Dmytro Dovhalets

**Encoded Images on Parallel Coordinates**

Parallel Coordinates are very easy to understand and use. They can be used to visualize multidimensional data. An image can be represented as a flattened vector, where the size of the vector is the number of pixels in the image. I wanted to encode images using an Autoencoder and graph the encoded representation of the images on parallel coordinates.

**Data**

I used the Mnist dataset which is a standard benchmark for hand written digit recognition. This dataset contains digits 0 through 9. Mnist has 60,000 images in the train set, and 10,000 images in the test set. The test set has 1,000 images for each digit. The original images from Mnist are 28x28 which is 784 pixels.

I did some preprocessing on the images to reduce the dimensionality. The original images which are 28x28 have a few pixels of black padding on the sides. The padding can be removed without distorting the information inside the image. I cropped the center of the image, removing the black padding and reduced the images to 22X22 which became 484 pixels. Figure 1 and Figure 2 show samples of the original images and the cropped images.



Figure 2: Cropped

Figure 1: Original

**Autoencoder**

An Autoencoder is a simple neural network where the input layer is the same size as the output layer. For the Autoencoder I used a python library called “sklearn”, it makes the construction of the neural network very straight forward. I trained the neural network on train set and tested on the test set. In the neural network I only had one hidden layer with 24 nodes in it. I experimented with different number of nodes in the hidden layer and each time I would reconstruct the images form the test set to see how well the network can encode them and decode them. Figure 3 has the reconstructed images with different number of nodes. For parallel coordinates the fewer dimensions I have the better. I choose to stay with 24, it seemed to be sufficient. Once I was happy with the neural network architecture and hyperparamets, I saved the encoded representation of the test images into a CSV file. I did not need 10,000 samples so I only took 50 to 100 samples from each class.

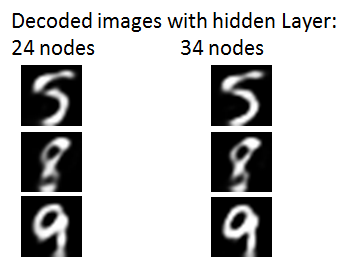


Figure 3

**Parallel Coordinates**

For parallel coordinates I used a python library “Pandas”. I have an implementation of parallel coordinates in C++ using opengl, but I choose to use this one for better quality of visualization. I plotted the 24 dimensional data on parallel coordinates different ways: using all of the classes at the same time, using only a few classes and using only one class at a time. Figure 4 has digits 0, 1, 2, and 3 plotted. Figure 5 has the digits 2, 3, 7, and 9 plotted.

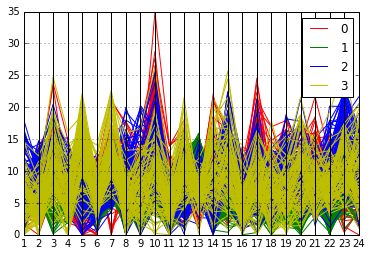
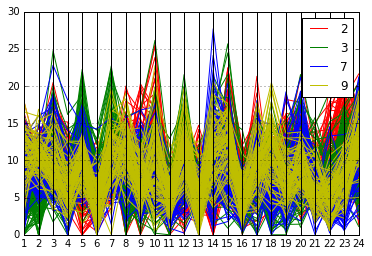


Figure 5

Figure 4

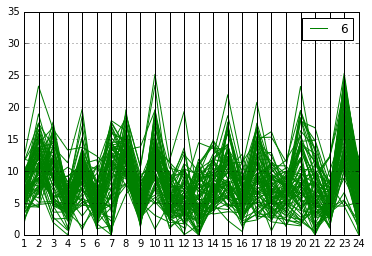
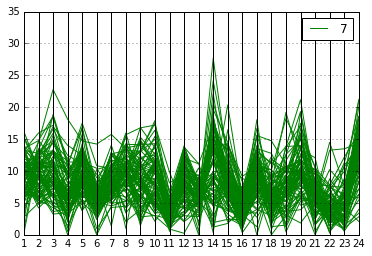
 Some patterns can be spotted where the dimensions spike up or down for certain classes while staying the same for others. Figure 6 and Figure 7 has plots of two classes separately and it easy to see that for one class the 23rd dimension always spikes up to the 24th dimension while the other class does the opposite.

Figure 7

Figure 6