Computer Vision - 16720A

[redacted]
Carnegie Mellon University
[redacted]

1. Theory

1.1. Theory

Prove that softmax is invariant to translation

$$softmax(x) = softmax(x + c) \tag{1}$$

Here,

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2}$$

and

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

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

$$softmax(x_i + c) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3}$$

Comparing equations (2) and (3)

$$softmax(x) = softmax(x + c) \tag{4}$$

If c = 0, Softmax is prone to both overflow - (very large numbers approximated to infinity) and underflow - (very small numbers approximated to 0). If c = max(x), the weight vector is entirely non-positive. This eliminates overflow. The problem of vanishing gradient is also negated since at least one vector element will have a value of zero. The range of output values would now be between (0, 1].

1.2. Theory

- What are the properties of softmax(x), namely what is the range of each element? What is the sum over all elements?
  The properties of softmax are all output values in the range (0, 1] and 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 over j different possible output classes** ".

- Can you see the role of each step in the multi-step process now? Explain them.
  The first step applies the standard exponential function to each element of vector X. The second step is to compute the sum of these exponentials. The third step involves normalizing each individual exponential by dividing by the sum of all these exponentials so that the sum of the output is 1. It maps it to a probability distribution over j different output classes.

1.3. Theory

Show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Figure 1: neural network with 2 hidden layers

For the sake of convenience, let us assume a neural network with 2 hidden layers. Let y be the output of the network and x be the input of the network. w stands for the weights and b stands for the bias. The suffixes of b and w stand for the layer number. The output is

$$y = h_2 w_3 + b_3$$

Substituting weights of individual layers

$$y = (h_1 w_2 + b_2) w_3 + b_3$$

$$y = h_1 w_2 w_3 + b_2 w_3 + b_3$$

$$y = (x w_1 + b_1) w_2 w_3 + b_2 w_3 + b_3$$

$$y = x w_1 w_2 w_3 + b_1 w_2 w_3 + b_2 w_3 + b_3$$

Let $W = w_1 w_2 w_3$ and $b = b_1 w_2 w_3 + b_2 w_3 + b_3$

$$y = xW + b$$

The combination of several individual weights can be written as single weight constant. Same goes for the bias term b. This also stands if we include any linear activation function. The multi-layer neural network can be thus reduced to a single case of linear regression since the output y is in the form of a straight line with slope W and intercept b.

1.4. Theory

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

Thus, the gradient of the sigmoid function is a function of $\sigma(x)$

1.5. Theory

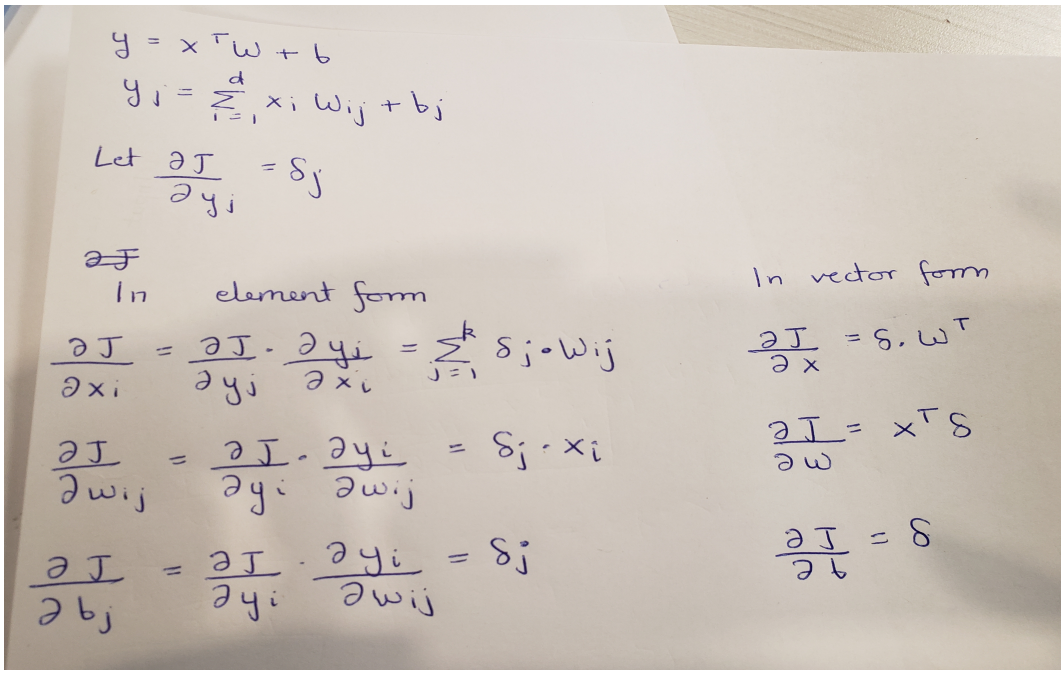


Figure 2: The derivatives of gradient

1.6. Theory

1. During backpropagation of neural networks, gradients are computed. By chain rule, the derivatives of each layer are multiplied down the network to compute derivatives of initial layers.
   As shown in the figure, the value of the derivative of sigmoid tends to 0. For a shallow network with few layers, this won't have a significant effect. But for deeper networks, several small derivative values are multiplied together and the net result tends to 0. This is the vanishing gradient problem where the gradient decreases exponentially as one moves down to the initial layers.

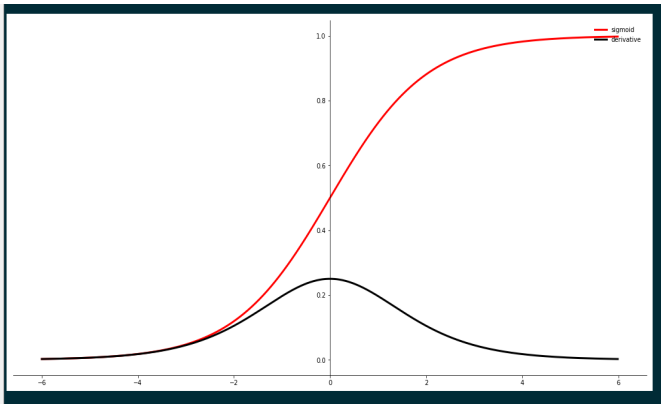


Figure 3: The sigmoid function and its derivative

2. The output range of $tanh(x)$ is [-1,1] and that of sigmoid is [0,1]. There are two reasons for preferring tanh over sigmoid.

   Since the data is centred around 0 after having been normalized, tanh will have a much stronger gradient value than sigmoid given the afore-mentioned range values.
   Sigmoid is asymmetric in nature meaning it introduces a bias in the gradients. Since tanh is symmetric, it avoids bias and the model tends to converge faster and the trained model may have better performance.

3. As shown in the figure, the derivative of tan h is greater than that of sigmoid centred around 0. tanh has a much stronger gradient value and hence the problem of vanishing gradient disappears.

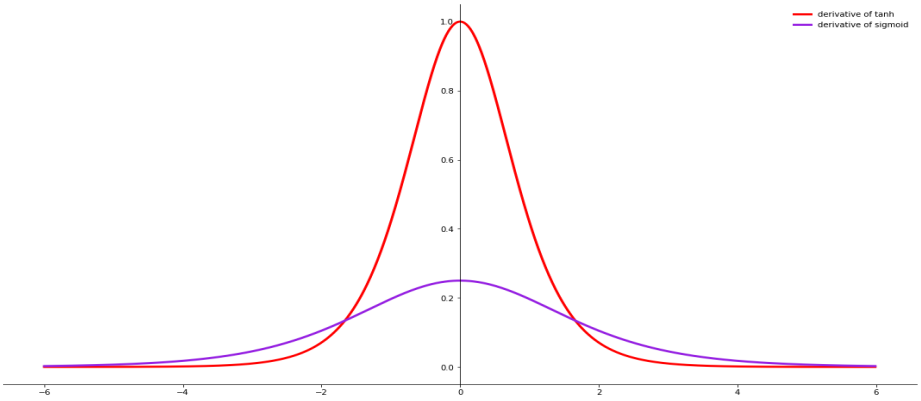


Figure 4: The derivatives of activation functions

4.

$$tanh(x) = 2\sigma(2x) - 1 \qquad where \qquad tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad and \qquad \sigma(x) = \frac{1}{1 + e^- x}$$

2. Implement a Fully Connected Network

2.1. Network Initialization

2.1.1 Theory

Neural networks tend to get stuck at local minima. So to avoid this, it is better not to initialize all the weights as 0.
If all weights are set to 0 during forward propagation, you would get the same value of output despite what the input signal is.
Also during backpropagation, whilst computing the derivative for the gradient, all the gradients will be the same. The parameters won't get updated. This will defeat the purpose of training

2.1.2 Code

2.1.3 Theory

Random initialization of weights leads to faster training and convergence and better loss. It also removes the symmetry and bias from the model. It meets the requirement of stochastic gradient descent and helps attain an optimal set of weights without letting the model get stuck at a local minima.
We scale the initialization depending on layer size so that the learning does not saturate and we avoid the vanishing gradient as well as the exploding gradient problem.

3. Training models

3.1.

3.1.1 Code

Learning rate is 0.01. Training accuracy is 98% over 50 iterations. Validation accuracy is 73.44%.

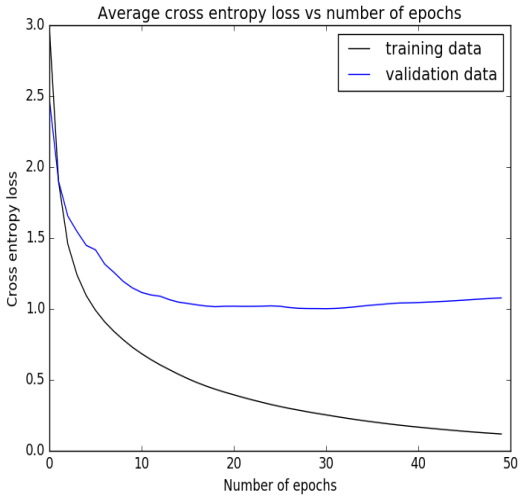


Figure 5: Average cross entropy loss vs number of epochs - learning rate 0.01

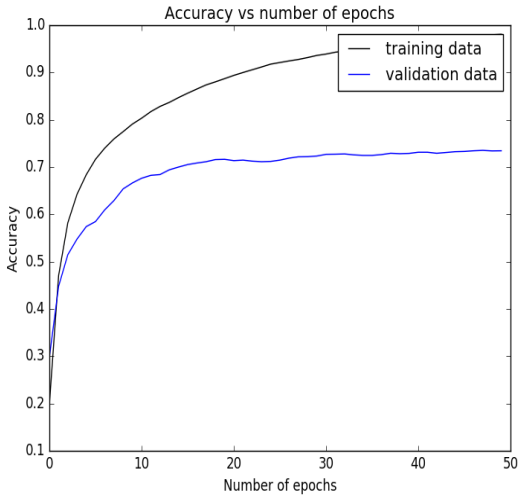


Figure 6: Accuracy vs number of epochs - learning rate 0.01

Figure 7: Learning Rate 0.01

3.1.2 Writeup

When learning rate is 0.1, training accuracy is 3%. Validation accuracy is 2.77%. When learning rate is 0.001, training accuracy is 82%. Validation accuracy is 71.88%. Learning rate determines how fast weights update themselves. During backpropagation, an estimation is made of the amount of error for which the weights are responsible. The weights are scaled using the learning rate during updation. When the learning rate is too large (0.1 in this case), performance of the model oscillates greatly over the iterations as demonstrated in the figure. This is caused due to diverging weights since oftentimes the derivative misses the zero point value. Thus a very high learning rate results in faster training at the risk of arriving at a sub-optimal set of weights. When the learning rate is too small (0.001 in this case), an optimum set of weights is reached but training takes a significantly longer time.

