Deep Learning: Minor 1

by

Palaskar Adarsh Mahesh

B20EE087

February 13, 2023

1 Question 1

1.1 Implementation Details:

1. DD = 07, MM = 08, YY = 02
2. ABC = 087
3. MNIST Dataset
4. Xavier Weight Initialization
5. Data Augmentation: Roatate by 10 degrees and add Gaussian Noise
6. Target Classes: 1,3,5,7,9
7. 5 convolution layers and 1 average pooling layer, with 8 filters in the first convolution layer.
8. 1 Fully Connected Layers with 256 nodes.

1.2 Xavier Weight Initialization:

Using this weight initialization technique, the neural network's weights are set up so that the variance of each layer's inputs and outputs is equal. To accomplish this, we divide each input layer's weights by 1/sqrt(n), where n is the number of inputs to that layer. This can prevent the gradients from exploding or disappearing during backpropagation. Therefore using Xavier weight initialization, the model can learn more efficiently, converge faster, have better generalization, and reduce overfitting.

1.3 Data Augmentation:

We perform data augmentation to increase the diversity of the dataset so that we can overcome problems like overfitting and improve generalization of the model. In this question, we perform a random augmentation of either rotating the image by 10 degrees in the anticlockwise direction, or add gaussian noise to the image, with a probability of 0.5.

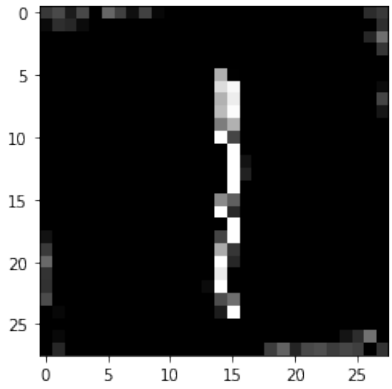


Figure 1: Digit 1 after Augmentation

1.4 Data Visualization using PCA:

Using PCA(Principal Component Analysis) we can reduce the number of features from 784 to 2 to visualize the different classes. Different classes are labeled with different colors in the figure below:

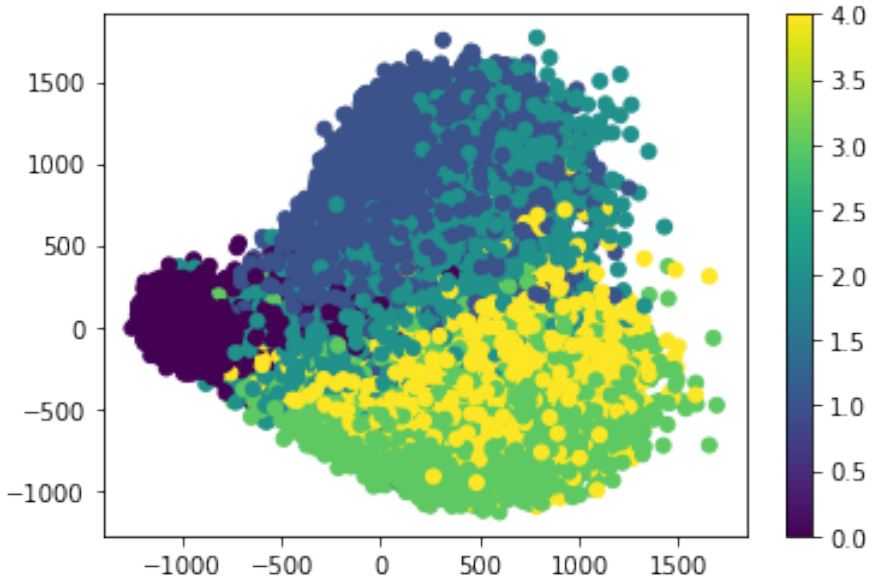


Figure 2: Visualization of Classes

1.5 Convolution Neural Network Model:

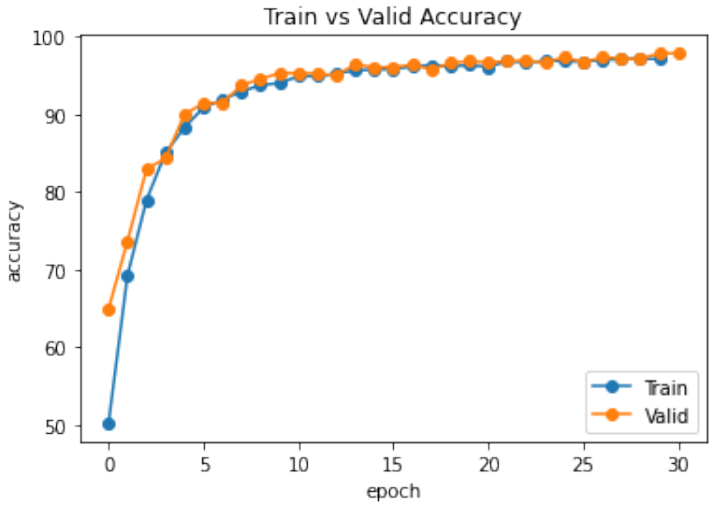
The implemented model has the following architecture explained below. We have used a kernel size of 3x3 to improve local recognition of features which would be lower if we use a larger filter size. A padding of 1 is added so that the edges and corners of the image are not neglected by the model and also helps fit proper output dimensions. Finally, we have used the ReLU activation function to aid computation speeds and make sure that we do not face the issue of vanishing gradients.

1. **conv1:** A 2D convolution layer with 1 input channel, 8 output channels, a kernel size of 3x3, a stride of 1, and padding of 1.
2. **relu1:** A ReLU activation function layer.
3. **conv2:** A 2D convolution layer with 8 input channels, 16 output channels, a kernel size of 3x3, a stride of 1, and padding of 1.
4. **relu2:** A ReLU activation function layer.
5. **conv3:** A 2D convolution layer with 16 input channels, 32 output channels, a kernel size of 3x3, a stride of 1, and padding of 1.
6. **relu3:** A ReLU activation function layer.
7. **conv4:** A 2D convolution layer with 32 input channels, 64 output channels, a kernel size of 3x3, a stride of 1, and padding of 1.
8. **relu4:** A ReLU activation function layer.
9. **conv5:** A 2D convolution layer with 64 input channels, 256 output channels, a kernel size of 3x3, a stride of 1, and padding of 1.
10. **relu5:** A ReLU activation function layer.
11. pool: An adaptive average pooling layer that reduces the spatial dimension of the data to 1x1.
12. **fc:** A fully connected layer with 256 input features and 5 output nodes which are required for prediction.

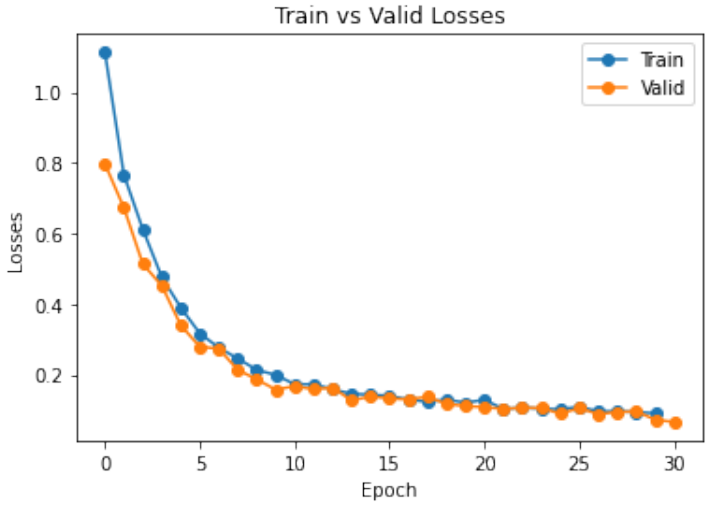
The forward method implements the forward pass of the network by applying the defined layers to the input tensor x. The input tensor x is first passed through conv1, followed by relu1 activation. This is repeated for the remaining conv and relu layers. Finally, the output of conv5 is passed through the pool layer and the view method is used to reshape the tensor into a 1D tensor. The reshaped tensor is then passed through the fully connected layer to produce the final output.

1.6 Results:

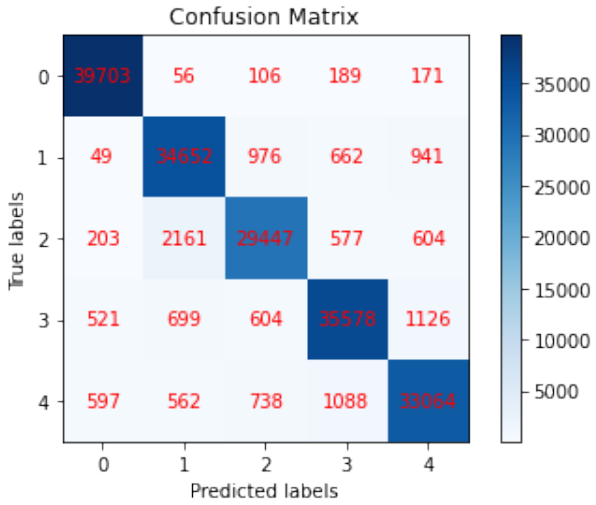
1.6.1 Train vs Validation Accuracy:



1.6.2 Train vs Validation Losses:



1.6.3 Confusion Matrix:



1.6.4 Classification Report:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	40225
1	0.91	0.93	0.92	37280
2	0.92	0.89	0.91	32992
3	0.93	0.92	0.93	38528
4	0.92	0.92	0.92	36049
accuracy			0.93	185074
macro avg	0.93	0.93	0.93	185074
weighted avg	0.93	0.93	0.93	185074

2 Question 2

Same details as Question 1 with the following additions/changes:

1. Number of AutoEncoder layer = 3
2. Classification Layer = 512 nodes

2.1 AutoEncoder and Classifier implementation Details:

The AutoEncoder reconstructs the input data by encoding it into a lower-dimensional space (latent representation or encoding) and then decoding it back to the original space. It has two parts, the Encoder, and the Decoder.

2.1.1 Encoder:

The encoder compresses the input data into a lower-dimensional representation. It has three convolution layers with increasing filters, a ReLU activation function applied after each layer and an adaptive average pooling layer at the end. The pooling layer is used to reduce the spatial dimensions of the output from the previous layer. We have used ReLU to reduce the number and computations and also ensure that there is no issue of vanishing or exploding gradients. The architecture has 3 convolution layers similar to the model in question 1,

2.1.2 Decoder:

The decoder takes the lower-dimensional representation and expands it back to the original dimensions. It consists of three linear layers with ReLU activation functions applied after each layer and a sigmoid activation function at the end. The sigmoid function maps the values between 0 and 1, which represent the reconstructed image's pixel intensities.

2.1.3 Classifier:

The classifier takes only the encoded representation from the autoencoder and outputs a class label. It consists of a single linear layer with 5 output units, one for each class, for which the input is a fully connected layer of 512 nodes of the encoder.

2.1.4 Training Process:

We first add noise to the input images, and the autoencoder is trained on these noisy images. The reconstructed images are then compared with the original images in the training dataset, and the corresponding MSE(Mean Squared Error) Loss is calculated. The autoencoder thus learns various features of the image in lower dimensions, and the reconstructed images become clear after each epoch.

If we do not use noisy images, then the spatial representation will be trained to represent the identity matrix that generates the same image present in the input; thus, the model does not learn anything.

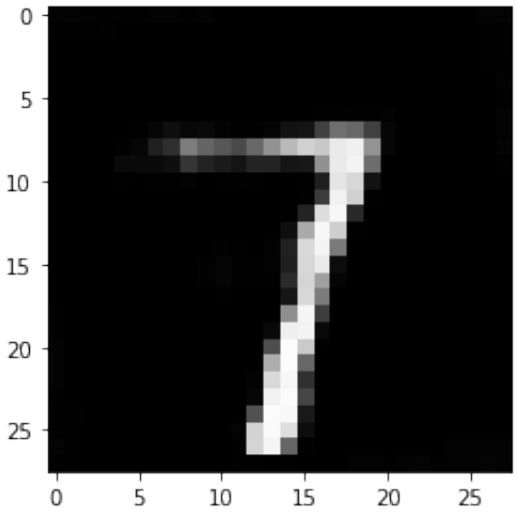


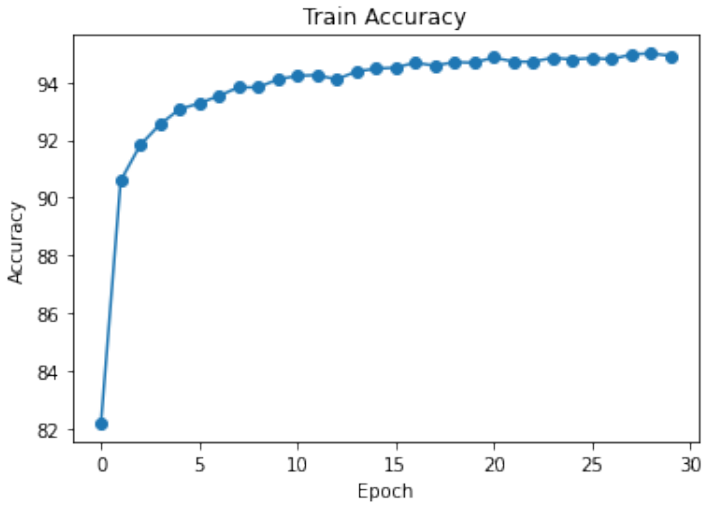
Figure 3: One of the Reconstructed Images

