Bootstrapping Optical Character Recognition

Ralph "Blake" Venté

April 2020

1 Description

Optical Character Recognition (OCR) has a wide scope of applications. What good is an autonomous vehicle that can't read the speed limit? What can we gain from fluently transcribing hand-written notes? Could we reach the point where the world's corpus of old mathematical texts are transliterated into rich, modern text documents?

In this research, I investigate the feasibility of bootstrapping methods for Optical Character Recognition. Given just a single ground truth image for every character, I investigate the effectiveness of data augmentation in the creation of an artificial dataset. Then, I investigate the performance of different machine learning algorithms on the ground truth, augmented, and a real-world sample.

1.1 Objectives

The main objective is to unify the past research to the modern day, relating all past research under one standard group of metrics. I will explore the differences between each of 3 models: kNN, Neural Networks, and SVM in each of 3 contexts: scanned documents, handwritten digits, and screen captures. Then, proceeding with the best model, I hope to implement a document segmentation system similar to the proprietary (guidelines, but no code was provided for this system) one in Suzuki's INFTY, with mine being open source and written from scratch. At the end of the day, this task will combine supervised machine learning (Optical Character Recognition), and unsupervised learning (clustering algorithms for Document Segmentation).

2 Accomplishments

1. Create synthetic ground truth dataset of 52 examples

(a) Used the `convert` utility provided by the ImageMagick package to transform vector graphics into raster images.

(b) For normalization, centered each character in a square 32×32 image frame.

(c) Implemented augmentations by scale, brightness, contrast, hue, and gaussian noise.

(d) Consolidated examples into a dataset of 1,040 examples.

2. Trained all 3 of k Nearest Neighbors, Support Vector Machine, and Convolutional Neural Network and documented results.

3. Built and iterated upon my neural network classifier acheiving 93.07 percent validation accuracy at $20 \cdot 52$ examples and 97.03 percent validation accuracy at $40 \cdot 52$ examples.

4. Created a naïve implementation of an Optical Character Recognition engine by using hierarchical clustering and a sliding window to classify characters.

5. Originally intended on exploring other forms of hierarchical clustering, however my scope of time was not sufficient for an implementation.

3 Overview

image_generation.py In this script, I generate a small alphabet of examples in LaTeX and render those examples into `pdf` files.

Then, apply normalization. I center each image (by center-of-mass) and normalize the dimensions of the image.

machine_learning.py I create and test 3 models in this file, generating artificial augmentations like those seen in Figure 1. I started with the examples credit the Tensorflow authors. I also use a sample implementeation of k nearest neighbors, and I use the support vector classifier from Python's Scikit. It was a time-consuming challenge to incorporate all of these models with the same data set, but I learned many lessons along the way.

hierarchical_segmentation.py In this file, I make a sliding window algorithm that iterates over portions of the image by their k means defined line-clusters. I extended a code sample from `https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/`.

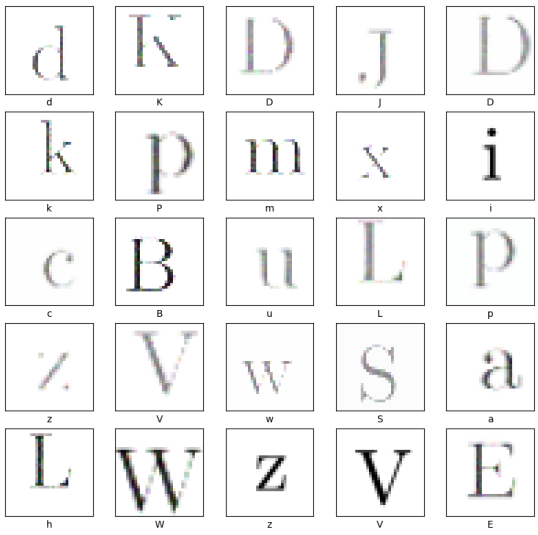


Figure 1: A subsample of the randomly generated examples created by my data augmentation scripts. The neural network classified more than 93 percent of examples in this set correctly. As a qualitiative note, the network appears to be resilient to broken strokes in the figure of the leftmost lowercase "z" above. This is a promising start for emulating the defects in scanned documents, but as my investigation shows, a larger network and more data are certainly in store for adapting these findings to scanned documents.

