Capstone 2: Text Search

Motivation:

When text documents are stored as chars, it is easy to apply dfa searching algorithms to find where substrings are in the given text. When documents are given though only as images, searching becomes a manual task, often time-consuming.

Problem Statement:

This project will attempt to convert images into a char array such that it is possible to search for given substrings in an automated way.

Data-Set:

The data is taken from this dataset on Kaggle. The data was manually extracted from NIST, lines of handwritten text and separated into 28 X 28 images of individual upper case chars.

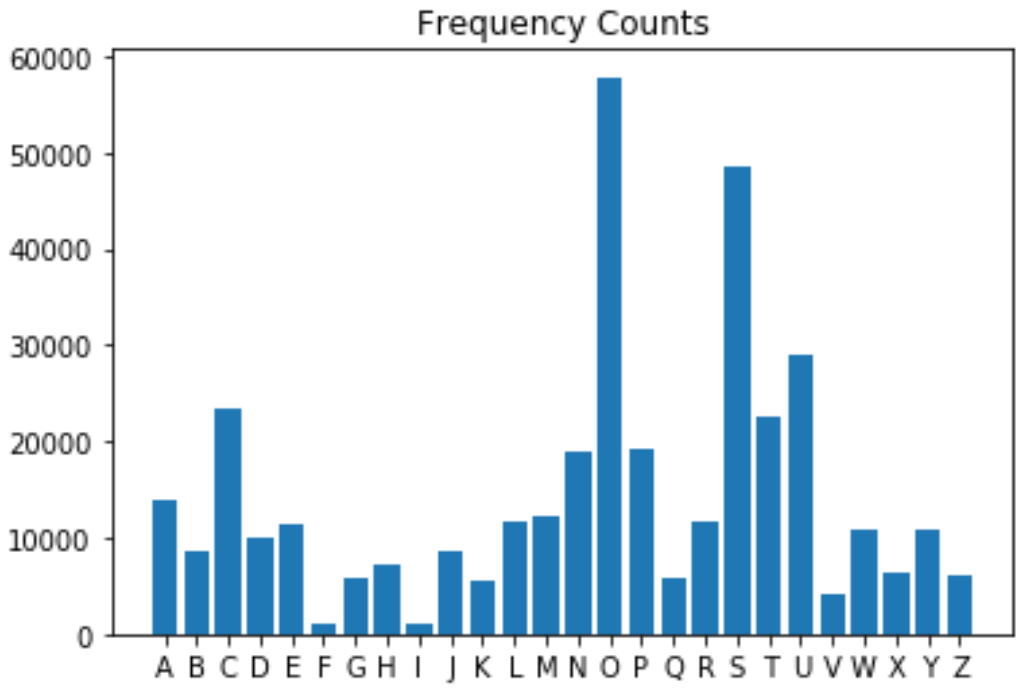
I imported the data as an array with numpy. The data was just a 372451, 785 array of floats. I then reshaped the array into 2 different arrays; one with shape 28 X 28 X 1, representing the actual 28 by 28 image, and the other a 372451 X1 array with a domain of 0 to 25 with the actual char value (ex - A, B, C…)

Exploration:

After seeing the actual shape of the data arrays, I wanted to see what one of the images actually looked like. To do this, I used the built-in Keras array_to_img function. (I could have used PIL.Image too but I was going to use Keras later anyways). I observed that the images were all black backgrounds with a white handwritten char in the middle.

I then wanted to confirm that the images of all the letters were together. Having this property isn't strictly needed, but makes the dataset a bit easier to use. I looped over all the pictures and observed that the number of times that the letter changed was only 25 times, confirming that they were all in order.

Finally, I was interesting in the distribution of the letters. I found that the letter that appeared the most number of times was 'O' and the least 'I'. I then plotted the distribution.



The number of pictures per letter isn't even close to being evenly distributed. The range between ‘O’ and ‘I’ was 57825 pictures with ‘O’ to 1120 with ‘I’, a range of 56,705 pictures.
The deviation was high at 13094 and the avg was 14325.

Because there are so many pictures with even ‘I’ having over 1000, this shouldn’t affect the model too much.

Statistical Analysis

I conducted a statistical analysis to see if the distribution of letters in the data set is comparable to the distribution in actual texts. The actual true frequency most likely differs depending on the context the text is taken from. I chose to use the count given here. http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/.

That dataset analyzed 4.5 billion chars of text in order to get their percentages. I immediately noticed that the data set I was using has a much different distribution.

I wanted to validate that this difference is significant. Intuitively, I thought it was significant because that dataset contained 4.5 billion chars and the percentages in my data set were quite different.

I set up a hypothesis with a single letter, ‘I’. The null hypothesis was that there was no significant difference. The alternate was that there was. I also represented this problem as a binomial

distribution. Success means that the letter ‘I’ appears. A failure means that it doesn’t. I used sqrt(pq/n) to estimate sigma as that is the standard error for using p as the point estimate.

Because n was really large at 4.5 billion chars, sigma was very small, almost close to 0. This happens because the binomial point estimator is consistent.
https://en.wikipedia.org/wiki/Consistent_estimator

Now the actual difference between the percentage of ‘I’ and the real percentage was large at 3 percent. Because of this, it is unlikely that if the data was taken from a data set that follows this distribution.

This shouldn’t affect the model but it is something that is important to notice about the data.

Preprocessing:

First I inverted the colors of half of the images (black background to white background). This would help the model be able to predict whether the char is black or white. I also then fitted the images to an ImageDataGenerator, a Keras class which allows for augmentation. I chose just to have some of the images be shifted vertically and horizontally up to 20 percent. I didn’t want to rotate the images because some of the chars were already slightly rotated as they were written by hand and letters can blend together if rotated too much. I then split the images into a test set and training set.

Modeling:

I used a simplistic convolutional neural network to train my model. Convolutional networks are widely used for image processing because of how they are able to extract features. There are 2 convolutional layers in my model which applies a 3 X 3 matrix. The model multiplies each pixel by the filter with matrix multiplication. By doing so, features of the chars, like diagonal lines or straight lines get noticed. Making half of the images black on white and half white on black made the model able to better predict in either case. Also, shifting the images randomly through augmentation helped with cases where the char was not centered directly in the image. I used a Relu activation function as commonly used in CNNs so that the network would not just be a linear function. After the convolution step, there was a layer that flattened them, enabling the creation of a dense layer with 26 nodes where each layer ‘voted’ which char the image was most like.For this I used a soft max activation function so that the total votes would sum to 1.

The model gave a 96 percent cross validation score showing that the model was successful.

Testing

It wasn’t easy to test the model on new data because generating the data depended on a tesseract library function that was inconsistent. It would break up the letters incorrectly, making it hard to automate a test. Also, much of the dataset I used in training the model appeared to be duplicated. I manually wrote out the Declaration of Independence to see how my model would perform. For the most part the model was accurate. A couple of the letters consistently were misclassified though like ‘N’ was confused with ‘V’.

Overall though, because the model performs well enough to act as a text searcher because although these letters were misclassified, the correct letter was in the top 3 predictions.

Additionally, the model was not confused by the fact that it was black text on a white background.

Future Enhancements

Firstly, it would be useful to expose an API based on the work in this project. It could be a narrow interface to allow a user to get back an array of letters or an array of top 3 predictions per letter for a text document. This would enable the user to use the letters to build his application whether the use case be for storing data, searching, or translating.

It also would be useful to expand the model to include lower case letters and letters from other languages. This would require a lot more data.

Finally, the model could be made more robust to account for some mistakes with letters that have diagonals and come to sharp points like ‘M’ and ‘V’.