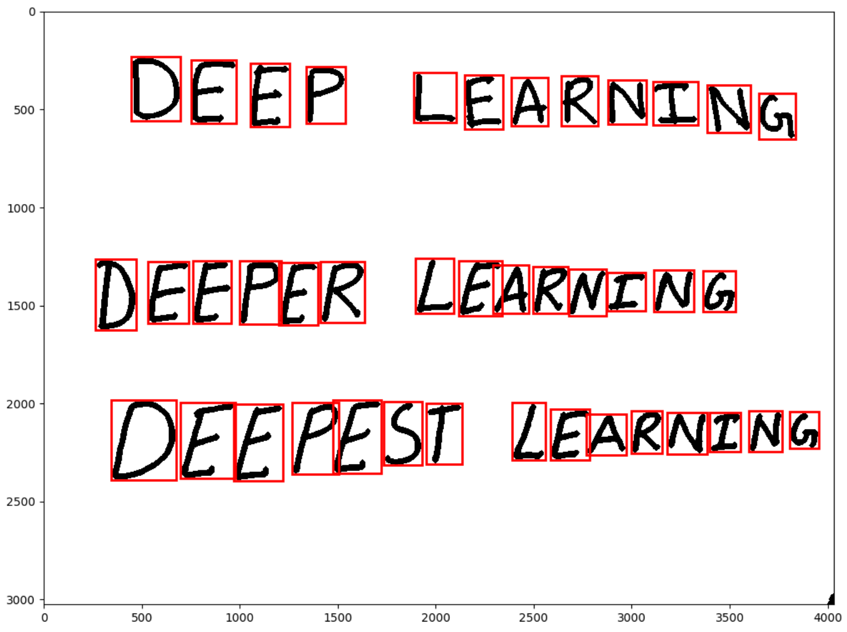
Figure 11: Text extraction results on 04_deep.jpg

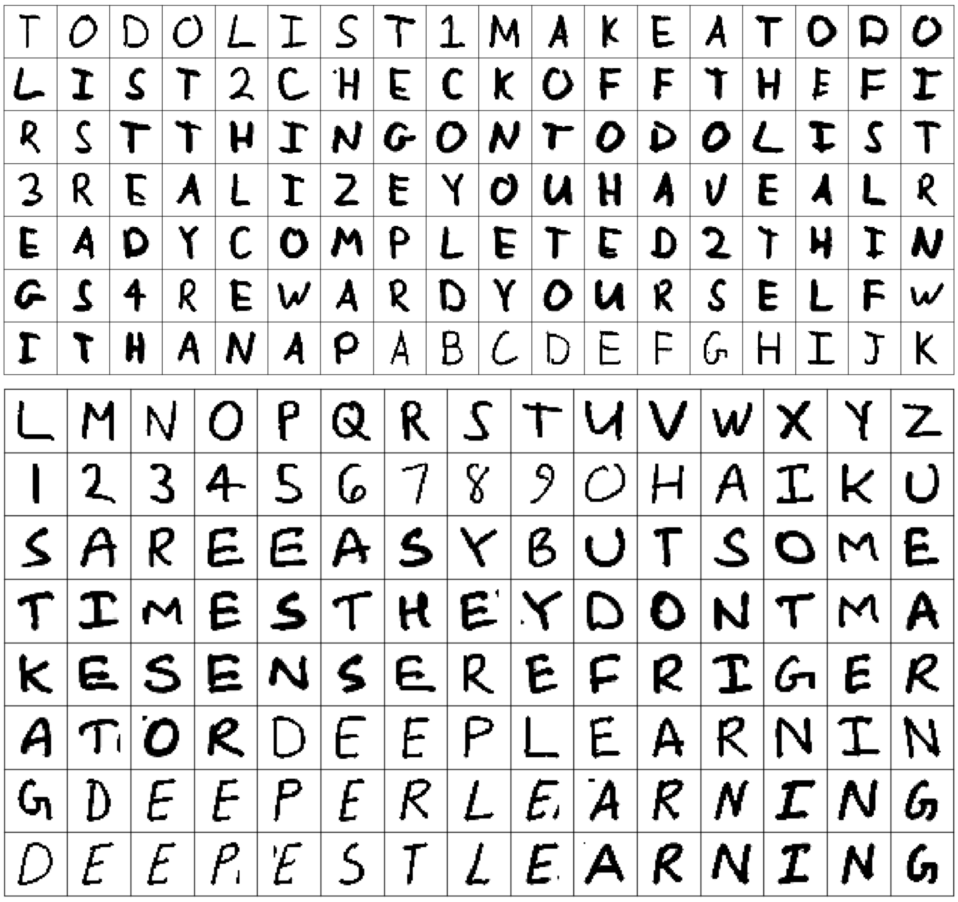
Figure 12: Text extraction results after cropping and padding

