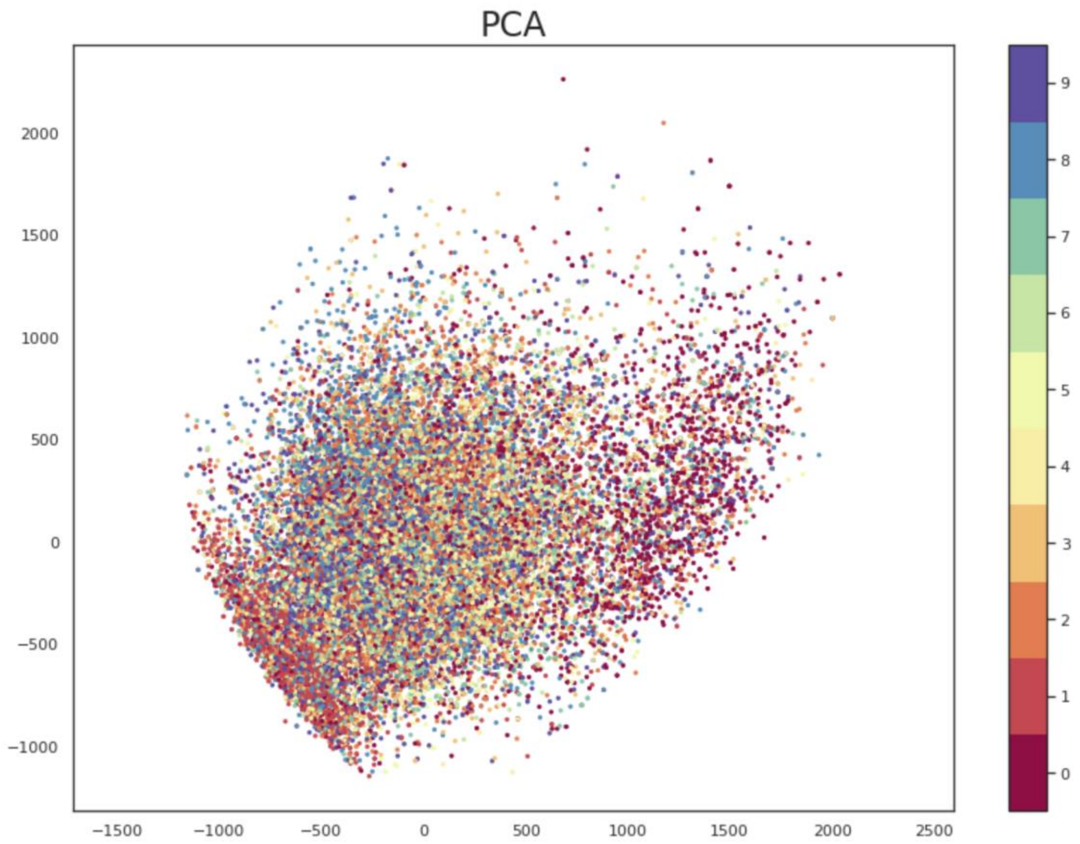
To run A4codes.py, it is required to have torch and numpy installed.

