CS148 HW 3 Anish Shenoy
Github Link: https://github.com/Anish2454/caltech-ee148-spring2020-hw03

Question 1

Train the default **`ConvNet`** and compute accuracy values on your train and val sets. Then add some form of data augmentation in the training dataloader, re-train, and re-compute the accuracy values. *(1 point) Describe your data augmentation scheme, report the results with and without augmentation, and discuss what you see:*

Default (No Augmentation) Accuracy (10 epochs):

- Validation: Average loss: 0.0908, Accuracy: 8753/8995 (97%)
- Train: Average loss: 0.0780, Accuracy: 49801/51005 (98%)

After looking through the list of available augmentations, I decided to randomly apply this set of transforms:

- transforms.RandomRotation(180)
- transforms.RandomVerticalFlip
- transforms.GaussianBlur(*3*, *sigma=(0.1*, *1)*)

I chose these augmentations because I thought they did a good job of simulating real-world differences in images like handwriting, orientation, size, etc. I also tried resizing the image but I quickly realized that this broke the model since all inputs need to be of the same size.

I applied these augmentations only on the test set, leaving the validation set untouched. Here are the results of the initial set of augmentation:

- Validation set: Average loss: 0.1459, Accuracy: 8627/8995 (96%)
- Train set: Average loss: 0.3959, Accuracy: 44616/51005 (87%)

After displaying some of the augmented images, I realized flipping and blurring a lot wasn't a good idea because it changed the input too much. So I rotated less (30 degrees), removed the vertical flip, and reduced gaussian blur. Results:

- Validation set: Average loss: 0.0842, Accuracy: 8776/8995 (98%)
- Train set: Average loss: 0.1096, Accuracy: 49329/51005 (97%)

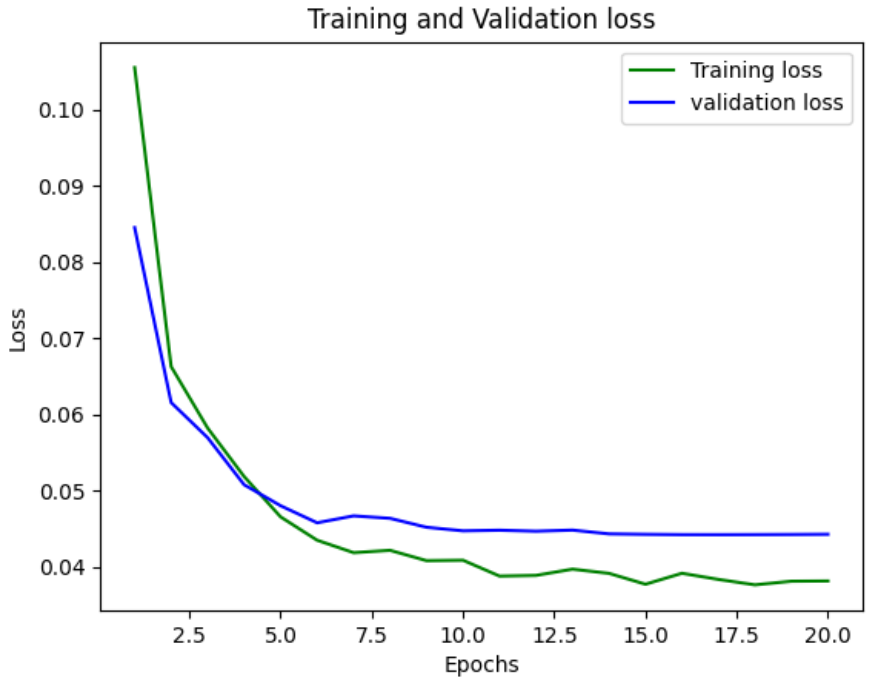
Augmenting the data seems to provide better generalization. The final results show that validation accuracy improved. Additionally, training accuracy decreased. This likely means the model was overfitting and augmentation helped regularize the model.

Question 2

(1 point) Discuss the process you went through to develop your architecture, and anything you observed as being particularly effective or ineffective.