4.4 Text Detection Results

```
TO DO LIST
I MAKE A TO 20 LIST
2 CHREK OFF THE FIRST
THING ON TO 00 LI5T
3 RFALIZE YOU HAVE ALREADT
COMPLETED 2 THINGS
4 REWARD XOURSELF WITH
A NAP

Accuracy: 85.22%
```

Figure 13: Text detection results and accuracy for 01_list.jpg

```
2BCDEFG
HIJKLMN
OPQR5TW
VWXYZ
1Z3MS67890

Accuracy: 77.78%
```

Figure 14: Text detection results and accuracy for 02_letters.jpg

```
HAIKUS ARE EASY
BUT SQMETIMES TREX DONT MAKE SEMQE
REFRIGERATOR

Accuracy: 88.89%
```

Figure 15: Text detection results and accuracy for 03_haiku.jpg

```
DEEP LEARMING
DEEPER LEARKIM6
DEEREST LEARNIN6

Accuracy: 85.37%
```

Figure 16: Text detection results and accuracy for 04_deep.jpg

5 Image Compression with Autoencoders

5.1 Building the Autoencoder

Code implemented.

5.2 Training the Autoencoder

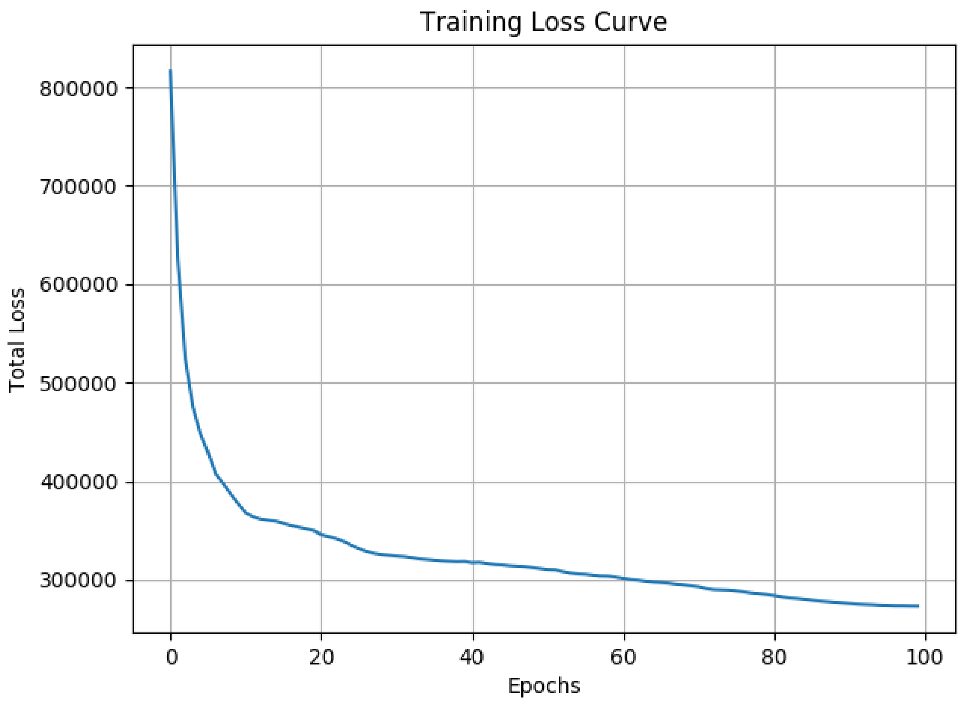

Figure 17: Training loss curve for the Autoencoder

It can be seen that the training loss decreases as the training progresses with a initial steep drop and a very relaxed or flat drop in loss towards the end.