Test accuracy with optimum learning rate of 0.01 over 50 epochs is 74.38 percent

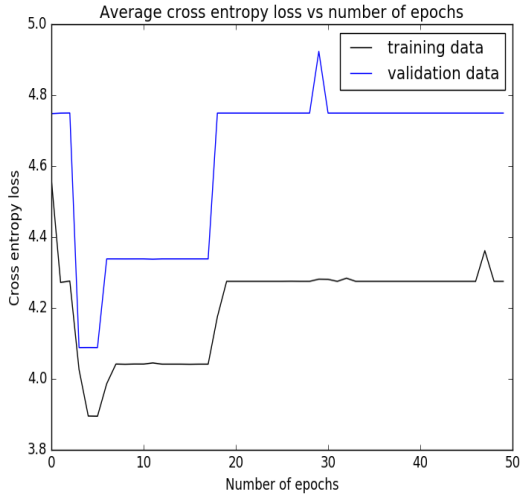


Figure 8: Average cross entropy loss vs number of epochs

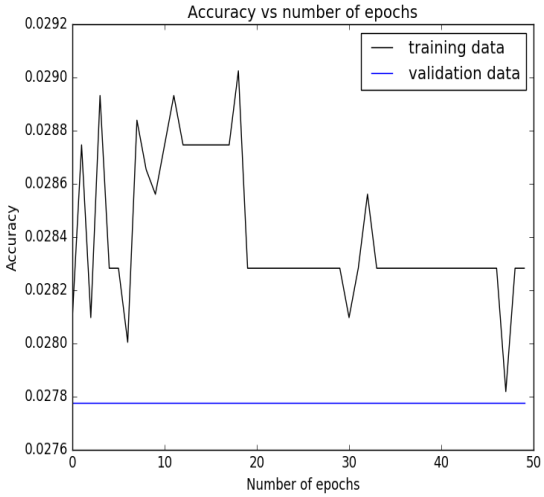


Figure 9: Accuracy vs number of epochs

Figure 10: Learning Rate 0.1

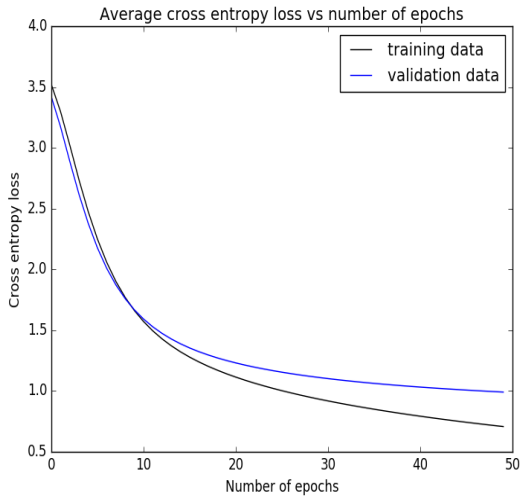


Figure 11: Average cross entropy loss vs number of epochs

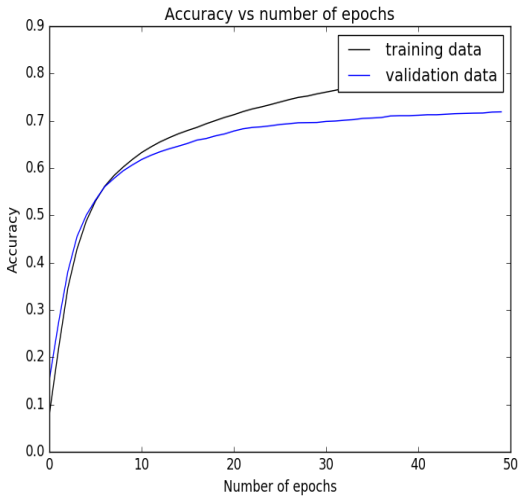


Figure 12: Accuracy vs number of epochs

Figure 13: Learning Rate 0.001

3.1.3 Writeup

In the figure visualizing the initial weights, no patterns or clusters are detected because they are initialized at random. However after learning, the weights show definitive patterns that correspond to edge detection and feature detection.

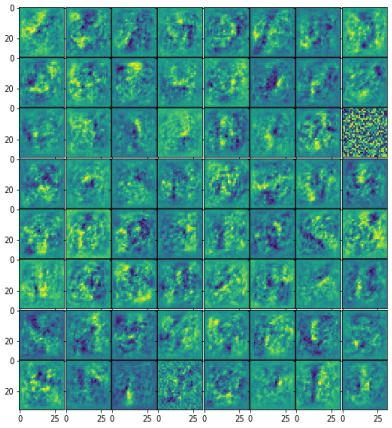


Figure 14: Weights after learning

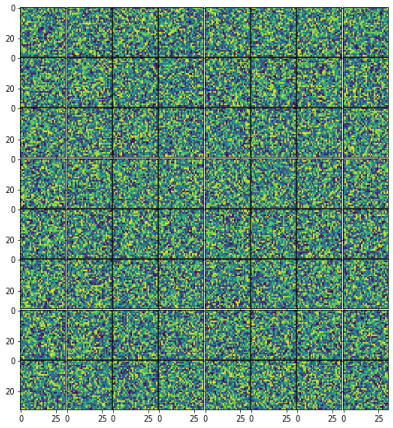


Figure 15: Initial weights

3.1.4 Writeup

The classes that are most commonly confused are (0, O), (5, S), (2, Z), (F, P), (1, I) and (G, 6) and vice-versa. They show similar patterns because they have similar edges and the neural network finds it difficult to distinguish them. Also the cnn finds it hard to learn curve features. All the cllases above have curves.

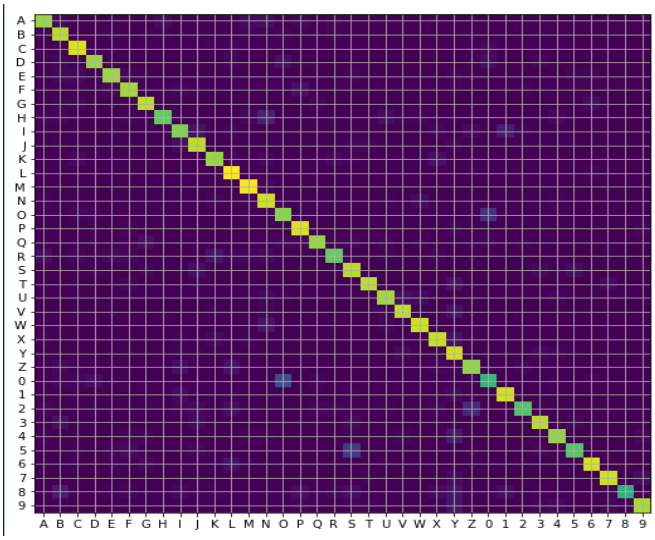


Figure 16: Confusion matrix - train data

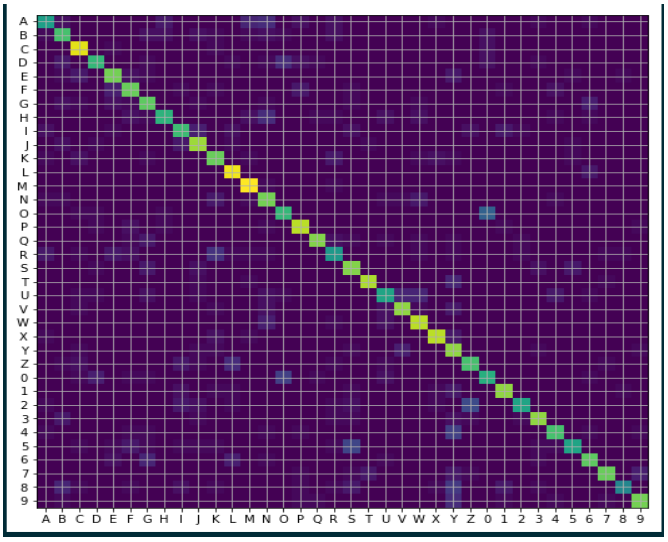


Figure 17: Confusion matrix - test data

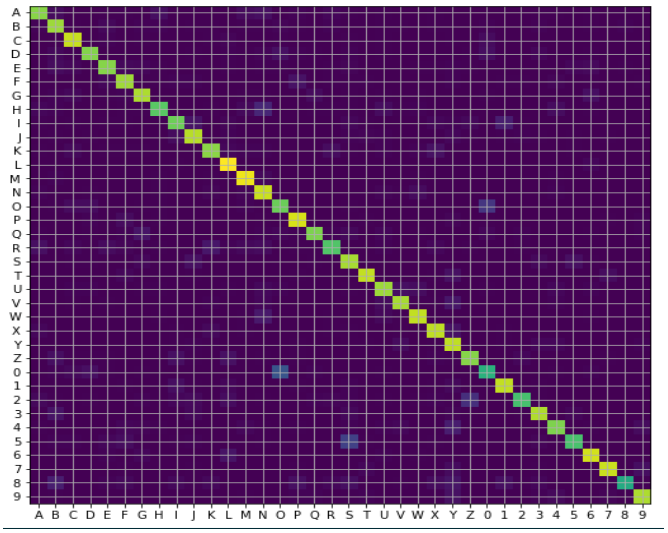


Figure 18: Confusion matrix - validation data

3.1.5 Writeup

Code Learning rate is 0.01. Training accuracy is 90% over 50 iterations. Validation accuracy is 75.4%.