Method 1: Dimensionality Reduction and Clustering

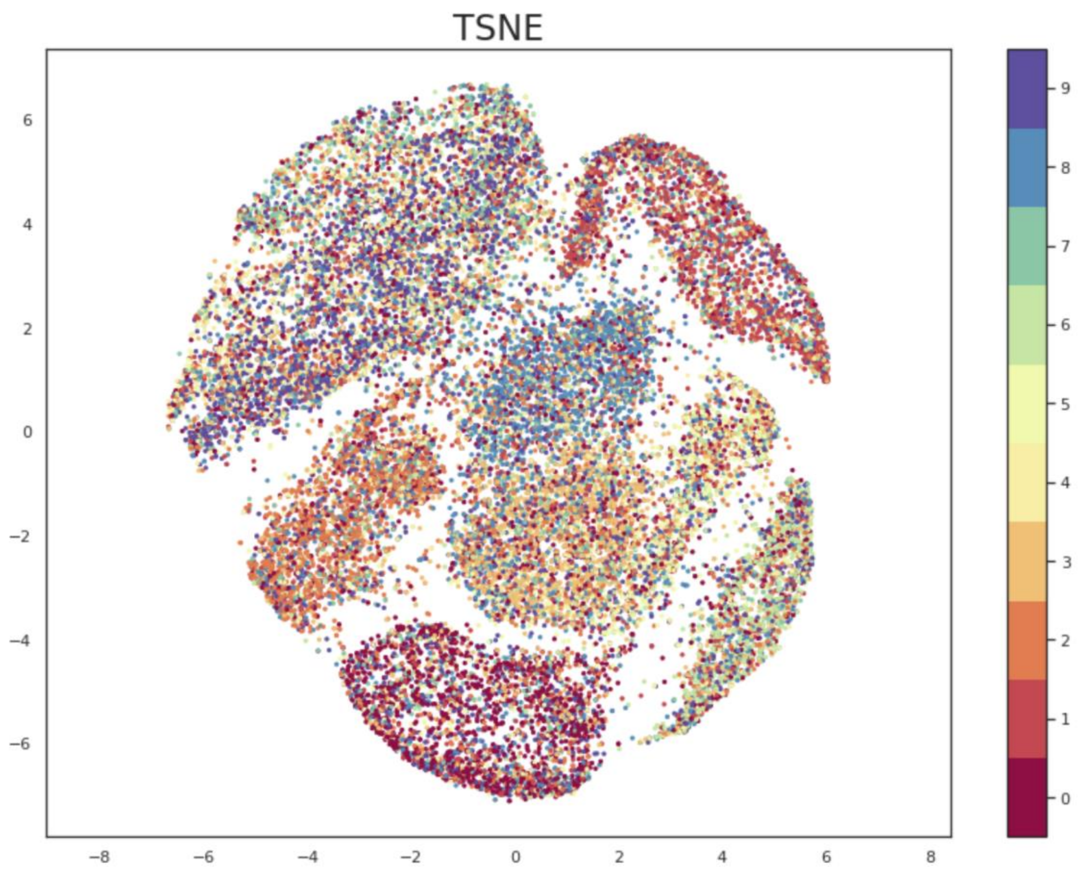
The 2352-dimensional nature of these vectors gives influences a dimensionality reduction model, especially given the amount of negative space in these images. The hypothetical algorithm for this approach was to separate the large image into its three components and label the three of them with the same label that was given. Next, perform dimensionality reduction to ideally see clusters of similar images forming in two-dimensional space. Then, a clustering algorithm could divide the space into 10 clusters, representing each number. Finally, assuming that the test data was chosen from an equally distributed set, each cluster could be labeled correctly by using the most common label from within that cluster that was given beforehand. This labeling would work since there would be a 33% chance of a number having a correct label, and a $66\%/9 = 7.33\%$ of having any other individual label, so the correct label will be the most common in a large enough dataset.

In class, we saw PCA as a dimensionality reduction tool. The following figure is the PCA projection of the training dataset to two dimensions, using the function PCA function from assignment 3.