5.3 Evaluating the Autoencoder

5.3.1

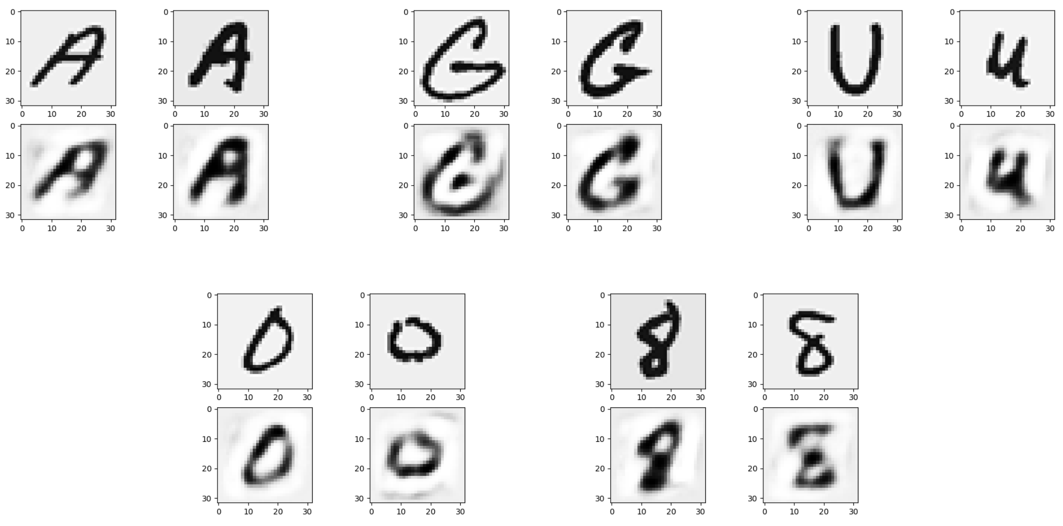

Figure 18: Reconstruction results of the Autoencoder for five classes from the NIST36 test dataset

The reconstruction results look blurry and the loss in high frequency components and features is clearly visible. This is because we are reducing the dimensionality at the bottleneck of the Autoencoder.

5.3.2

The average Peak Signal-Noise Ratio (PSNR) across all the validation images was **15.9145**.

6 Comparing against PCA

6.1 Projection Matrix

The size of the projection matrix is **32** $\times$ **1024** and its rank is **32**. The reconstruction on test images is shown in *Figure 19*.

6.2 PCA Reconstruction Results

The reconstruction results look blurry and the loss in high frequency components and features is clearly visible. This is because we are reducing the dimensionality and taking only 32 principal components. Compared to Autoencoder, the text reconstruction is more blurry and the background reconstruction is not as good as Autoencoder.