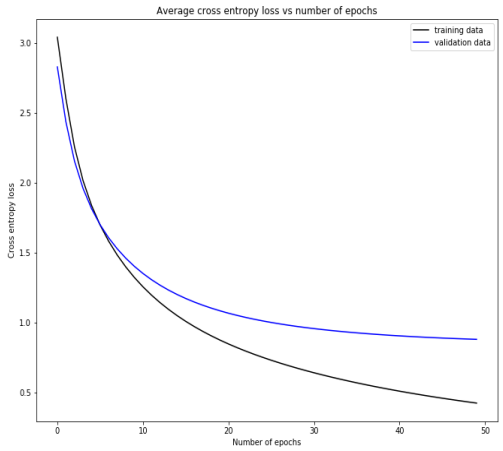


Figure 19: Average cross entropy loss vs number of epochs - learning rate 0.01

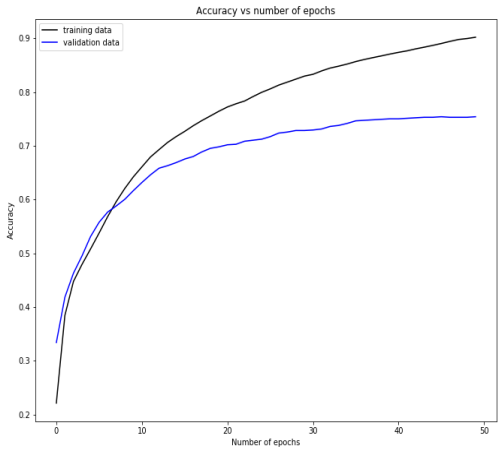


Figure 20: Accuracy vs number of epochs - learning rate 0.01

Figure 21: Learning Rate 0.01

Writeup When learning rate is 0.1, max training accuracy is 6%. Maximum validation accuracy is 8.8%. When learning rate is 0.001, training accuracy is 82%. Validation accuracy is 71.88%. Learning rate determines how fast weights update themselves. During backpropagation, an estimation is made of the amount of error for which the weights are responsible. The weights are scaled using the learning rate during updation. When the learning rate is too large (0.1 in this case), performance of the model oscillates greatly over the iterations as demonstrated in the figure. This is caused due to diverging weights since oftentimes the derivative misses the zero point value. No features are learnt. Thus a very high learning rate results in faster training at the risk of arriving at a sub-optimal set of weights. When the learning rate is too small (0.001 in this case), an optimum set of weights is reached but training takes a significantly longer time.

Test accuracy with optimum learning rate of 0.01 over 50 epochs is 52.33 percent

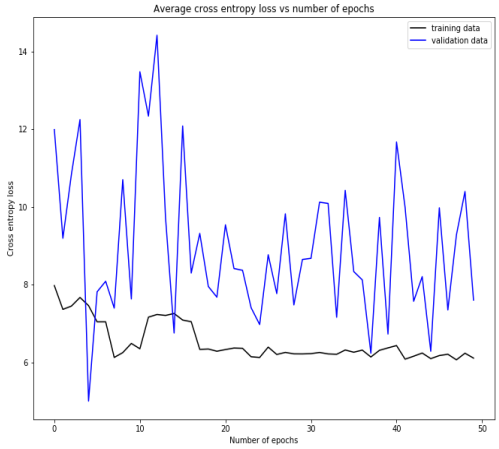


Figure 22: Average cross entropy loss vs number of epochs

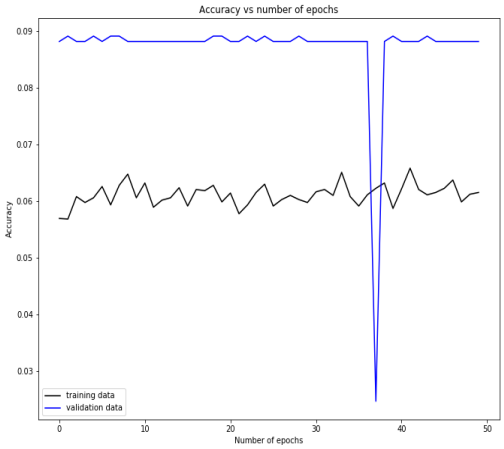


Figure 23: Accuracy vs number of epochs

Figure 24: Learning Rate 0.1

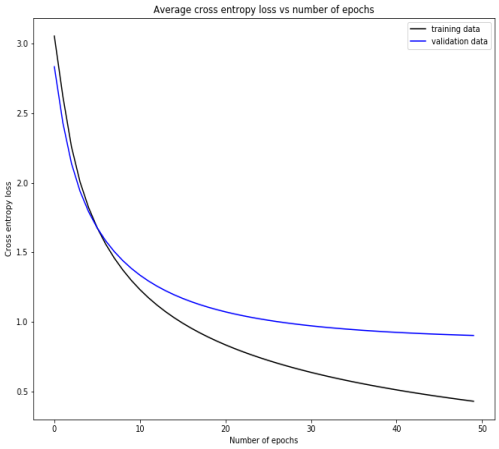


Figure 25: Average cross entropy loss vs number of epochs

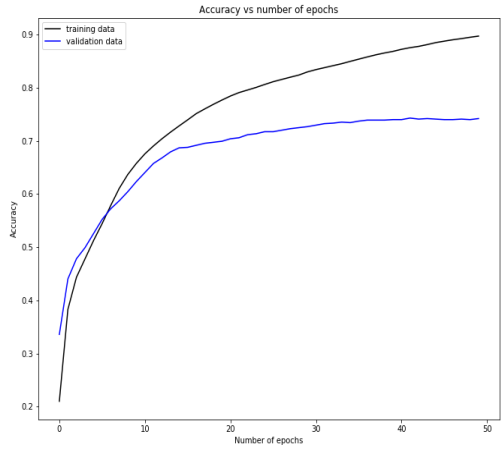


Figure 26: Accuracy vs number of epochs

Figure 27: Learning Rate 0.001

Writeup

In the figure visualizing the initial weights, no patterns or clusters are detected because they are initialized at random. However after learning, the weights show definitive patterns that correspond to edge detection and feature detection.

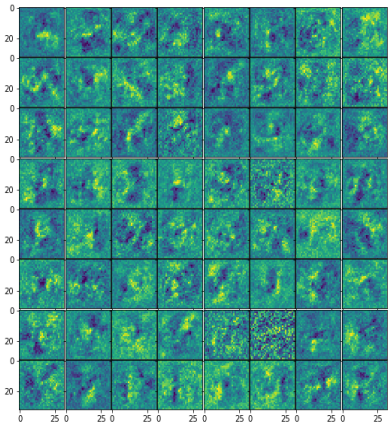


Figure 28: Weights after learning

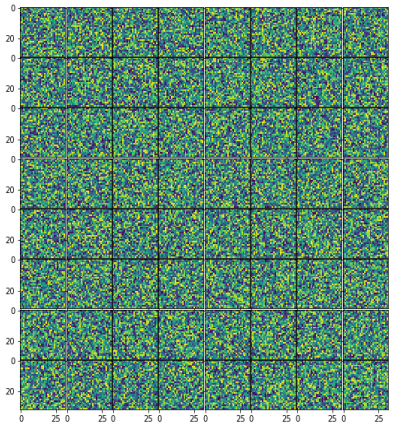


Figure 29: Initial weights

Writeup Similar results are not observed. There is almost perfect or perfect accuracy during training and validation. However test accuracy is surprisingly low as is also evidenced by the confusion matrix. This may be due to the fact that there are several classes in the test set (in this case letters or numbers) that have not been seen in the train set or validation set. As a result of these anomalies, we get poor test accuracy.

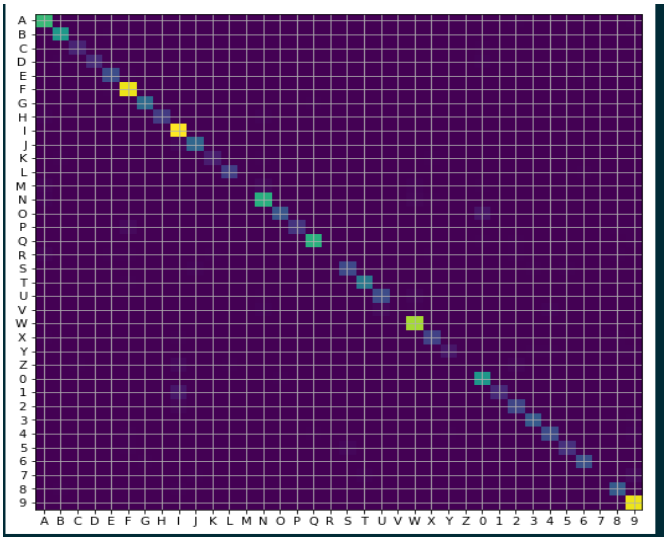


Figure 30: Confusion matrix - train data

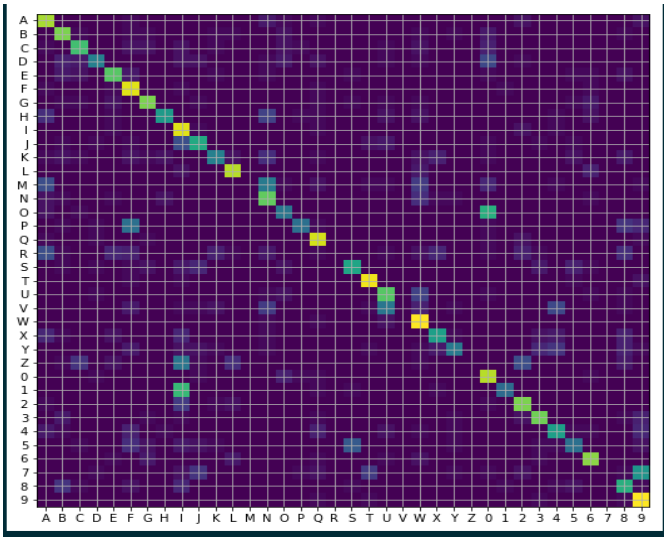


Figure 31: Confusion matrix - test data

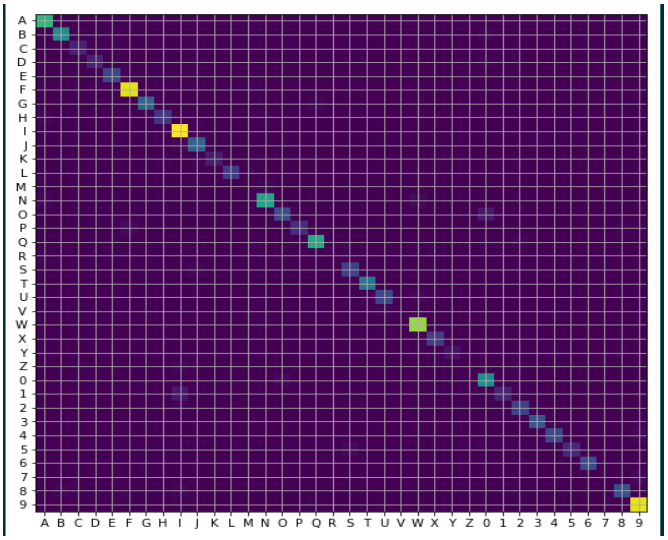


Figure 32: Confusion matrix - validation data

4. Extract Text from Images

4.1. Theory

The two big assumptions that the network makes are

1. The network assumes that consecutive characters are not connected and can be separated by bounding boxes. This is not always the case especially with regards to handwritten digits. Consecutive characters may be joined while writing

2. Another assumption is that individual characters are fully connected. There may be gaps whilst writing individual letters. Also it would fail in the case of lower case letters like i and j because the network would detect 2 bounding boxes per letter.

