Google StreetView Image-To-Text Converter Final Report

Kevin Huang, Xinran Pan

October 2016

1 Introduction

Street signs, billboards ads, business names all contain words and phrases formed by alphabetical characters and numerical values. Similar characters are also frequently found in challenge-response tests such as "Completely Automated Public Turing test to tell computers and Humans Apart (CAPTCHA)". Our project takes a Kaggle dataset of Google Street View images of letters and numbers (similar to the mismatched characters and numbers found in CAPTCHA), with the goal of creating an image-to-text converter using different techniques covered throughout ORIE 4741. We hope these techniques lead to useful methods for identifying and classifying simple imagines.

2 DataSet

The original Kaggle dataset contains picture images, each broken down by pixel values. The images comprise of the 26 uppercase and 26 lowercase letters of the alphabet, and 10 digits, ranging from 0 to 9. After carefully looking through the data, we realized we needed to make some initial modifications to our training dataset. Kaggle provided a few datasets to choose from, including a file for the actual image sizes as well as a file with resized images. We decided to use the resized, 20x20*.bmp file as it standardizes the size of each image to 400 pixels and provides features of the same size as well.

3 Feature Selection and Preprocessing

The next step was to read the pixel values from a *.bmp file by using a python library (PIL -¿ Python Imaging Library).This library gave us the handle to perform basic features with the data and obtain the pixel data. The original values each pixel took were in the form of a 3-tuple. RGB-tuples describe colored pixels but are challenging to work with because each pixel contains three values. Fortunately, grayscale pixels form black and white letters and digits that are as equally distinguishable as their colored counterparts. Therefore, we converted all images into grayscale. Now, instead of having three values at each pixel, we have one. Finally, we separated the data into a test and training set to estimate the out-of-sample errors. The test set contains 1228 images (20% of the entire dataset) and the training set contains 5055 images (80% of the entire dataset). We then wrote the cleaned-up data into a *.csv file for easy data manipulations and imported both training and tests into Julia. For each of these images, we also created a dictionary containing the average pixel value of all the images to compare to, a dictionary containing images that have been transformed with edge detection, and inverted images to compare to.

4 Julia DataFrames

We read both *.csv files as data frame arrays in Julia. Each array has a ":File" with the corresponding file, ":Value" and ":1 -¿ :400" representing the pixel data. In the ":Value" column of the training

set, each uppercase/lowercase letter and digit represents what the image's letter or digit is. In our separated training set, there are a total of 5055 images and 400 features for each 20x20 pixel image. Each of the features contains a value ranging from 0 to 255 where zero is black, 255 is white, and the values between them are a shade of grey.

5 Messy Data

A quick glance at the cvs does not provide information as to whether the file contains messy data (missing or corrupted data). Every entry to each feature contains a pixel value. A quick count of values less than 0 or greater than 255 shows that all values are recorded and indeed between 0 and 255. From observing the dataset, there does not seem to be any corrupt data either.

In addition to observing the images' pixel values, we also observed the images themselves. From the sample of characters we looked at, we noticed some were illegible. Examples of illegible images include blurry, off-centered, and/or cropped characters, or images containing multiple letters/numbers. We acknowledge taking a sample from the 5055 images is not an efficient or fool-proof method for finding illegible characters but it is the simplest approach. Overall, from the observed sample, there does not seem to exist many obscure images.

6 Dataset Visualization

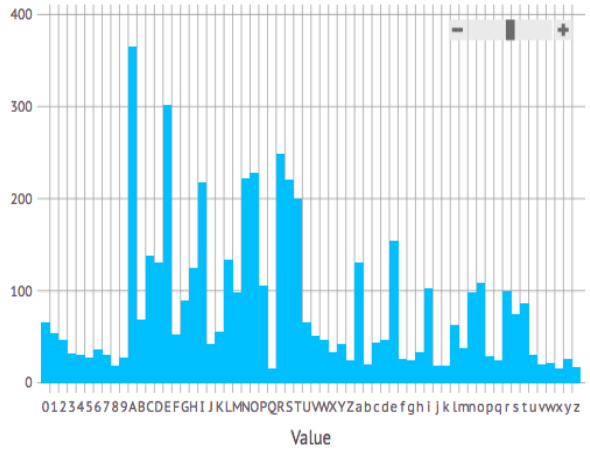
To get a better sense of the distribution of letters and numbers, we created a histogram of all 52 upper and lower case letters and 0 to 9 digits. This shows us which values we may have more or less of, and if we have enough data to properly classify each of the values.