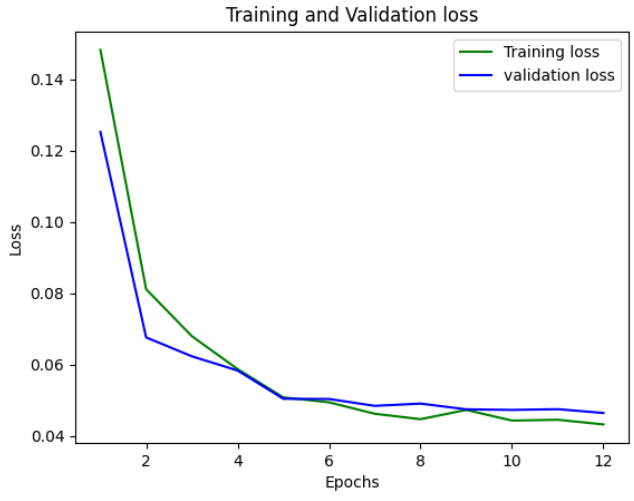
My first idea was to try the default model without any dropout layers. Results:

- Validation set: Average loss: 0.0447, Accuracy: 8879/8995 (99%)
- Train set: Average loss: 0.0408, Accuracy: 50390/51005 (99%)
- Plot:



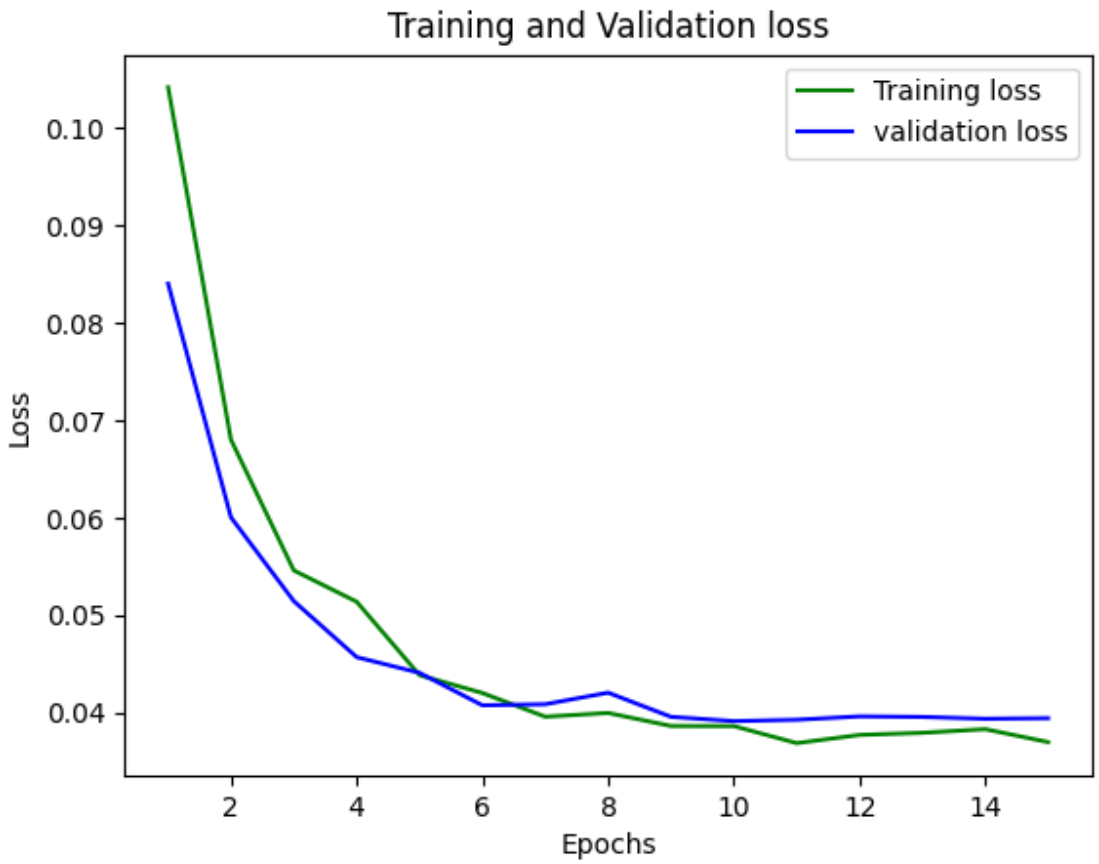
As expected, removing the dropout layers caused the model to overfit. This is shown by the large gap between validation and training loss. To reduce overfitting, I added back one dropout layer. Results:

- Validation set: Average loss: 0.0464, Accuracy: 8866/8995 (99%)
- Test set: Average loss: 0.0432, Accuracy: 50303/51005 (99%)
- Plot:



This brought the test error and validation error closer together, but it increased overall error for both the train and validation sets. To solve this, I added two batch-norm layers instead of dropout as a (hopefully) more effective form of regularization. According to the Resnet paper, we should add batch norm immediately after convolutions and before non-linearity. Results:

- Validation set: Average loss: 0.0395, Accuracy: 8886/8995 (99%)
- Train set: Average loss: 0.0370, Accuracy: 50439/51005 (99%)
- Plot:



This accuracy was the highest I could achieve.

Question 3

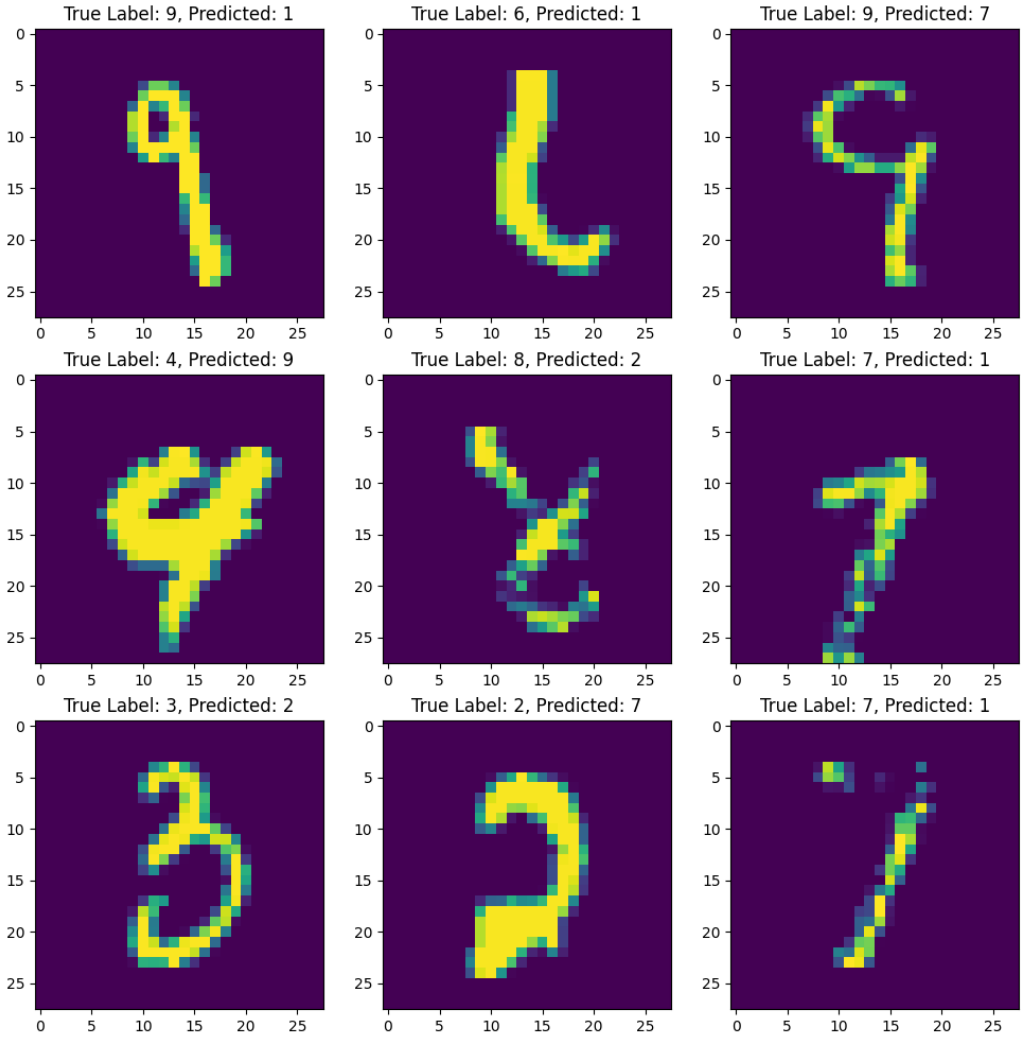
Re-train on a random subset of half, one quarter, one eighth, and one sixteenth of the training set and re-compute the accuracy values. *(1 point) Report the results. Plot the training and test error as a function of the number of training examples on log-log scale. Discuss what you see.*