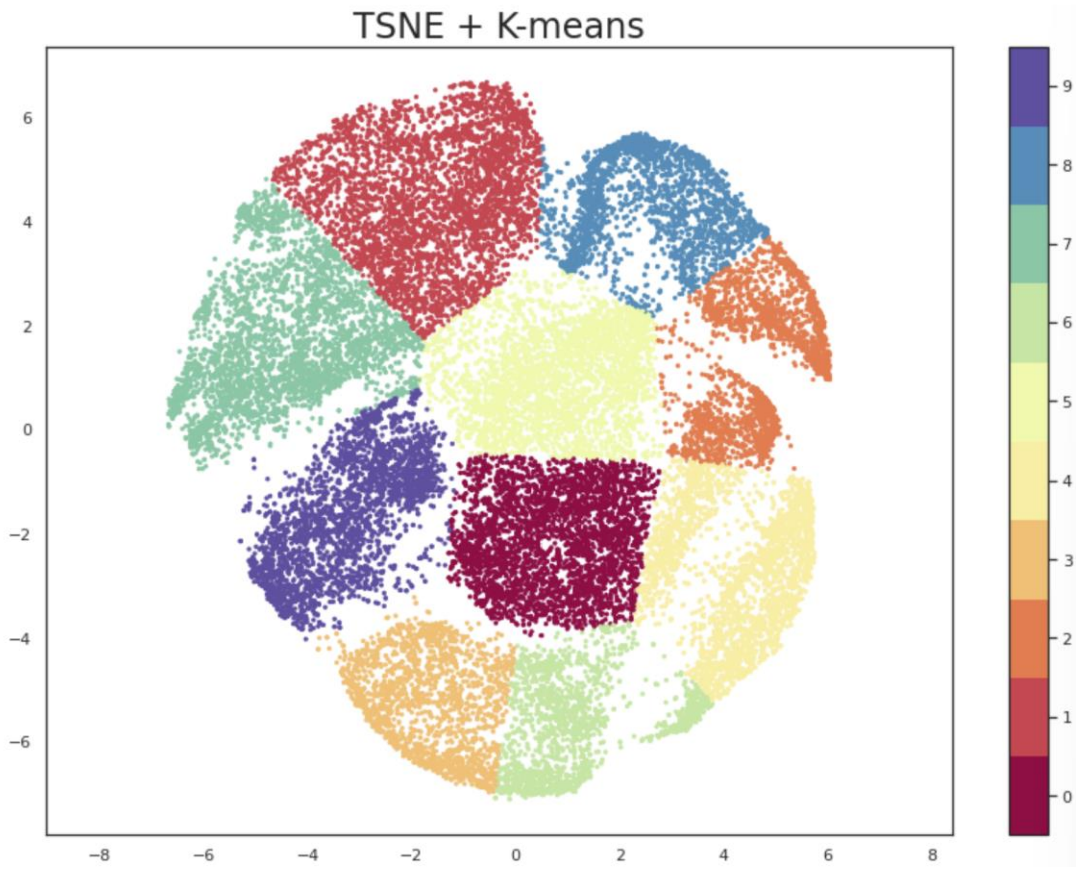
The model is trying to place similar numbers together. The zeroes are far to the right and the ones are far to the left and even some of the other numbers show some separation like how the eights are further right than the fives. However, this model does not do a good enough job of separating the numbers into distinct groups for a clustering algorithm to be able to separate them clearly into 10 unique clusters.

Another dimensionality reduction tool that is commonly used on the MNIST dataset is TNSE. The following figure is the TSNE projection of the training dataset to two dimensions using sklearn.manifold.TSNE.



This model is working better than the PCA model. The reduction works well once again for some numbers like zero and one which have their own clusters, and this time with unique borders. For other numbers such as four, seven, and nine, the model continues to have trouble and places the vectors together in a large cluster in the top left.

The next step is clustering. In class, we saw K-means clustering for this purpose. The problem with this approach is that K-means is based on clusters having a center so the result is not very useful on clusters that have long and narrow shapes like those in the figure above. The following figure is the TSNE projection after K-means clustering.



There exist other algorithms which could be more performant for density-based clustering, however, this idea was not further explored for this report.

Method 2: Convolutional Neural Network

Convolutional neural networks work well on image data since they can isolate unique features that are common to image classes. The model used in this assignment is inspired by the VGG image classification network seen in class.

The team working on this model found that an effective architecture for image classification was to have several convolutional layers. These layers used a padding size with zeros which kept the dimensions of the image intact. This is followed by a 2x2 pooling layer which effectively halved the dimension of the image and pass the new shape into a similar convolutional structure. These steps were repeated and the resulting matrix was flattened and passed into a feedforward network. The result is a vector with the same length as the number of unique labels in the dataset which would indicate the result that the model has chosen.

The paper for this model (Simonyan, 2015) discusses certain parameters accounted for in this process. Several of these parameters include the following.

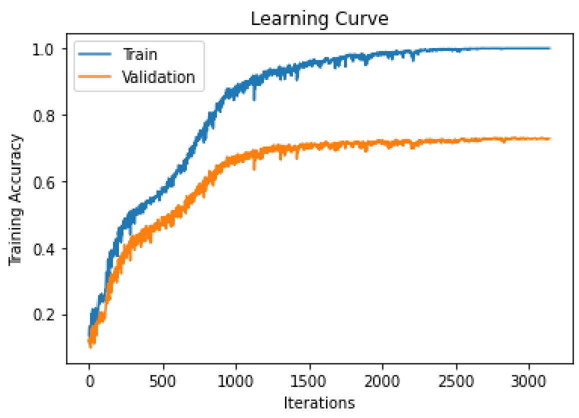
1. “The only pre-processing we do is subtracting the mean RGB value, computed on the training set, from each pixel”.
2. “The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution”.
3. “use of very small (3 × 3) convolution filters in all layers”.
4. “A stack of convolutional layers … is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each”.
5. “The final layer is the soft-max layer”.
6. “All hidden layers are equipped with the rectification (ReLU) non-linearity”.
7. “the training is carried out by optimising the multinomial logistic regression objective using mini-batch gradient descent (based on back-propagation) with momentum”.
8. “The batch size was set to 256, momentum to 0.9”.
9. “The training was regularised by weight decay (the L2 penalty multiplier set to 5e−4)”.
10. “The learning rate was initially set to 10e−2”.
11. “The main difference is that we replace the logistic regression objective with a Euclidean loss”