3. The third assumption is that all the letters are the same size because the bounding boxes are the same height.

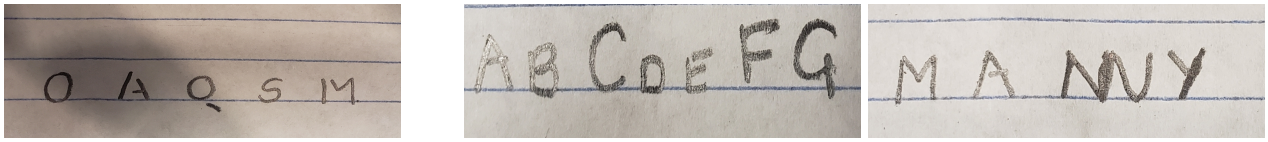
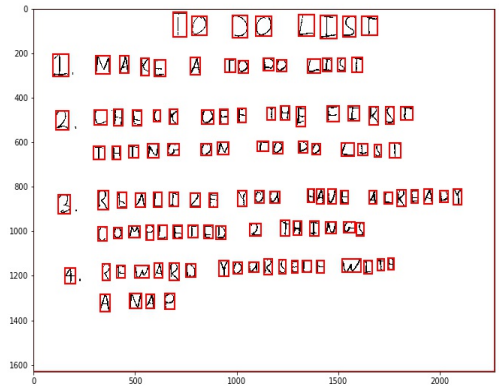


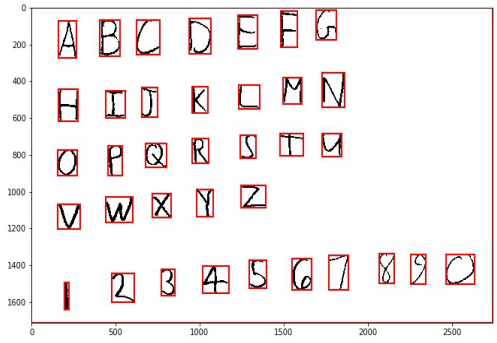
Figure 33: Cases of Failures

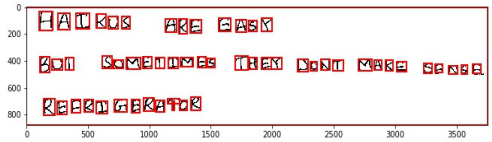
4.2. Code

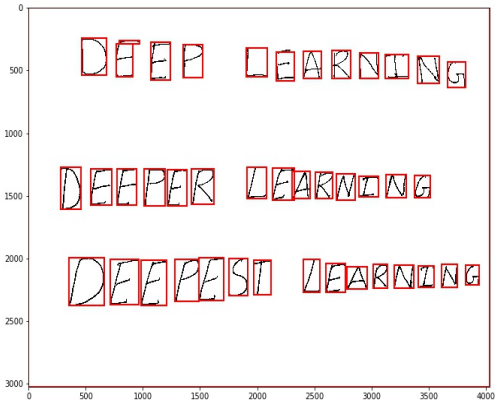
4.3. Writeup

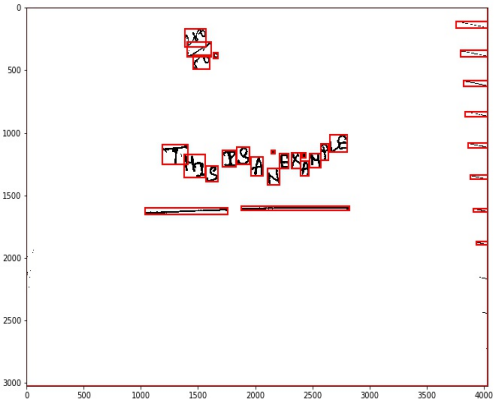
Bounding boxes are as follows

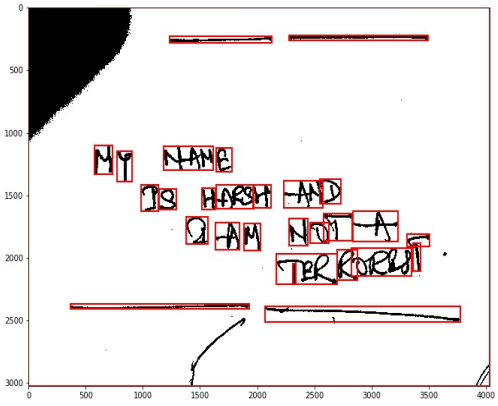












4.4. Code/Writeup

The output is as follows:
Image 1
TQ JQ LIST
I NAK6 A TQ QQ LIST
I LNICK QFF 9HE FIFST
THTNG QN TQ DQ LIST
I 8IALIIB XQJ UNAUE AL86ADT
LQBPLFILB I ININGI
9 8FWA8B 8QW8IBLH WITA
ANAP

Image 2
ABLBFFG

HIIELBN
Q8QEITW
UWX9L
IL3GSGT89I

Image 3
HAIWWQ A8G WAGT
BWT SQFBTINEG THET DOWT FAKE SGWQE
EBGBIGB8AQK

Image 4
DBFF LLAKAING
0FFTF8 L8A8NING
DEFEFSI IEA8NING

Image 5
FF
T
BTWBB
9QBOBA3
QQ9NTTAO
INBWI
FF
F1

Image 6
44
F
WQ4
0
4
TQ9Q4
QT3QQBQB
3NA4
FF4
4

5. Image Compression with Autoencoder

5.1. Building the autoencoder

5.1.1 Code

5.1.2 Code

5.2. Training the autoencoder

5.2.1 Code

The training loss decreases as the iterations progress as is usually the case in training. Initially during the first 20 epochs there is a sharp decrement in training loss, followed by a more gradual one and ultimately convergence. It may not be necessary to train the autoencoder beyond 40 epochs as overfitting might occur.

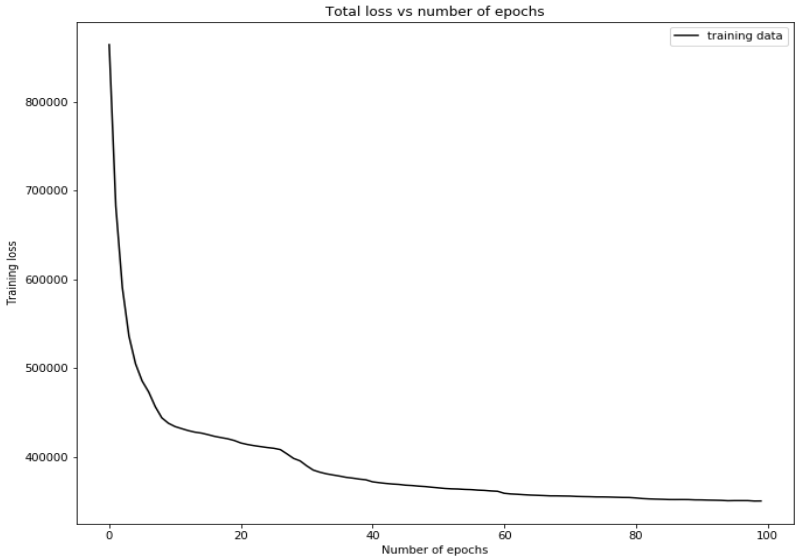


Figure 34: Total training loss vs epochs

5.3. Evaluating the autoencoder

5.3.1 Writeup/Code

The first and third columns are the original validation images. The second and fourth columns are the corresponding reconstruction images. Each row is a new class.

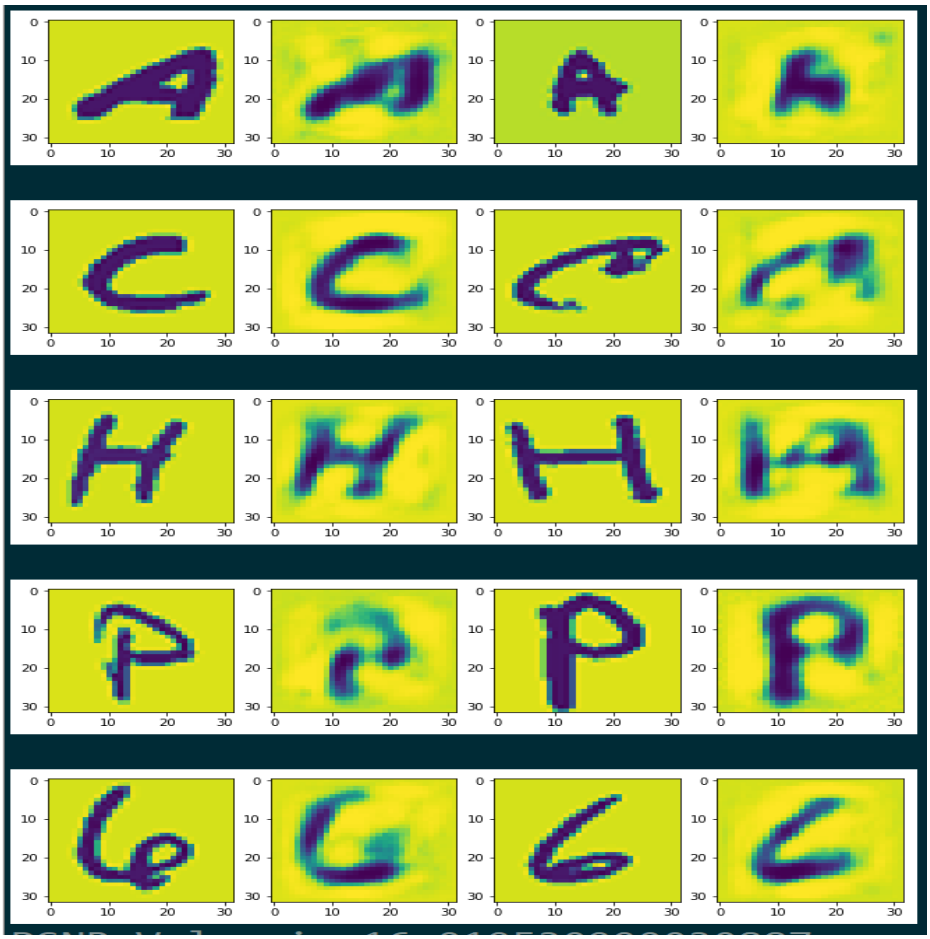


Figure 35: Autoencoder

The reconstructed images are blurred and there is some noise as compared to the original validation images. The output images have learnt the input characters and numbers perfectly and reconstructed it but there may be a chance of no intelligence being used in feature detection.