4 Training

I use the implementations of SVM from Python's Scikit library and a default implementation of k-NN. I varied the parameter k over 3, 5, and 7, getting the best validation set accuracy with $k = 5$. I got the best validation accuracy on the neural network after expanding the original 1 fully-connected layer to 3. The network is modeled after arcitectues in LeCun et al. 1995. The complete architecture is found in Figure 2.

5 Results

Unsurprisingly the convolutional neural network wins out on this task. Performance for k-Nearest Neighbors degraded when increasing k above 5. In the next steps, the neural network my model of choice.

Layer	Output
Input Convolutional Layer	$30 \times 30 \times 32$
Max Pooling 2×2	$15 \times 15 \times 32$
Convolutional Layer	$13 \times 13 \times 64$
Max Pooling 2×2	$6 \times 6 \times 64$
Convolutional Layer	$4 \times 4 \times 32$
Flatten	$\emptyset, 1024$
Dense	$\emptyset, 64$
Dense	$\emptyset, 64$
Dense	$\emptyset, 64$

Figure 2: Convolutional Neural Network Architecture. Adding the last two fully connected layers yielded a 3 percentage point improvement. the $\emptyset$ symbol indicates a fully-connected layer.

	kNN	CNN	SVM
MNIST Digits	0.48	.99	.92
baseline	1.00	1.00	1.00
augmented	0.36	0.93	.34

Figure 3: Results of different ML methods on OCR

6 Inference Over Time

There are a number of mechanisms to transfer a classifier like the one I have made into a sequence model that reasons over timesteps, but perhaps the most simple is the sliding window technique, but with a few notable changes for efficiency.

6.1 Methodology

First and foremost, I use Canny Edge detection and a Hough Line detection algorithms from OpenCV, showing the results in Figure 4. Inspired by the research in Saha et al. 2010, the Hough tranformation was an excellent selection for a base-line.

These allow my model to separate the image by textline, but the results aren't perfect as many overlapping lines are detected. For this reason, I used k Means to bring the overlapping sample lines down into 1 representative line that corresponds to a single line of text. For the parameter k, we need an accurate number of lines on the page. In my sample image, there are 25 lines per page,

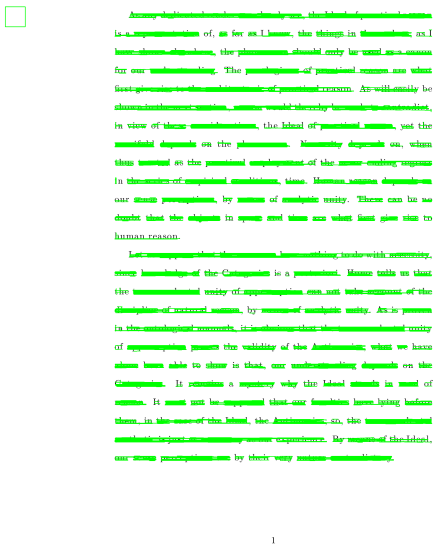


Figure 4: (a) Hough lines provide a very promising baseline for text segmentation. The Hough tranformation not only captures information about the vertical separation of lines, but also enables the discovery of the horizontal bounds of the text as well.

but this is a point for improvement.

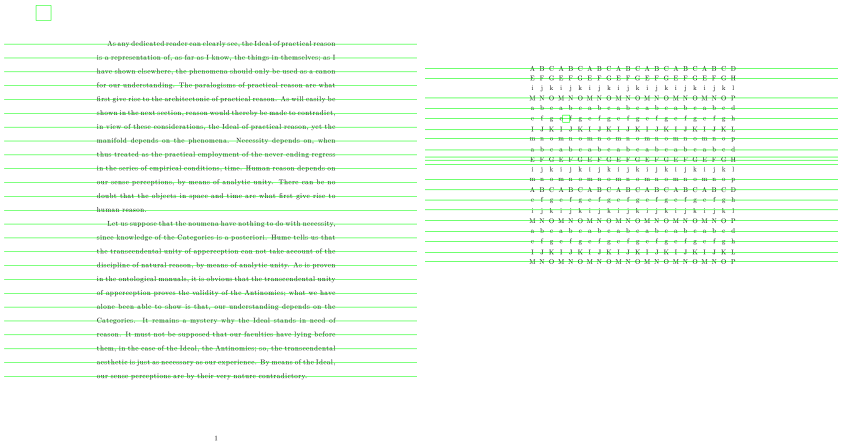
After I have these metrics, I slide a 32×32 window over the image, leading me to my next problem. I had already modified the text document to give my model enough space between characters, but there were still cases where a window would contain more than one character. For this reason, I considered using a region-finding algorithms, also in OpenCV to break the window into contigous regions, however this introduces another issues. Characters such as the lowercase `i` and `j` are composed of two contigous regions, not one. Any diacritics above the character would be separated away.