- Test Set accuracy of tweaked model:
  - Test set: Average loss: 0.0286, Accuracy: 9908/10000 (99%)
- 50% of data:
  - Training set: Average loss: 0.0462, Accuracy: 29593/30000 (99%)
  - Test set: Average loss: 0.0413, Accuracy: 9858/10000 (99%)
- 25% of the data:
  - Training set: Average loss: 0.0584, Accuracy: 14738/15000 (98%)
  - Test set: Average loss: 0.0505, Accuracy: 9835/10000 (98%)
- 12.5% of the data:
  - Training set: Average loss: 0.0861, Accuracy: 7312/7500 (97%)
  - Test set: Average loss: 0.0743, Accuracy: 9757/10000 (98%)
- 6.25% of the data:
  - Test set: Average loss: 0.1188, Accuracy: 3621/3750 (97%)
  - Test set: Average loss: 0.1049, Accuracy: 9693/10000 (97%)

As the number of training samples grows, training and testing error decrease. This makes sense because more data allows the model to learn more about the underlying pattern.

Question 4

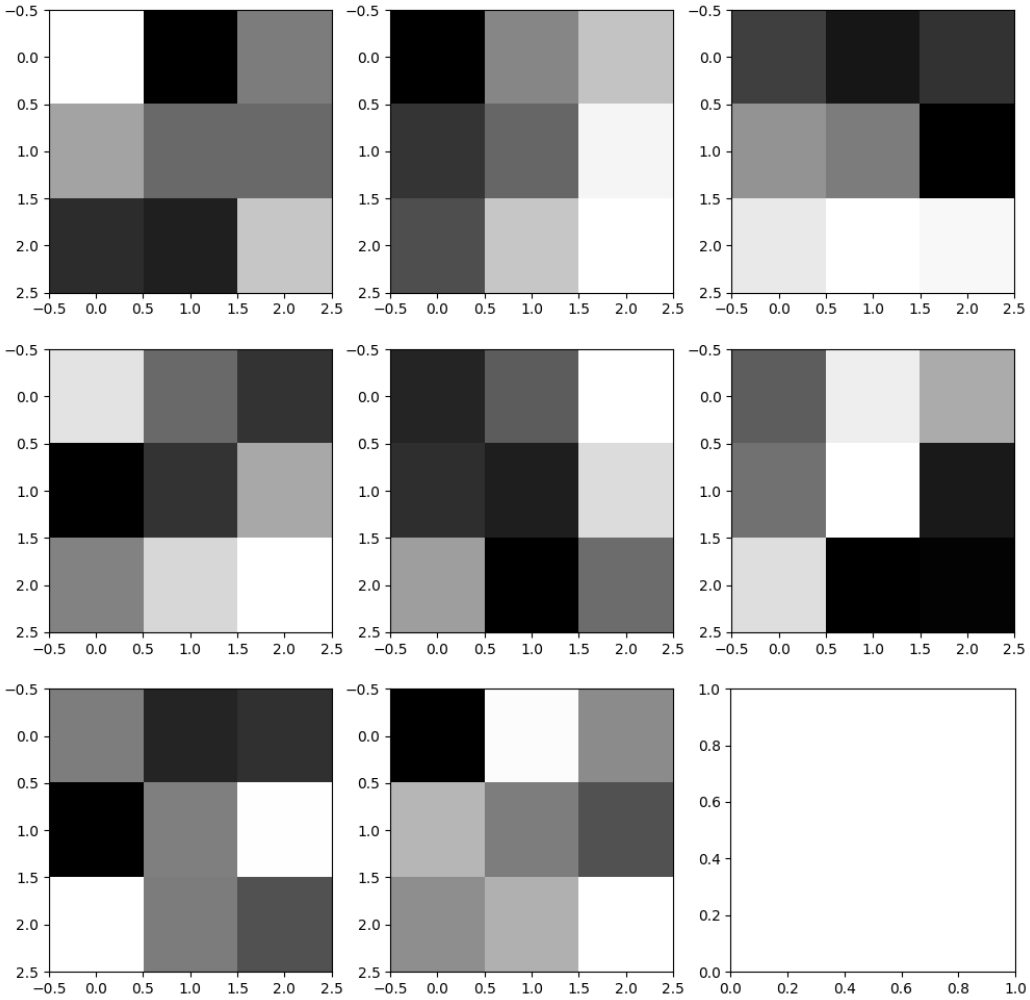
Present at least 9 examples from the test set where your classifier made a mistake. *(1 point) Present them in a 3x3 grid in your report. Discuss what you see.*