5.3.2 Writeup

The PSNR value is 16.0195.

6. Comparing against PCA

6.1. Writeup

The projection matrix has shape (32, 1024). The rank is 32.

6.2. Writeup

The first and third columns are the original test images. The second and fourth columns are the corresponding reconstruction images. Each row is a new class.

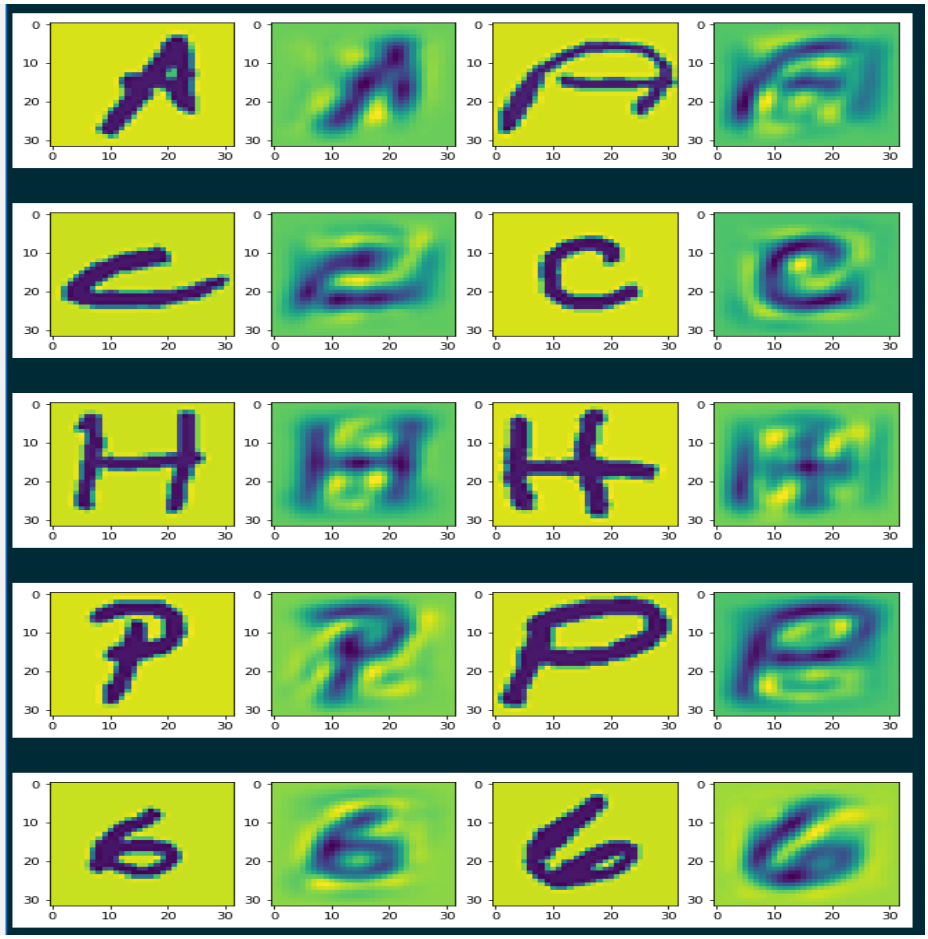


Figure 36: Principal Component Analysis

As compared to the original test images, the reconstructed ones are blurry and have a high level of noise, though the characters and numbers seem to have been mapped perfectly in the output. The auto-encoder provides better visualization outputs as compared to PCA with less blurry images with less noise. Ultimately PCA is restricted to a linear map, while auto encoders can have nonlinear en-oder/decoders. A single layer auto encoder with a linear transfer function however is nearly equivalent to PCA, where the subspace for a particular letter will be the same.

6.3. Writeup

The PSNR is 16.359. It is better than the autoencoder. This is becuase a higger PCA doesn't necessarily mean better image visualization. The output may be subject to a shift which is poorly captured by evaluation methods such as PSNR even though output images may look better. Structural similarity index may be better at evaluating which of the two is better as compared to PSNR.

6.4. Writeup

Autoencoder parameters: 1024*32 + 32*32 + 32*32 + 32*1024 + 1024 + 32 + 32+ 32 = 68704
PCA parameters: 32*1024 = 32768
An autoencoder has more parameters and can thus capture linear as well as non-linear functions and the resulting features much better than PCA which is only capable of a linear mapping. This accounts for the difference in performance and visualization

7. PyTorch

7.1. Train a neural network in PyTorch

7.1.1 Code/Writeup

Training loss and accuracy over time on NIST36 . Maximum training accuracy is 87.22%. Test accuracy is 79.55%.

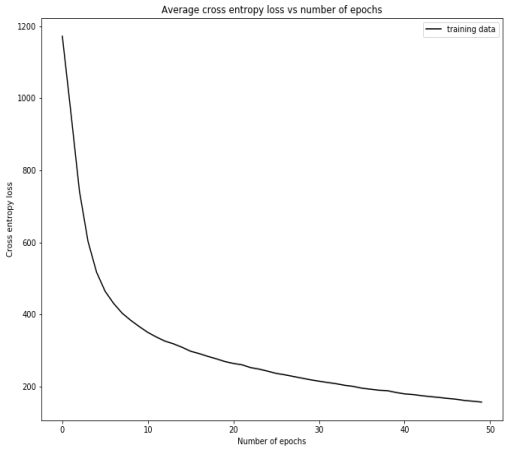


Figure 37: Training loss vs epochs

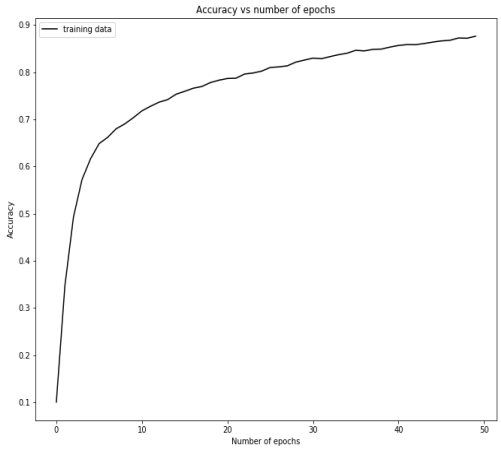


Figure 38: Accuracy vs epochs

7.1.2 Code/Writeup

Mean training loss and accuracy over time on MNIST. Maximum training accuracy is 96.02%. Test accuracy is 98.87%. Experiment conducted using GPU in google collaboratory for faster training.

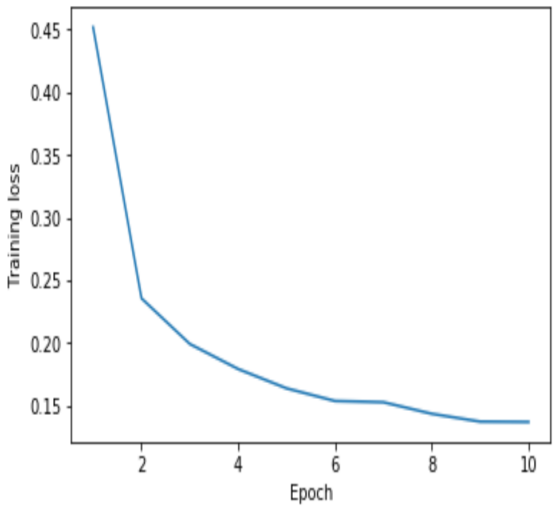


Figure 39: Mean training loss vs epochs

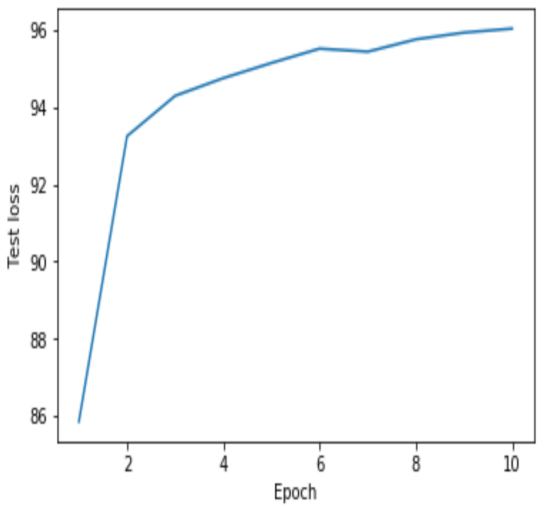


Figure 40: Accuracy vs epochs

7.1.3 Code/Writeup

Training loss and accuracy over time on NIST36. Maximum training accuracy is 94.51%. Test accuracy is 89.78%.

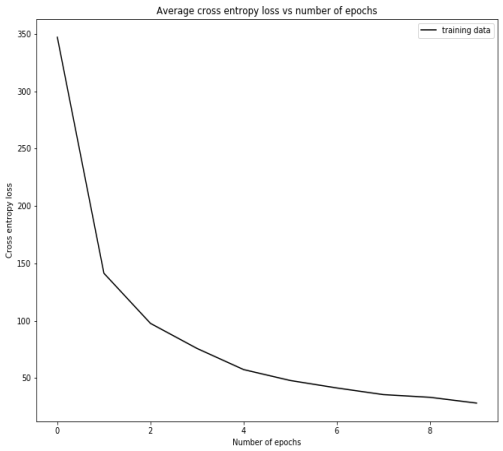


Figure 41: Training loss vs epochs

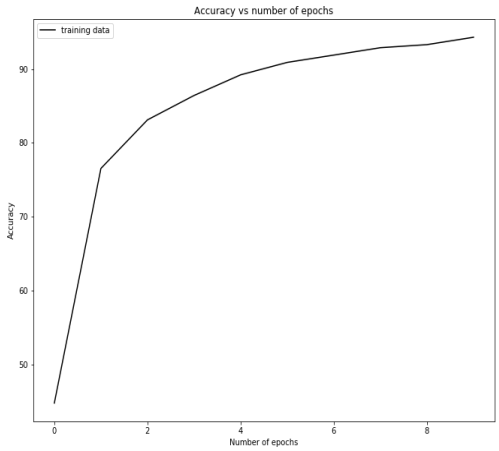


Figure 42: Accuracy vs epochs

7.1.4 Code/Writeup

Training loss and accuracy over time on EMNIST. Maximum training accuracy is 93.64%.