6.3 Evaluating PCA

The average Peak Signal-Noise Ratio (PSNR) across all the validation images was **16.2839**. The PCA PSNR was slightly better than the Autoencoder's PSNR. However, on visualization of the reconstructed images, Autoencoder seems to perform better. This is also true because the Autoencoder has non-linearities introduced by the ReLU activation function, which helps to learn the classification function

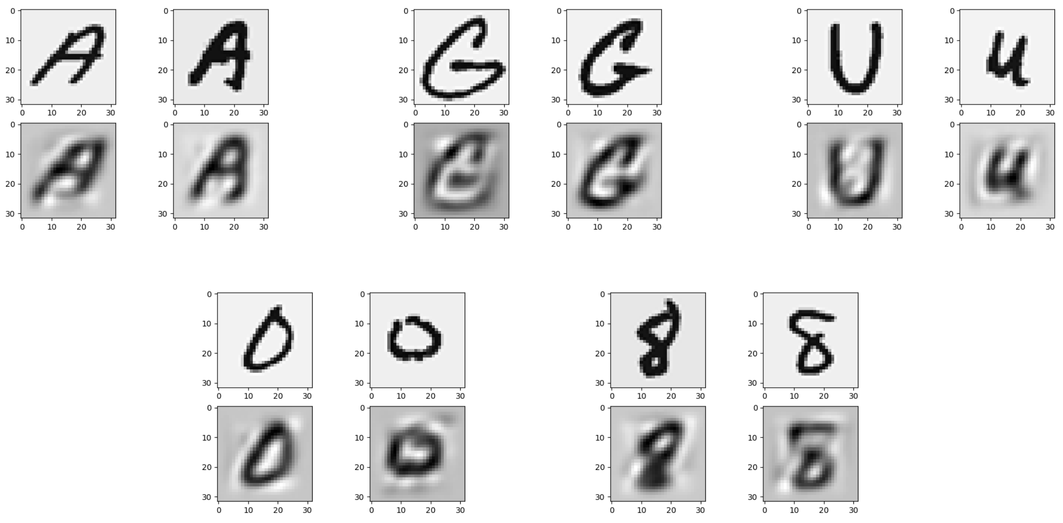
Figure 19: Reconstruction results of PCA for the same five classes from the NIST36 test dataset

better, compared to dealing with just linear operations using SVD in PCA. Furthermore, the number of learned parameters in Autoencoder is almost double that of PCA.

But as per the PSNR, PCA is better and the reason for this could be that in Autoencoder, the high frequency component reconstruction is more visible compared to PCA, although not prefect. These imperfect higher frequency components may contribute to more noise and thus the PSNR of PCA could be better than the PSNR of Autoencoder.

6.4 Number of Learned Parameters

The number of learned parameters for Autoencoder including the weights and biases is **68,704**.
The number of learned parameters for PCA is **32,768**.
The reasons for better performance are already discussed above in *6.3*.

7 PyTorch

7.1 Train a Neural Network in PyTorch

7.1.1 Fully Connected Network with NIST36 Dataset

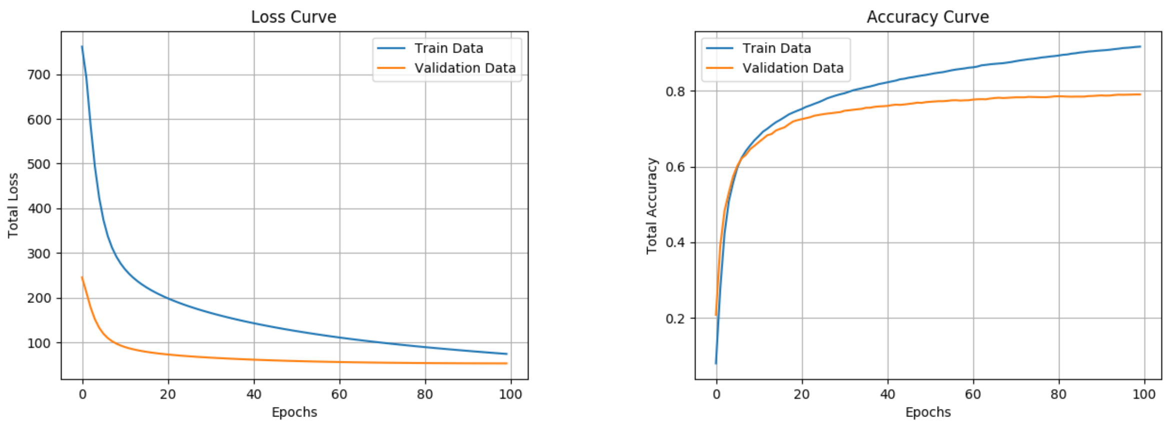

Figure 20: Loss and accuracy curves for Fully Connected Network with NIST36 Dataset