The classifier seems to make an error on inputs that are tough to distinguish. In particular, it has trouble classifying images that could potentially belong to multiple classes. For example, the 7 in the corner could also be interpreted as a 1. Some of these images could potentially be mislabeled as even human annotators would disagree on them.

Question 5
Visualize at least 9 of the learned kernels from the first layer of your network. *(1 point) Present them in a 3x3 grid in your report. Discuss what you see.*

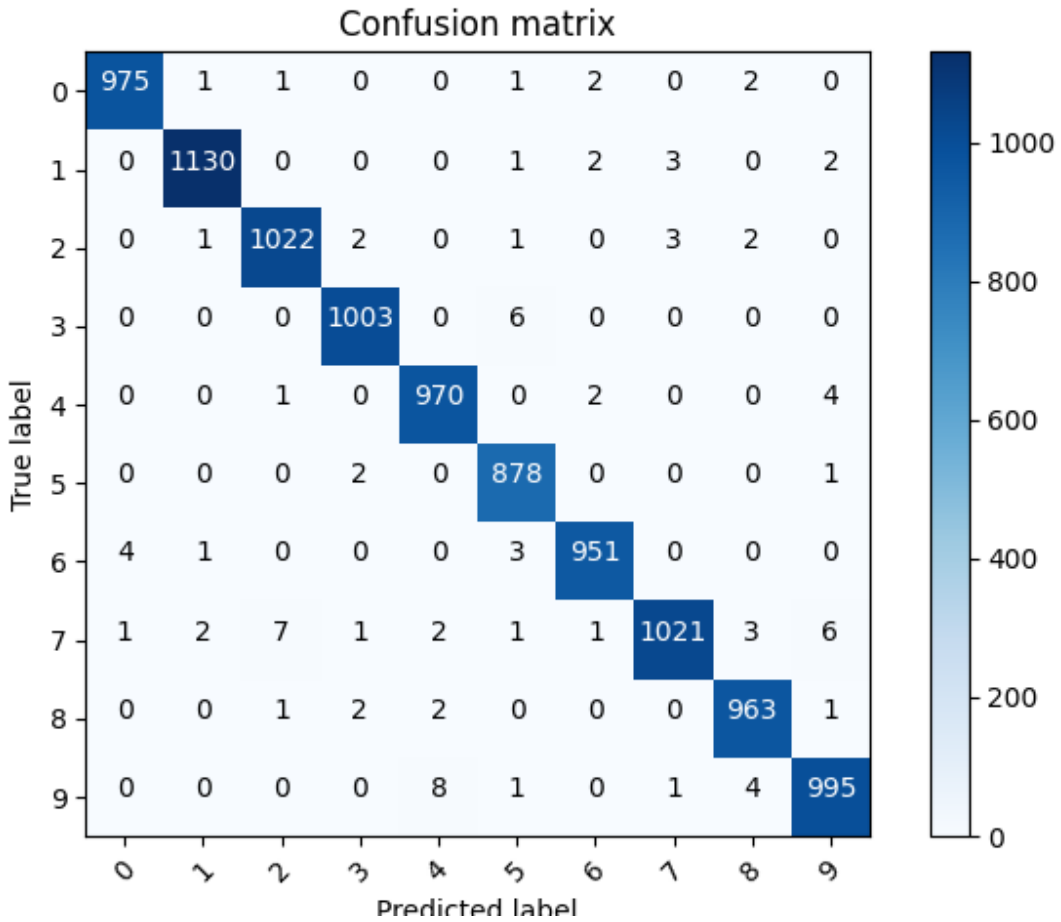
I only used 8 kernels in my first layer.