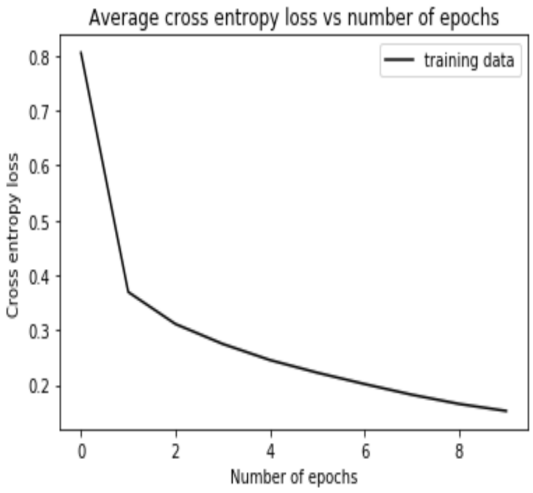


Figure 43: Mean training loss vs epochs

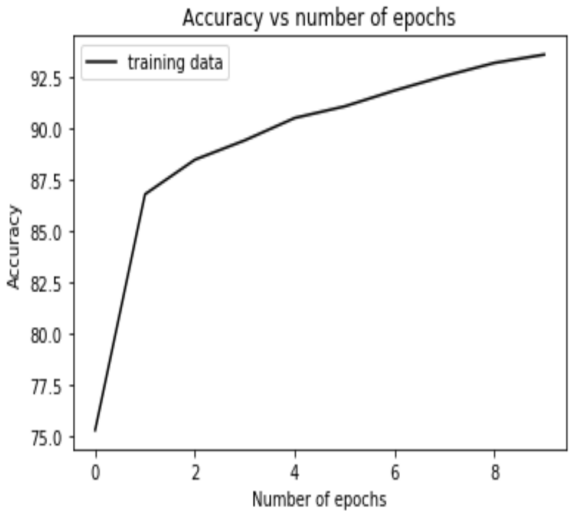


Figure 44: Accuracy vs epochs

7.1.5 Code/Writeup

The dataset in question is data1 which is the NIST36 dataset

1. Skip Connections
   Maximum accuracy is 96.51%. Test accuracy is 87.44%

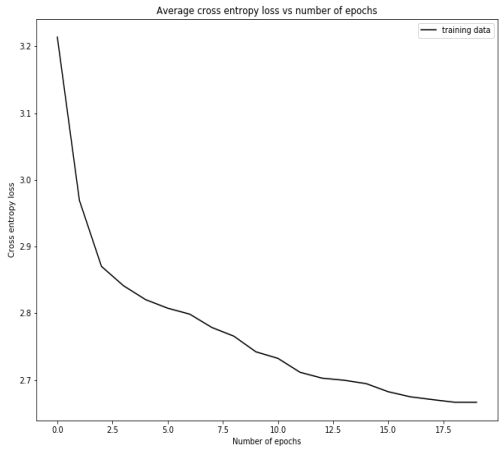


Figure 45: Mean training loss vs epochs

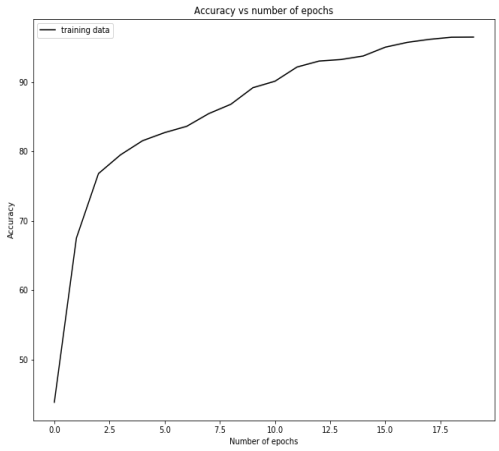


Figure 46: Accuracy vs epochs

2. Residual Connections
   Maximum training accuracy is 94.51%. Test accuracy is 87.78%.

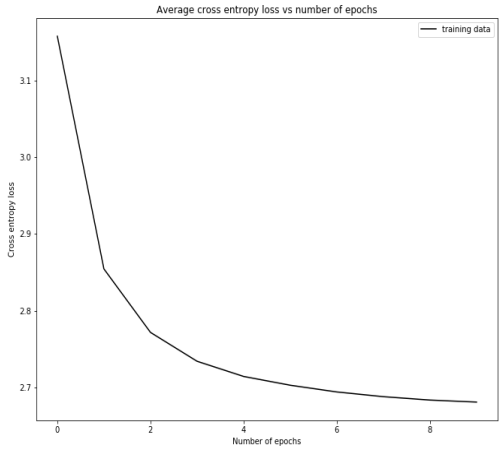


Figure 47: Mean training loss vs epochs

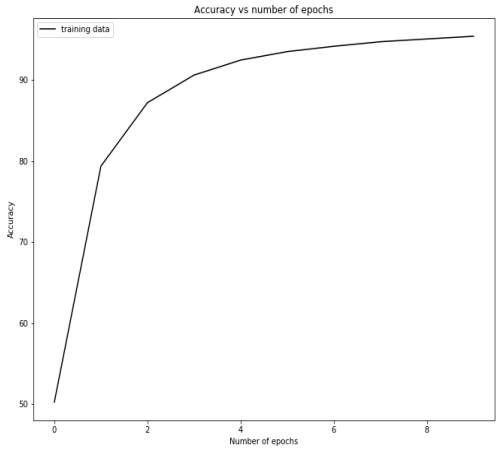


Figure 48: Accuracy vs epochs

3. A creative design that you want to try

   Maximum training accuracy is 86.39%. Test accuracy is 76.90%.
   The design I deployed is ResNet in ResNet courtsey of [1]. According to this paper, the traditional CNN can learn the abstract features of the image, while the residual CNN or residual block can learn the residual difference which lurks behind the image. The main goal was to combine these two in order to attain better performance. Both skip connections and residual blocks are deployed. The architecture is drawn in the later question.
   The network has two general residual blocks, and the whole process can be divided into residual stream and transient stream. The two streams were concatenated at the end. Maxpooling is done to reduce dimensions where essential.
   The performance on the test set is a little below other methods for detecting handwriting. I suspect this is either due to overfitting or the model fails to learn complex features. In theory, this method makes sense but tuning layers and deciding which ones must be concatenated or added for optimum results harms performance.

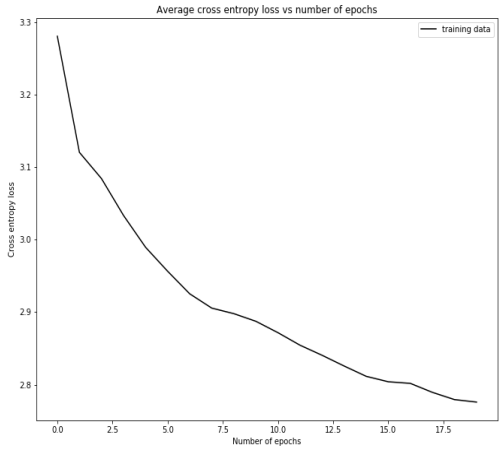


Figure 49: Mean training loss vs epochs

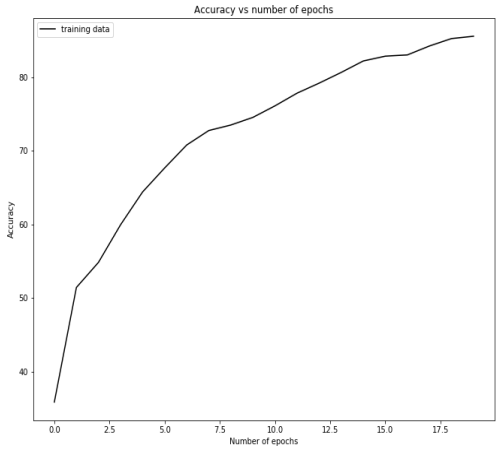


Figure 50: Accuracy vs epochs

1. The conceptual difference between residual connections and skip connections. Back it with equations and intuitions! [10 points]

   Skip connection is basically skipping some connections in the neural network and feeding the output of one layer to another layer further ahead skipping a few layers in between.
   Traditionally skip connections were of 2 types: short skip connections and long skip connections. Short skip connections are now mostly called residual connections and include adding output of a convolution layer to an earlier layer. Long skip connections are now often referred to as just skip connections and are used in networks like UNet and DenseNet. They involve concatenating the output of a convolutional layer to a prior layer.

   Mathematically speaking in the case of **residual connections**, the network attempts to learn a residue. Let us consider a neural network whose input is x. We aim to learn the true distribution of x denoted by $H(x)$. The residue is denoted by

   $$R(x) = H(x) - x \tag{5}$$

   $$H(x) = R(x) + x \tag{6}$$

   The network in its entirety is trying to learn $H(x)$. But since x is being added in the final layer, the individual residual blocks are actually attempting to learn only the residue $R(x)$. Hence these connections are called residual block connections.

   For traditional **skip connections** like the ones in unet and densenet, the network in its entirety attempts to learn a hypothesis $H(x)$. The network with the skip connection would learn a function F(x) which is representative of some fine features. The general understanding is that coarse features are learnt in the early stage of the network and they will be represented by x. But they are often lost during upsampling and as we proceed deeper in the network. So for optimum results, F(x) and x should be concatenated.

   $$H(x) = concatenate(F(x), x) \tag{7}$$

   So basically residual blocks learn residues and skip connections facilitate learning of both fine and coarse features. They are shown in the figures below.