2.1.5 Classification Process:

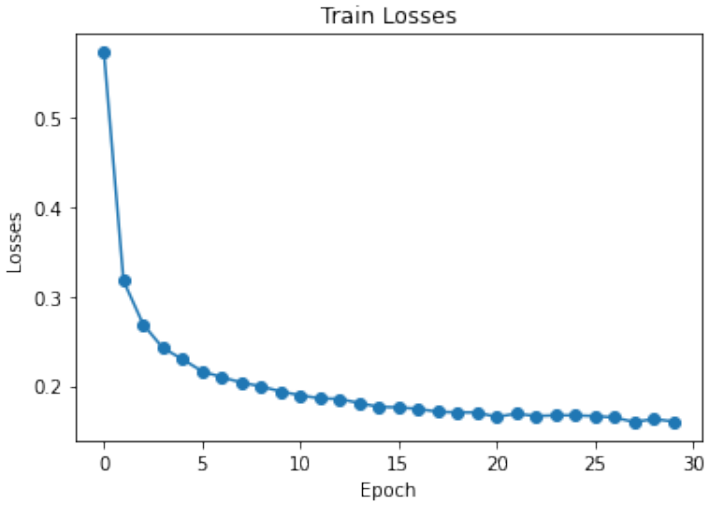
To perform the classification of digits, we exclude the decoder layer and use the encoder features as input to our linear classifier, and train this artificial neural network for 30 epochs to finally get the classified outputs.

2.2 Results:

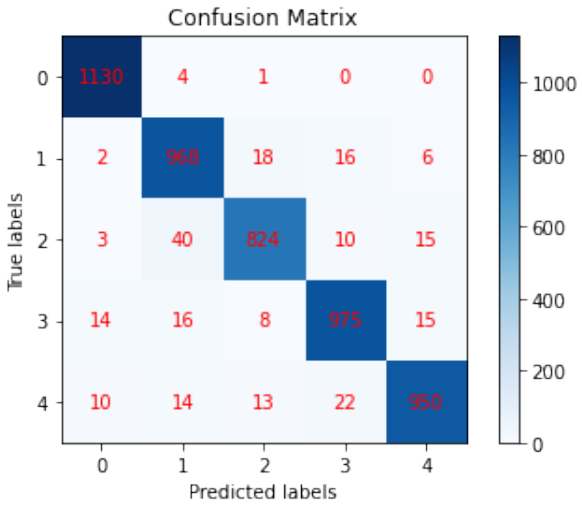
2.2.1 Train Accuracy vs No. of epochs:



2.2.2 Train Losses vs No. of epochs:



2.2.3 Confusion Matrix:



2.2.4 Classification Report:

	precision	recall	f1-score	support
0	0.97	1.00	0.99	1135
1	0.93	0.96	0.94	1010
2	0.95	0.92	0.94	892
3	0.95	0.95	0.95	1028
4	0.96	0.94	0.95	1009
accuracy			0.96	5074
macro avg	0.95	0.95	0.95	5074
weighted avg	0.96	0.96	0.96	5074

3 Comparision of CNN and AutoEncoder

A CNN is a supervised learning algorithm that learns the mapping from an input image to its corresponding neuron for classification. It minimizes the error between the predicted and true labels.

An AutoEncoder is an unsupervised learning algorithm that reconstructs its input by encoding input data into lower dimensions and decoding back to its original form. It minimizes the reconstruction error between the input and the reconstructed output.

Obtained accuracies for both models on the test dataset:

1. Convolutional Neural Network = 97.79
2. AutoEncoder Classifier = 95.52

Even though the accuracy of CNN is slightly less than that of AutoEncoder, upon closer observation of the classification reports, it is seen that all the precision, recall, and f1-scores are highers for numbers 3,5,7, and 9 for the auto-encoders. Therefore, on a larger and more generalized dataset, we can say that the autoencoder classifier model can give more accuracy across all the classes. At the same time, it will also enable the representation of the dataset in a lower dimension thus making it more compact.