7.1.2 Convolution Neural Network (LeNet-5) with MNIST Dataset

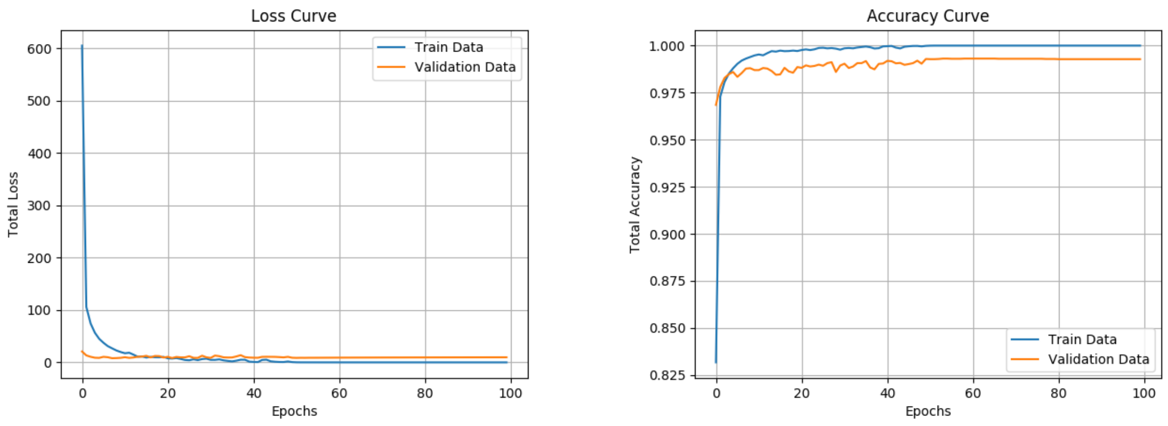

Figure 21: Loss and accuracy curves for Convolution Neural Network (LeNet-5) with MNIST Dataset

7.1.3 Convolution Neural Network (LeNet-5) with NIST36 Dataset

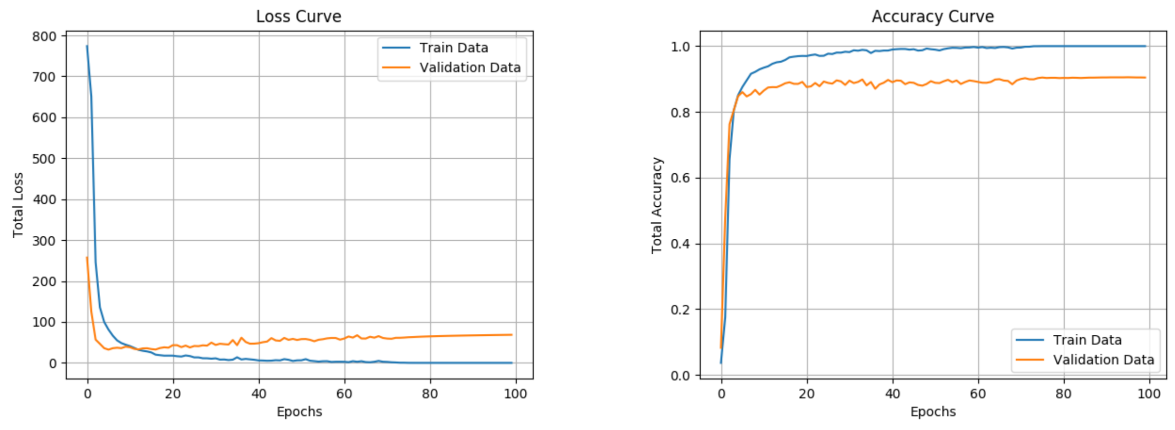

Figure 22: Loss and accuracy curves for Convolution Neural Network (LeNet-5) with NIST36 Dataset