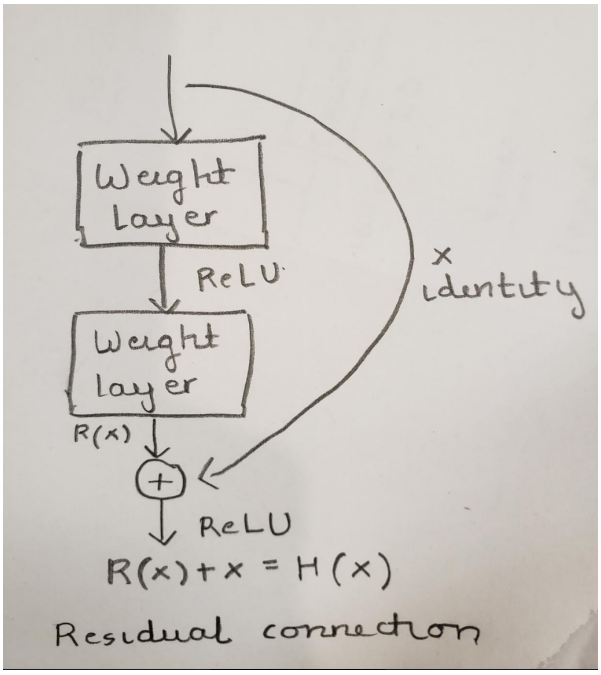


Figure 51: Residual connection

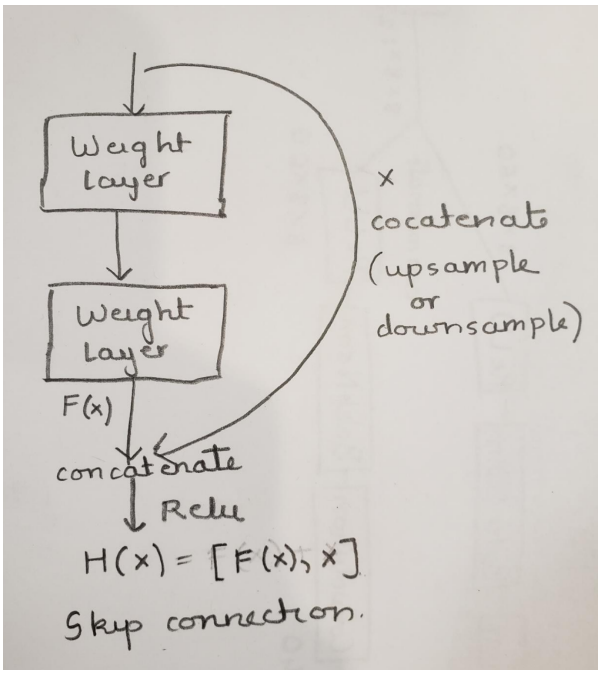


Figure 52: Skip connection

Both of them help avoid the vanishing gradient problem. They facilitate the uninterrupted gradient flow during backpropagation to update the network. They also facilitate faster convergence during training.

The image below shows us the losses of Resnet -156 with and without addition of previous layers. Addition of an earlier layer to a latter one facilitates a smoother loss function which means better performance.

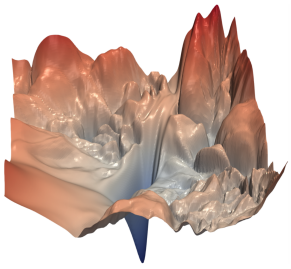


Figure 53: Resnet-156 loss without connections through addition

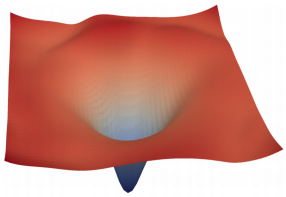


Figure 54: Resnet-156 loss with connections through addition

Figure 55: Visualising loss space in Neural networks for residual connections https://arxiv.org/abs/1712.09913

2. For each network design explain your network design (Draw it!) and back it some reasoning, why you think this can work better? Feature visualization might help, think about gradients, use the ideas mentioned in the papers

Skip connections

The network makes use of concatenated skip connections.The network as shown in the figure below has 2 layers followed by ReLU activation. Each main layer has 2 convolutional layers which are individually batch normalized and activated. The input to that main layer is then cocatenated with the output of the same main layer.

Network depth is limited by the issue of vanishing gradients during back-propagation. Skip connections are added around non -linearities and they facilitate the uninterrupted gradient flow to update the network. Thus the problem of vanishing gradient is avoided.
It also facilitates faster convergence during training.
In this case as shown in in the features, it allows the subsequent layers to re-use middle representations, maintaining more information which can lead to better performances.

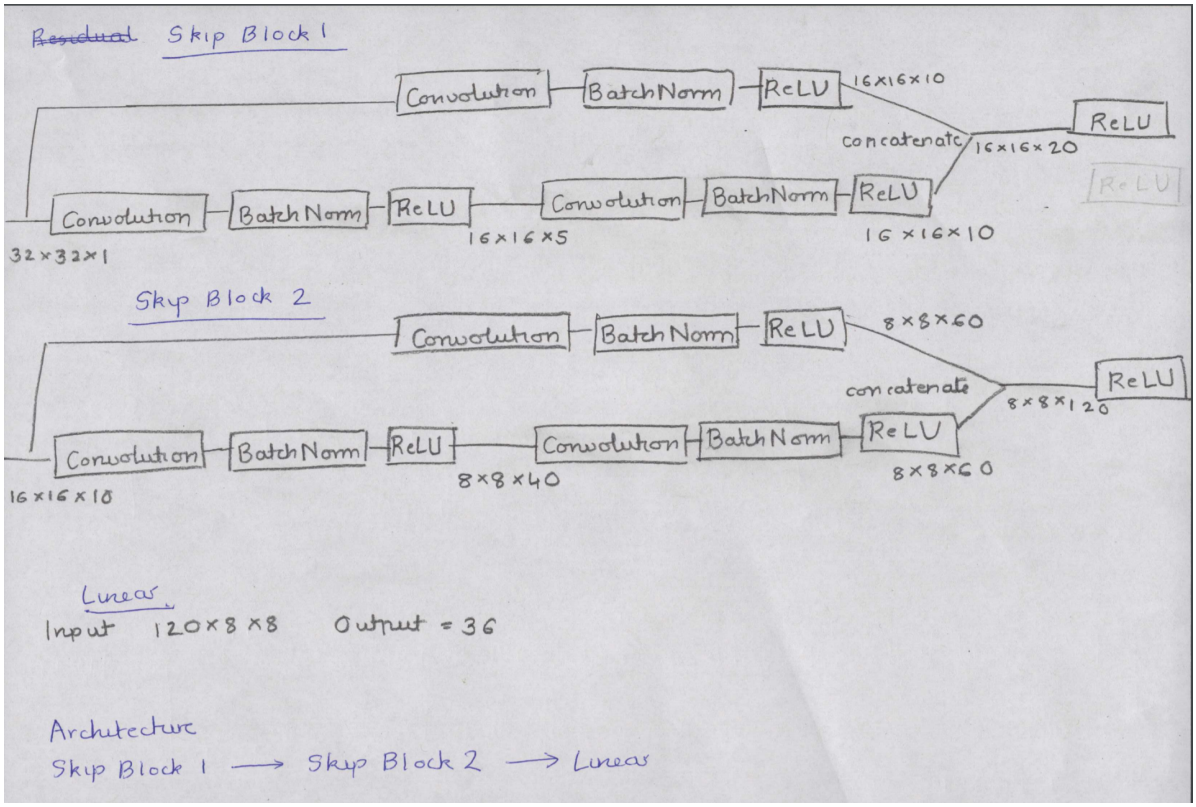


Figure 56: Skip connections architecture for my algorithm

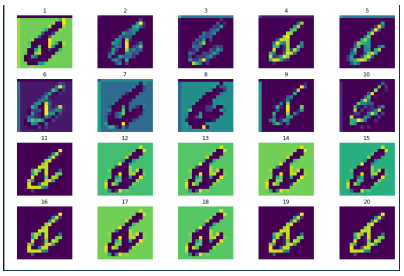


Figure 57: Features after first convolution

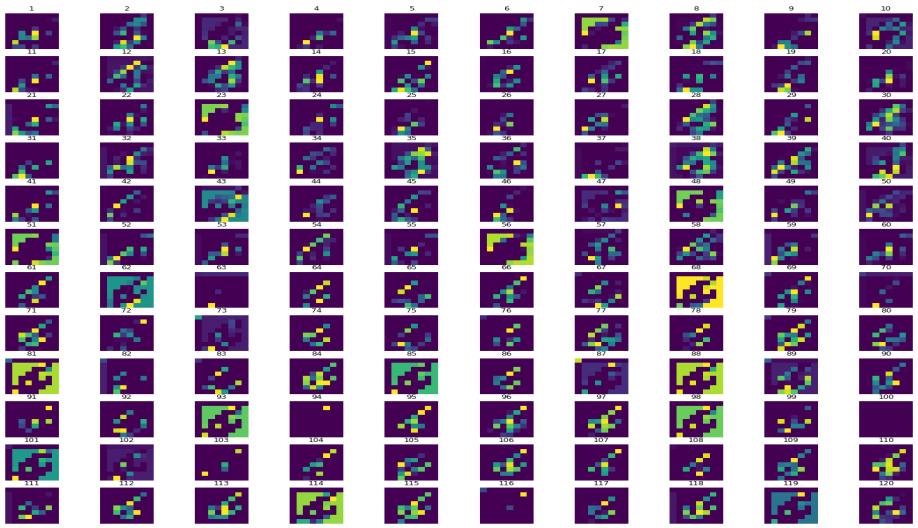


Figure 58: Features after second convolution

Residual connections

The network makes use of connections with element wise addition..The network as shown in the figure below has 2 layers followed by ReLU activation. Each main layer has 2 convolutional layers which are individually batch normalized and activated. The input to that main layer is then added to the output of the same main layer.

It can be viewed as an iterative estimation procedure to some extent where the features are refined through the various layers of the network as shown in the figure and a residue is learnt in each layer which is added to the original input.

It is a compact solution that keeps the number of features fixed.

It helps fix the vanishing gradient problem as it provides a clear path to the gradient for backpropagation.