These initial kernels don't seem to show much of a noticeable pattern to them. However, we would expect them to be responsible for identifying local features of the image (ex: curves, straight lines, etc.). One thing that I noticed was that these kernels tend to have "black" spots in different parts of the filter, suggesting that they are emphasizing different portions of the 3x3 grid. In future layers, the network would be distinguishing "higher-level" features.

Question 6

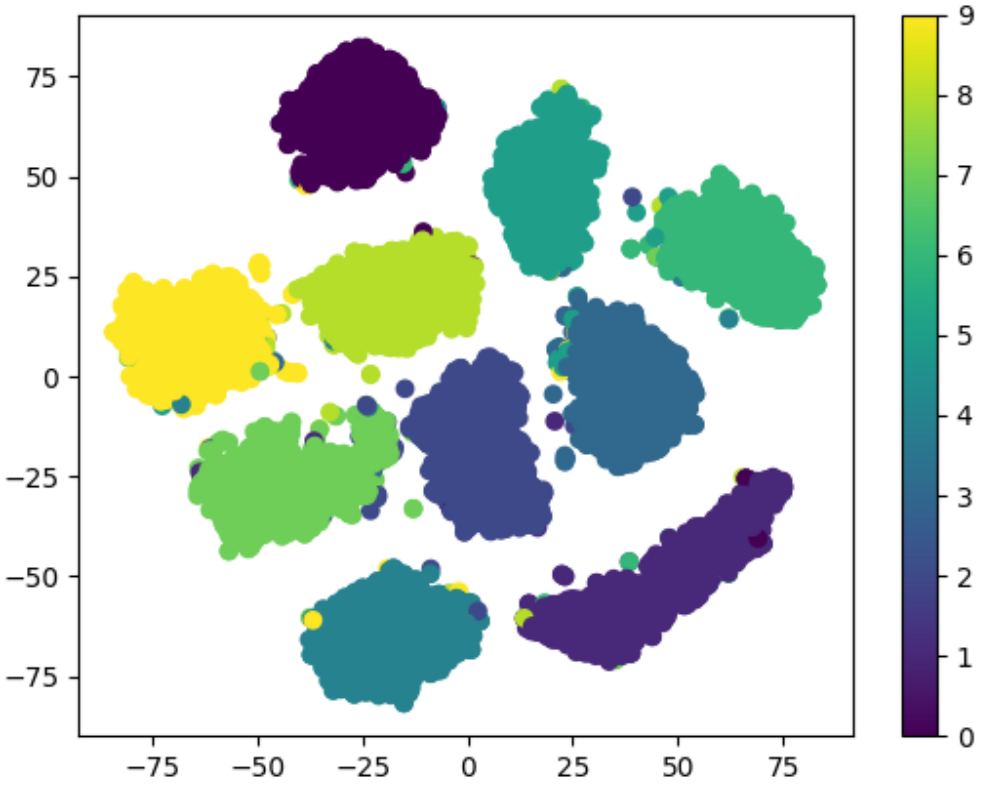
Generate a confusion matrix for the test set. *(1 point) Present it in your report. Discuss what you see.*



Most points in the test dataset are properly classified which is reflected by the large numbers on the diagonal. For misclassifications, we can see that certain pairs of numbers were often confused for each other. For example, there were 7 times that a “7” was labeled as a “2”. This makes sense because those digits look similar and could be confused under certain situations like bad handwriting.