7.1.4 Convolution Neural Network (LeNet-5) with EMNIST Dataset

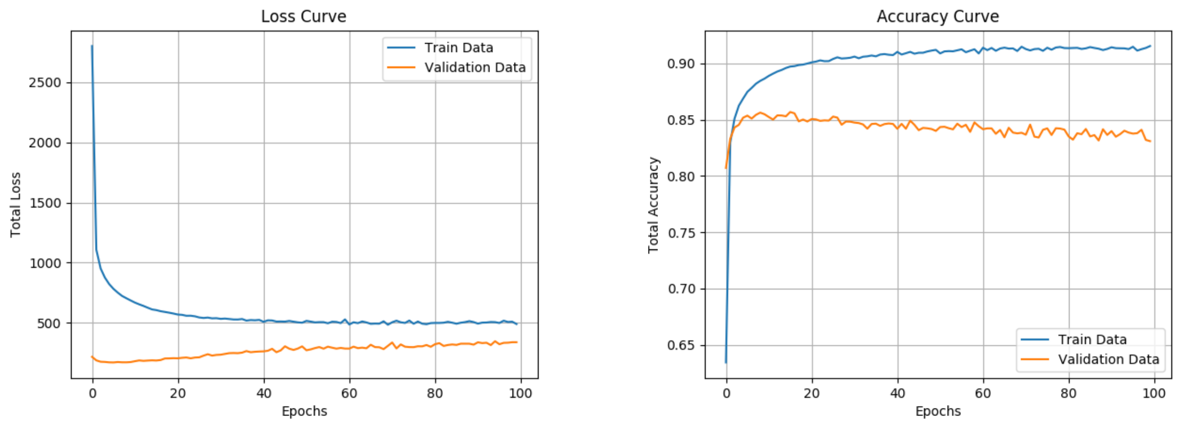

Figure 23: Loss and accuracy curves for Convolution Neural Network (LeNet-5) with EMNIST Dataset

```
JD DO LI5T
I MARE A TO DO LIST
2 CHECK OFF THE FIfST
THINQ ON TO DO LZST
3 REAEE2E YOU HAVE AfREADY
COMPLErED 2 THINGS
4 REWARD YOURSELF WITH
R NAP

Actual Accuracy: 87.83%
Absolute Accuracy: 87.83%
```

Figure 24: Text detection results and accuracy with LeNet-5 for 01_list.jpg

Two accuracies are reported here. The actual accuracy is case sensitive and the absolute accuracy is reported after converting all labels and predictions to upper-case.

```
ABCDEfG
HIJKLMN
OPQRStY
VWXYZ
12345G78gD

Actual Accuracy: 83.33%
Absolute Accuracy: 88.89%
```

Figure 25: Text detection results and accuracy with LeNet-5 for 02_letters.jpg

```
HAIKUS ARE EASY
BUT SOMETIMES THEY DONt MAKE SEnGE
REfRI6ERAtOR

Actual Accuracy: 88.89%
Absolute Accuracy: 96.30%
```

Figure 26: Text detection results and accuracy with LeNet-5 for 03_haiku.jpg

```
DEEP LEARNING
dEEPER LEARNIN6
DEERE5J LEARNZNG

Actual Accuracy: 85.37%
Absolute Accuracy: 87.80%
```

Figure 27: Text detection results and accuracy with LeNet-5 for 04_deep.jpg

7.2 Fine Tuning

7.2.1 Fine Tuning Flowers-17 Dataset on SqueezeNet 1.1 vs Training LeNet-5 from Scratch

The results of fine-tuning SqueezeNet 1.1 classifier (validation accuracy of 0.86) was much better than training LeNet-5 (validation accuracy of 0.49) from scratch. This was expected because SqueezeNet has a much deeper architecture and learns more features. Furthermore, SqueezeNet was trained on a much larger dataset, ImageNet compared to Oxford Flowers-17.