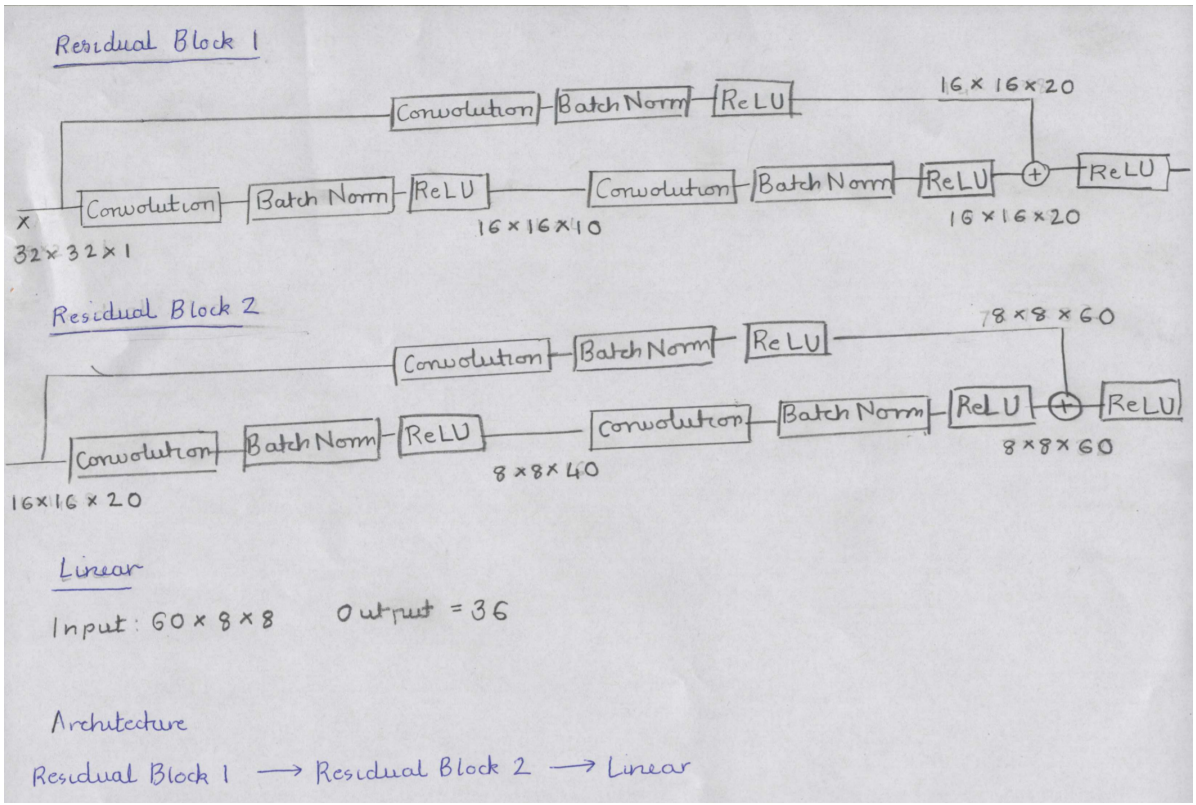


Figure 59: Residual blocks architecture for my algorithm

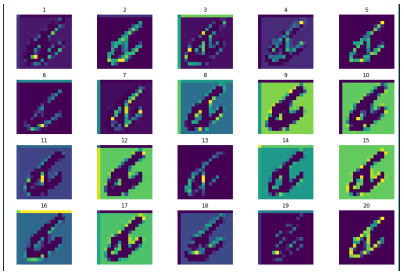


Figure 60: Features after first convolution

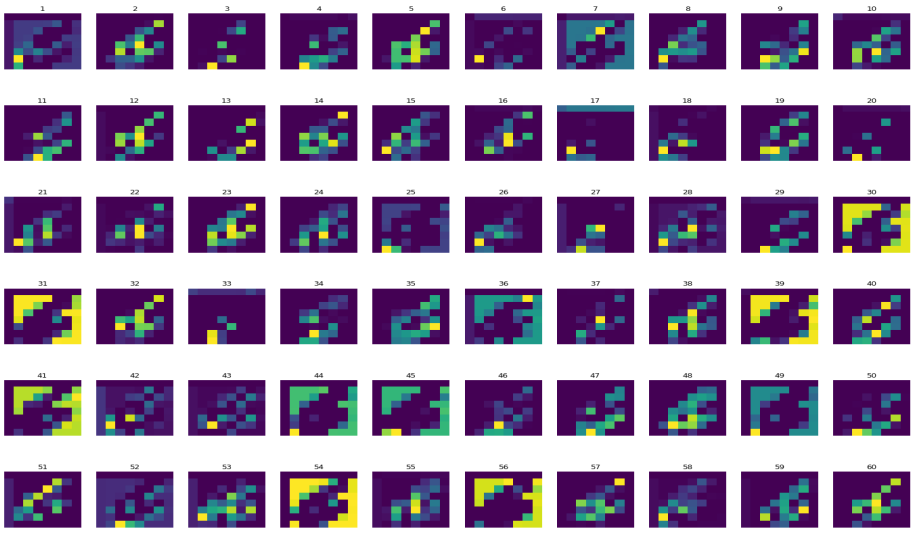


Figure 61: Features after second convolution

Creative design

The network has two general residual blocks, and the whole process can be divided into residual stream and transient stream. The two streams were cocatenated at the end. Maxpooling is done to reduce dimensions where essential.

The performance on the test set is a little below other methods for detecting handwriting. I suspect this is either due to overfitting or the model fails to learn complex features. In theory, this method makes sense but tuning layers and deciding which ones must be concatenated or added for optimum results harms performance.

According to this paper, the traditional CNN can learn the abstract features of the image, while the residual CNN or residual block can learn the residual difference which lurks behind the image. The main goal was to combine these two in order to attain better performance. However it fails at learning appropriate features as is evidenced by the image below.

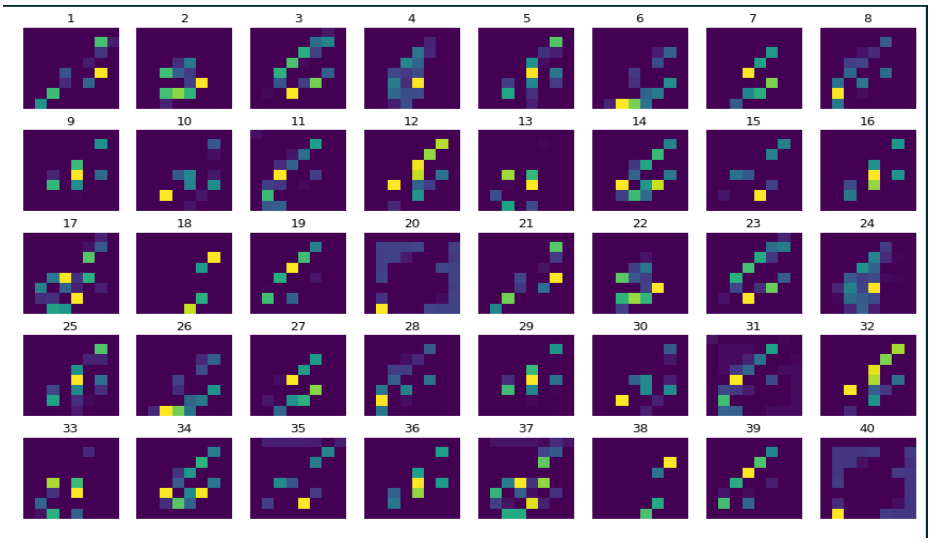


Figure 62: Features after first passage through model

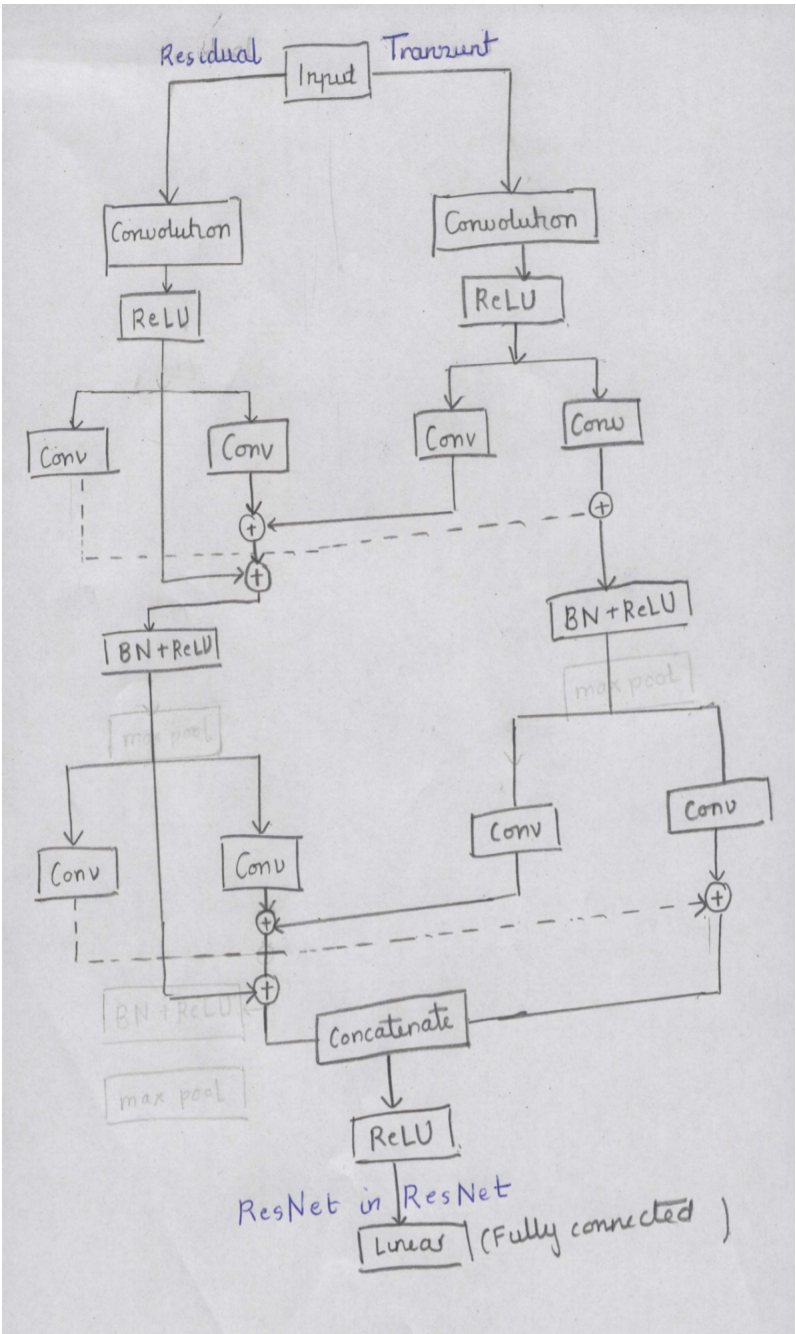


Figure 63: ResNet in ResNet

7.2. Fine Tuning

7.2.1 Code/Writeup

Training accuracy after fine tuning with squeezenet is over 96.81%. It only requires around 10 epochs to converge and then another 10 epochs with a small learning rate to be fine tuned. Compared to squeezenet, the model from scratch takes a lot longer to train and over a 100 episodes. The accuracy attained after

this extensive period is 93.01 % and is not as high as squeezenet. Squeeznet has better recognition accuracy and the ability to converge much faster.

8. References

[1] S. Targ, D. Almeida, and K. Lyman, “Resnet in Resnet: Generalizing Residual Architectures,” ArXiv:1603.08029 [cs.LG], March 2016.