It turns out there is a large variance between the number of images we get for each character. We are not exactly sure how this may affect the results of any model we choose later on. It seems

	File	Value	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20
1	100	Z	222	222	222	222	222	221	220	220	219	220	117	105	161	117	145	218	218	218	217	217
2	1015	A	109	107	108	104	108	108	105	106	106	107	105	108	108	103	106	107	106	106	106	108
3	1019	l	24	27	24	21	22	22	22	23	22	21	21	22	20	19	22	22	19	23	23	23
4	1027	o	66	70	63	66	63	66	67	66	66	66	62	61	64	64	64	68	70	69	67	67
5	1032	r	157	158	158	154	154	162	157	152	159	157	153	158	157	156	155	155	152	154	157	156
6	1035	V	10	8	9	8	13	8	8	12	10	10	8	10	10	7	8	9	12	10	13	16
7	104	5	63	64	67	59	60	66	69	71	73	71	68	66	67	67	67	62	63	67	66	66
8	1046	d	120	125	125	123	114	130	134	126	113	133	136	145	157	171	180	174	174	178	177	179
9	1048	4	154	174	172	171	173	174	174	173	173	171	174	178	174	174	175	174	166	168	171	152
10	1056	4	150	150	150	151	151	150	149	152	151	152	153	151	150	150	150	151	152	152	151	150
11	1059	s	114	76	84	82	82	80	80	80	81	82	83	86	83	82	94	85	77	84	79	86
12	1060	r	81	97	95	94	93	94	95	96	96	96	95	95	95	96	98	100	100	96	96	79
13	1065	A	33	34	33	36	37	36	38	36	37	32	31	34	37	31	35	31	29	33	30	28
14	1066	O	174	174	173	175	173	170	173	172	172	172	175	172	173	174	172	171	172	169	173	172
15	1070	E	64	66	62	68	64	66	64	64	65	65	66	61	64	63	65	65	65	64	64	63
16	1077	H	[illegible]	133	132	132	133	134	132	133	134	134	133	133	132	132	131	131	128	130	130	128
17	1083	A	104	102	105	103	100	103	100	101	106	107	112	118	119	125	126	127	127	126	125	125

Figure 1: "DataFrame loaded into Julia"

the characters with fewer number of images may be much harder to classify compared to characters with many number of images.



7 Simple Regression

For our first, initial method, we ran a simple linear regression using all of the pixels as individual separate features. From our cleaned-up dataset, our X matrix has the dimensions 5055 x 400, representing the 400 pixel features, and our y vector has the dimensions 5055 x 1, representing the 5055 letters and digits. We mapped every

characters/digits to a numerical value (its ASCII value) to give every element in the y vector a numerical representation. We did this using Julia's character-to-integer conversion. However, these initial results were not very promising, even if tested with in-sample data (which should give artificially high accuracy).The regression indicated that our model only predicted 88/5055 characters correctly, with a 1.6% accuracy, which is almost the same as a random guess.

In conclusion, using a simple linear regression does not make much sense for our model. Each image is different, and there is not necessarily a linear correlation between each character. Due to the way we mapped each character to a numerical representation, we may have given the datapoints new characteristics.

Though the predictions are inaccurate, this method is still useful for acting as our baseline system. The results from the regression model provides a reference point from which we can drastically improve upon to obtain character-to-text translations with a much higher percentage of accuracy.

8 K-means Clustering

For our second approach, we used the method of k-means clustering. The goal is to distinguish between the 62 different characters and partition the 5055 characters from our training set into one of 62 clusters. To do this, we used python's Scikit-Learn Standard Library's K-Means function to minimize the sum of the distances between centroids and the image pixel values. The algorithm starts with an initial set of 62 means and then alternates between two steps. The first step, the assignment step, adds each data point to the cluster which minimizes the Euclidean distance between the point and mean (the least sum of squares for that cluster). The second step, the update step, finds the new mean of the cluster as the centroid of all points in that cluster. This centroid is the updated mean for the next iteration until the assignments no longer change. By using this algorithm, we can predict which cluster each image is grouped into, and identify and classify which shape it falls under. Ideally, this method should result in 62 clusters, each with a high number of its character's numerical representation.

After we run the clustering algorithm, we also have to remap the labels that we obtained from the clustering algorithm to the ASCII values of our data. We remapped the labels to the ASCII values with the highest occurring ASCII value for each label. We predicted the images in the test set by comparing the distances to each of the cluster centers that we obtained from our k-means clustering. This netted in various different results, as Python's k-means clustering didn't seem to converge with even a million iterations, ranging from $2 - 5\%$ accuracy. However, we were able to provide a guess of where to start the centers instead of initializing them. Our guess was taking pixel values of the average of all the images of the same character. In this way, we provided a good place on where k-means to start, and this result converged to 5.4% accuracy. We believe that Python's Scikit-Learn K-Means algorithm oscillates based on the random vector that is used to generate it.

9 Feature Selection and K-Means

9.1 Binarization $\Rightarrow$ K-Means Clustering

We tried to improve on our k-means algorithm by making the pixel values as different as possible so that it captures more of the essence of the image. For every pixel that was under 127(half of the maximum value), we would set it to 0, and for every pixel above 127, we would set it to 255. We tried to do this to see whether or not by binarizing the picture if it would help increase our accuracy. After binarizing all of our images, we would perform

K-means clustering on it. As it turns out, this does not increase our accuracy at all, obtaining around a 3.4% accuracy. We believe that because each picture had pretty large differences between pixels, each different pixel would result in a much greater difference than a more smoothed image.