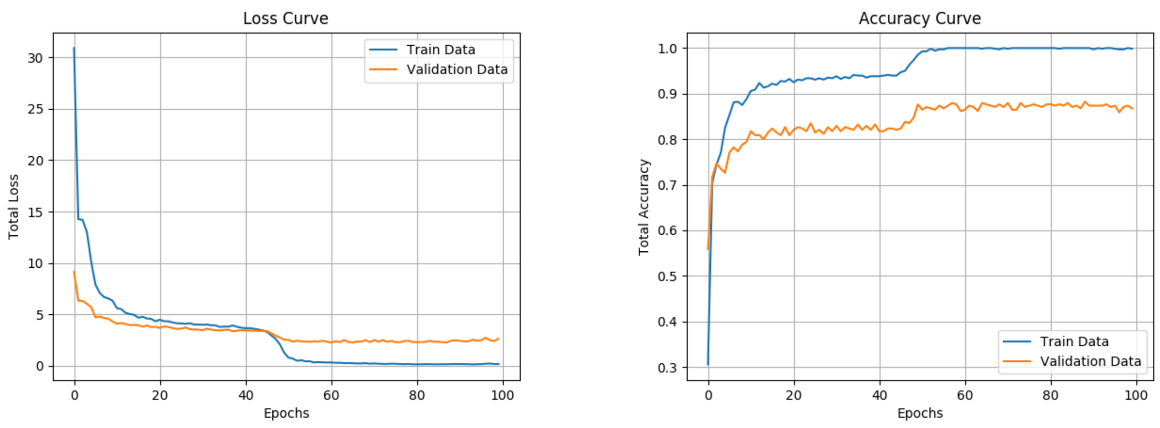

Figure 28: Loss and accuracy curves for fine-tuning SqueezeNet 1.1 with Flowers-17 Dataset

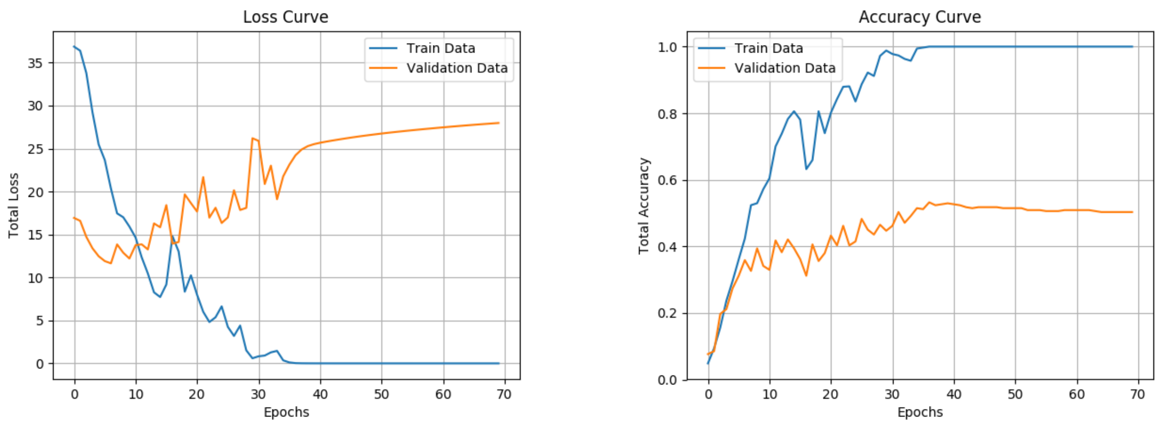

Figure 29: Loss and accuracy curves for training LeNet-5 with Flowers-17 Dataset from scratch