Question 7

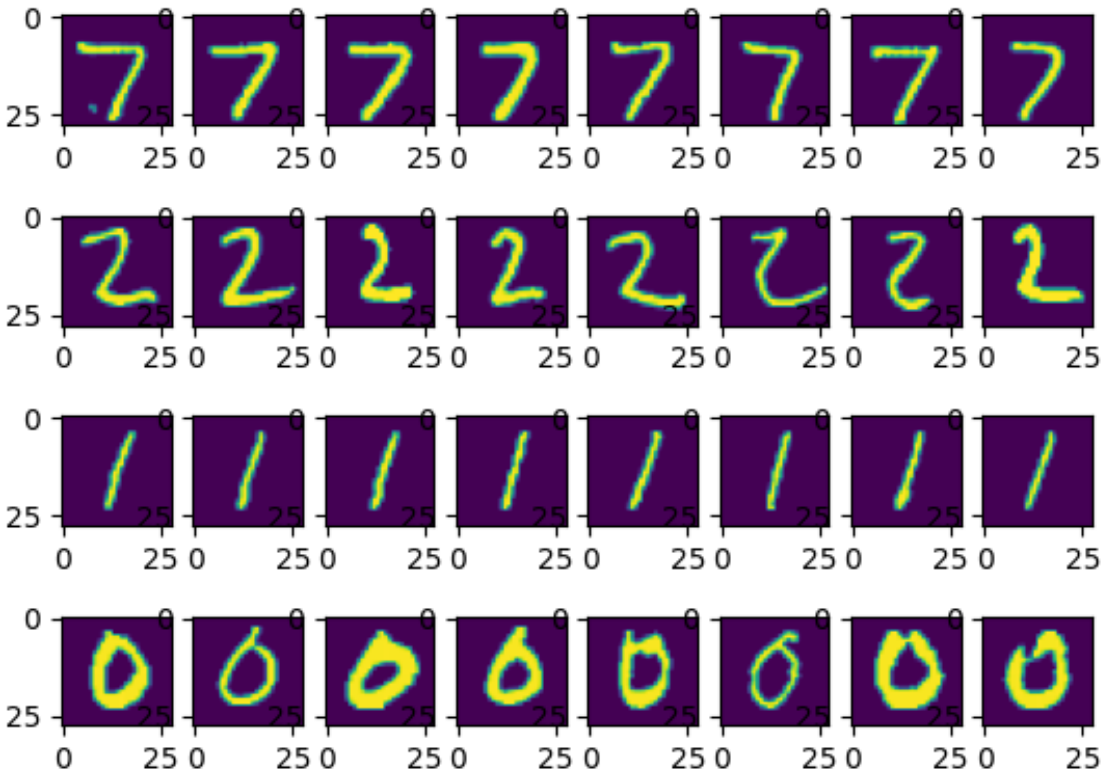
Visualize this high-dimensional embedding in 2D using tSNE (each class should have its own color). *(1 point) Discuss what you see.*



Using T-SNE, we can see clearly distinguished clusters that correspond to the labels. Points that are closer together in this embedded space are very likely to share the same label. This makes sense because this clustering was done using the features from the second to last fully connected layer. At that point in the CNN, the model is distinguishing between high-level, global features. These features are highly correlated with the label.

Question 8

Choose one image I_0 with feature vector x_0 from the test set. Find the 8 images I_1, I_2, …, I_8 in the test set whose feature vectors are closest in Euclidean distance to x_0. Repeat this process for at least 3 more choices of I_0. *(1 point) Present your results in an nx8 grid of images (where n is at least 4). Discuss what you see.*



As expected, points that were close together in the embedded space correspond to the same label. In addition, they also seem to be similar in their orientation. For example, all of the 1s in the third row are tilted slightly to the right at almost the same angle. This further supports the idea that the features from the second to last fully connected layer represent some high level, global feature of the image that corresponds to the digit shown in the picture.