9.2 Principal Component Analysis ⇒ K-Means Clustering

The next method we used was principal component analysis (PCA). PCA takes orthogonal transformations (linear transformations on the inner product space) to change the correlated values of our pixel values into uncorrelated variables. Our goal for using PCA is to reduce dimensionalities to reduce noise in the data. Since some of the pixels in each image contain values which are not useful for classifying characters (ie. the pixel values corresponding to an image's background), it may helpful to get rid of noise not pertaining to the classifying problem we want to solve. In order to get a sense of how many directions we wanted out PCA to project down to, we took a look at the eigenvalues of the covariance matrix of all of all images. After examining our eigenvalues, we saw that it dropped off around 25 eigenvalues. We assumed that this was the point where the more important informations was clustered together. We ran k-means clustering on this dataset and obtained a value of 5.5% accuracy.

10 Spectral Clustering

The final technique we used was spectral clustering. This technique also uses dimensionality reduction to lower the number of dimensions in our dataset of 400 pixel features before clustering the reduced dimensions, using k-means. We defined our graph by creating a adjacency matrix based on the distances between each image. We took the 10 closest neighbors and created edges for these neighbors. These edges represent the similarity between the two images. Our motivation for trying to use spectral clustering was that we wanted to see if our datapoints would exist in a subspace that would be better described by connectivity rather than distances to clusters. After our spectral clustering was run, we only obtained 0.8% accuracy. After obtaining these results, we realized that something was very off. We checked the distance function to each of the other points in the graph and realized that the distance function is NOT a good indication of similarity. In many cases, two images of the same character would not be in the lowest 20 distances between the average images. Because these images are never connected in the spectral clustering connectivity, it is reasonable for it to fail pretty miserably.

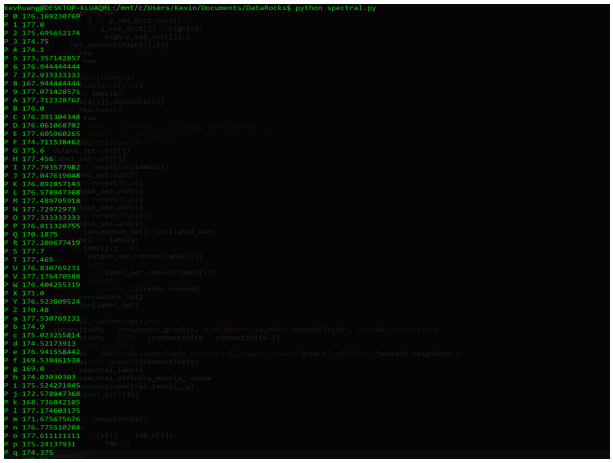


Figure 2: "DataFrame loaded into Julia"

11 Consideraton:Convolutional Neural Networks

Because of time constraints, we could not implement our fifth idea: using convolutional neural networks to classify images. A neural network is made out of layers of artificial nodes called "neurons" that take the pixel values as inputs and transforms them into value outputs. These outputs have different weights depending on the importance of each pixel and its relationship to the character inside the image. To further break down the process, we would use convolution. Convolution prevents

us from having to feed the entire 400 pixel values into a neural network. Instead, we break the 20x20 pixel image down into small "tiles," each containing the same dimension. The tiles would output a reduced number of values using the same neural network as before and narrow down the pixels to decide if the image matches with its corresponding character. This transformation is non-linear and can remodel our original pixel data into few values the images can accurately map to. This is the current industry standard technique for classifying images, and is expected to run well. This is due to the neural networks being able to recognize patterns within the image, and not be constrained to the linearity of the data.

For a clear, easy-to-understand, and comprehensive explanation of convolutional neural networks, read here: 'https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721.nnr4f7f86'

12 Conclusion

Our goal was to create an accurate image-to-text converter focusing on multiple clustering and classifying techniques. Unfortunately, our methods, did not produce the desired, high percentage of accuracy for correctly predicting each image's corresponding character text. This was due to a number of reasons. One significant reason was because of the dataset images themselves. We did not anticipate the challenges of working with such convoluted images. For each character or letter, the variation in images were drastic. Even after converting to grayscale, we had darker characters against lighter backgrounds and vice versa, which made it difficult to compare the pixel values with each other. Many of these images were rotated by many degrees, some as many as 90 degrees. Thus, binarization did not help increase accuracy. Unfortunately, K-Means did not as well. There are also numerous reasons for why this might have occurred. The dataset was too varied to have an algorithm that tries to classify fixed points in space to do well. We can demonstrate this by comparing the distances between the images, and realizing that the euclidean distance does not indicate anything between the images. Our dataset had too many variances in the data for clustering to mean anything.

Without further exploration and implementation of convolutional neural networks, we would not recommend our methods for use in production. Ideally, we would need more time to refine these techniques and algorithms to obtain more reliable results with high accuracies. We are confident, however, that the use of convolutional neural networks would help us achieve our desired goal, something we would fully implement if we were not limited by time constraints.