One solution that was too costly to explore in detail would be to train a neural network with Long-Short Term memory and an attention mechanism: the attention mechanism is a learned parameter of the network that would allow it to "focus" on portions of the input space one at a time. I didn't explore that because it would mean I'd have to dramatically change how I created synthetic

data not to mention the much larger network size I would need to train.

To give a proof-of-concept, I made a contrived sample document that is overly idealistic. In natural text, it is standard for the letter-forms to be very close together. But given my system relies on there being one character in each window, my unrealistic sample document houses each character in a square on a matrix. This is the only way I could reliably get predictions from my neural network because even though there was invariance hard-wired for translation and scale, no such measures were taken for the segments of other characters.

6.2 Results



(a) (b)

Figure 5: (a) Performing k means on hough lines provides a good method for breaking a page up into component lines. As long as those lines have sufficiently similar weight. In this case, the shorter lines did not impair the behavior. (b) In this case, the relative weight of the lines was very different, so even though the value of k was set correctly, one textline has two extraneous prediction paths, and two textlines are missing their own.

On some samples, k means gave good results. This was true in the case in Figure 5 (a), where there was only one outlier to keep the text lines from having similar amounts of black and white pixels. When this balance was not present, some lines of text got extra paths and some lines of text were missing their paths. To conclude k Means is very limited in this scope. Even if the user is able to provide the correct number of lines, k means (with its random initialization)

is not guaranteed to find the correct matches for lines. As an alternative for future exploration would be gaussian mixture models

7 Segmentation and Hierarchical Clustering

Initially, iterating through the windows yielded very poor results for lowercase characters and spotty (but much better) performance on upper case characters. I must conclude that the initial set of 1040 samples did not generalize well over into practice. This is surprising to me because of all of the variation in the test set that would otherwise indicate that the neural network should be invariant to features like translation and occlusion. I explain how I overcame this obsticle later.

What is expecially peculiar is that in my experimentation, the size of the window had an influence on performance. I have witnessed the best performance on a window size of 38×38. This makes some sense because when I was augmenting translation and scale, I chose a padding of 6 pixels around the image. Perhaps I would get better results if I increased the number of random examples beyond 20. I tried generating 40 random examples, but I quickly ran into a computational burden: without GPU acceleration I could not iterate on any models with this dataset, however, I did see a big improvement if validation accuracy when training the same network on this dataset, leading to dramatically better predictions when I apply it to the segmented synthetic image.

Ultimately, the initial $20 \cdot 52$ examples must be considered insufficient. I got the best results with double the number of examples. Other research also suggests the use of salt-and-pepper noise instead of purely random noise as I have used. I have uploaded a video demo to YouTube for viewing purposes containing a complete run-through of generation on my synthetic example.[1]

Clustering algorithms like k Means do not usually give information about hierarchies between the elements. However, by pairing it with k means, I was able to break apart lines in an input image and then iterate over the lines separately.

8 Conclusion

In summary, synthetic examples provide a promising start for optical character recognition. However, additional work needs to be payed to making the synthetic data closer to representative of true examples before effort can be put into training models on this data.

[1] `https://youtu.be/OD2Sk3rxXRc`

References

[LeC+95] Yann LeCun et al. “Comparison of learning algorithms for handwritten digit recognition”. In: *International conference on artificial neural networks*. Vol. 60. Perth, Australia. 1995, pp. 53–60.

[Sah+10] Satadal Saha et al. “A Hough transform based technique for text segmentation”. In: *arXiv preprint arXiv:1002.4048* (2010).