Several of these design choices were accounted for in this report. Two pooling layers are used rather than the four used in the paper since $28/(2 * 2) = 7$ which is an odd number and another pooling layer loses information on a reduction to 3x3 and padding with zeros causes more noise at this point than a third convolutional layer can recover in accuracy. Similarly, a convolution stride of 1 and padding to preserve shape in convolution layers were used for the same reason of 28x28 being too small for other options.

Preprocessing was tested by adding padding of zeros around each image to increase the dimension to 32x32 to that $32/(2 * 2 * 2) = 4$ and a third convolutional layer could be added, but this decreased the accuracy of the model. Preprocessing was also tested by subtracting the mean pixel value from each pixel on the training set, but this did not affect learning at all. Therefore, there is no preprocessing used in this report's model.

The softmax layer, ReLU activation, gradient descent, and Euclidian loss were used in this model and were tested minimally.

Testing with 30 epochs (batch size 100), the model shows that it can completely learn the training dataset in about 20 epochs. For this reason, the model uses 20 epochs to learn, and an additional 5 more on a smaller learning rate for confirmation. This model trains in about 1.5 minutes with GPU on Google Collab.



Various decays and momentums were tested for the gradient descent function. The following table shows validation set accuracies for decay against momentum.

	Decay					
Momentum	0	5e-6	5e-5	5e-4	5e-3	5e-2
0.95	0.7088	0.7092	0.7080	0.7470	0.7024	0.0972
0.9	0.7398	0.7424	0.7398	0.7396	0.541	0.1000
0.85	0.7390	0.7376	0.7402	0.7388	0.7206	0.3424
0.8	0.7358	0.7356	0.6974	0.7360	0.7172	0.3316
0.75	0.7332	0.6884	0.7302	0.7332	0.6990	0.2106

All numbers are similar and there is no clear trend that could show outliers, so the largest value was picked: decay = 5e-4, momentum = 0.95.

Learning rates were tested on 30 epochs. The following table shows validation accuracies for these values.

Learning rate			
1e-4	1e-3	1e-2	1e-1
0.3680	0.7454	0.7012	0.0972

In this test, 1e-3 was able to fully learn the training, and 1e-2 made some mistakes in learning the dataset. For efficiency, it was found that 1e-2 for 20 epochs and 1e-3 for another five can consistently learn the training dataset so this is what is used for this report.

The number of fully connected layers was tested where the first layer reduces to 512 nodes, and the last layer reduces to a vector of length 10. The following table shows validation accuracies for these values.

Number of FC layers			
1	2	3	4
0.586	0.756	0.758	0.713

Based on this data, either two or three FC layers are appropriate choices. During the other tests shown in this report, the choice of 3 FC layers was shown to work slightly better so this was chosen for the model.

The number of nodes in the fully connected layers was tested against the size of the convolution filter (with padding that keeps shape intact). The following table shows validation accuracies for FC node count against convolutional filter size.

	Convolutional filter size			
Fully connected node count		3x3	5x5	7x7
	1024	0.7508	0.7094	0.7584
	512	0.7414	0.7546	0.7638
	256	0.7496	0.7540	0.7132
	128	0.6050	0.7078	0.6688
	64	0.5518	0.7556	0.7574
	32	0.7420	0.7552	0.7094
	16	0.6936	0.7490	0.759

There is once again not an obvious trend that would show any outliers, so the parameters that gave the largest accuracies were chosen: filter size = 7x7, fully connected node count = 512.

Based on these findings, the final Convolutional neural network model achieves an accuracy of about 76% on the validation data.

References

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*. https://doi.org/10.48550/arXiv.1409.1556