Rapid Natural Scene Text Segmentation

Ben Newhouse, Stanford University

December 10, 2009

1 Abstract

A new algorithm was developed to segment text from an image by classifying images according to the gradient composition of each edge. This algorithm is translation, scale and rotation invariant, and touches each pixel approximately 70 time less than a similar sliding window based classification scheme. The gradient angles were bucketed into ten buckets, which were then used as features to an SVM trained with a variety of kernels. A kernel of polynomial degree 10 achieved a weighted success rate of approximately 76.9%.

2 Introduction

In recent months, great attention has been given to the idea of "Augmenting Reality" with additional data. At Yelp, we developed one of the first such applications that used the phone's magnetometer, accelerometer, and GPS sensors to attempt to label venues located in front of the iPhone's camera. While this application was generally well received, its usefulness is heavily constrained by the inaccuracies of the GPS (+/-40m), and the vulnerability of the magnetometer to passing magnetic fields.

The next generation of augmented reality devices will inevitably rely on computer vision techniques. This has already begun with the release of Google Goggles in early December 2009. These computer vision techniques can be largely divided up into two branches of algorithms: algorithms for generic pattern recognition, and algorithms for text recognition. The first of these two patterns is relatively mature, as demonstrated by such algorithms as SIFT and SURF. The second of these two problems is surprisingly complex. OCR on document has historically been one of the true success stories in machine learning. Due to lighting and geometric transformations in the natural world, OCR on natural scenes is nowhere as near as well developed.

While, it is true that images can be offloaded to a server to be processed, we believe that this is less ideal than an algorithm that can be done on a mobile device such as an iPhone in real-time.

3 Process

In order to design an anlgorithm to be run on a mobile device, the mobile device must be characterized. This proved extremely difficult with the iPhone as no API for accessing the live video stream has ever been disclosed publicly by Apple or anyone else on the internet. After spending a significant amount of time looking for symbols in low level private C libraries, an API was discovered that provided the programmer with access to buffers of 512x384, 640x480, 2048x1536 with the pixel data encoded as YCbCr.

Next, a basic 784 input, 10 output neural net with two intermediate layers was trained using backpropagation on MINST data (using the FANN open source library), to an error rate of approximately 1%. Using this neural net, an iphone application was developed to recognize handwritten digits in the top-left 28x28 pixel block of the luminance portion of the image naively thresholded at a fixed threshold.

Further algorithms (such as adaptive thresholding and convolution of first order Haar wavelets) were compared on the iPhone and a Core 2 Duo MacBook running OS X 10.6. It was found that the iPhone runs

at approximately 5% the speed of the Core 2 Duo and in particular is very slow in performing floating point computations.

Generalizing the previous implemented OCR algorithm to the entire image would require touching each pixel approximately 748 times (for 196,608 pixels in a 512x384 image). This is approximately 147 million operations just on loads/stores. Incorporating the time taken in cache misses for such an algorithm would further slow down its execution. If an algorithm is to be used in real-time classification, it cannot be based on an algorithm that uses sliding windows for analysis.

The algorithm we developed is called Gradient Edge Trace Segmentation or GETS and consists of the following steps:

1. Adaptive threshold the image according to the following formula:

$$\text{pixel}[x,y] = \text{pixel}[x,y] \quad (1)$$

$$+ \quad (128 - \frac{\sum_{x-n}^{x+n}\sum_{y-n}^{x+n}\text{pixel}[x,y]}{(n+1)^2}) \quad (2)$$

2. Convolve horizontal and vertical Haar wavelets and take the tangent between the sums if the sum of the sums is greater than a set threshold

$$\alpha = \text{pixel}[x,y] - \text{pixel}[x,y+1] \quad (3)$$

$$\beta = \text{pixel}[x,y] - \text{pixel}[x+1,y] \quad (4)$$

$$\text{pixel}[x,y] = \tan\frac{\alpha}{\beta} \text{ if } \alpha+\beta > \gamma \quad (5)$$

$$\text{pixel}[x,y] = 0 \text{ if } \alpha+\beta \leq \gamma \quad (6)$$

3. Flood fill the remaining edges according to the following algorithm:

```
for pixel in image,
  if pixel has not been touched
    and pixel is an edge:
    Initialize a new component
    Create a stack with current pixel
    While the stack isn't empty:
      Pop a pixel from the stack
      Add untouched edge neigbor pixels
```

4. Take the set of angles in component, sort them into m equally sized buckets. Use each bucket as a feature to an SVM which classifies the edge as being either text or non-text.

The motivating hypothesis behind this algorithm that text has very complicated edges in that they have many angles and the transitions from positive to negative angles more often than generic shapes. If the frequency of an angle in an edge is scaled according to the length of the edge then this algorithm becomes scale invariant. If we rotate the array of edges to start with, say, the longest angle (ie. the longest stretch of pixels with the same angle value), then this algorithm becomes rotation invariant. Finally, because every edge in the image is traced, it is translation invariant.

This algorithm was developed initially in python as abstractions could be made in order to keep track of how many times a given pixel is touched for a given algorithm under development. Additionally, developing in Python allowed for faster iteration once we had characterized the abilities of the iPhone.

In order to test this algorithm, 53 assorted photos were taken of a dorm room comprising assorted books, documents, walls, clothes, etc. Next, these photos were individually processed by drawing magenta lines through all text in the image. Next, training and test data was compiled by running our algorithm and classifying each edge if it intersected part of a magenta line drawn on the classified image.

4 Results

Running our algorithm on the 53 classified images to obtain samples on which to train, our algorithm extracted 24,489 edges (edges of length less than three were ignored). Of these, 2103 were text edges and 22,786 were non-text edges. On average, each pixel was touched approximately 11 times (according the counts provided by our Python abstraction) to In order to derive the features from which to train on, the data was first analyzed in Matlab. Using Matlab, random text and non-text gradient edge traces were plotted (plots can be found in Appendix A). In these plots, it was found that angles in non-text

edges tended to take a few discrete values, whereas text edges tended to take continuously variable edge traces. To digest this phenomenon into a more discrete feature, the edge angles were sorted into bins, as if to be shown in a histogram (which can also be found in Appendix A). To ensure scale invariance, buckets were normalized by dividing each by the total number of angles/edge pixels. From inspection, it can be seen that text is highly likely to have more non-zero buckets than non-text.

Next, an SVM was used to classify each edge, and k-fold cross validation was used to evaluate the performance of each (with $k = 5$ in our case). In addition, as a baseline, edges were deterministically classified as text if they have less than or equal to n zeros, where n was found best to be 2. Linear, Gaussian, and Polynomial of Degree 10 kernels were tested in additional to our deterministic classification, resulting in in the following confusion matrices.

SVM - Linear Kernel

labeled truth	not text	text
not text	14202	506
text	8584	1597

Success: 63.5%
Weighted Success: 69.2%

SVM - Gaussian

labeled truth	not text	text
not text	17559	512
text	5227	1591

Success: 76.9%
Weighted Success: 76.4%

SVM - Poly-10

labeled truth	not text	text
not text	18193	547
text	4592	1556

Success: 79.3%
Weighted Success: 76.9%

Text if less than or equal to 2 empty bins

labeled truth	not text	text
not text	20673	965
text	2113	1138

Success: 87.6%
Weighted Success: 72.4%

From this we can see, that using an SVM is largely useful for decreasing the number of false negatives, marking text as non-text (which the deterministic approach is very vulnerable to). This is very important however, since it is better to try to classify some non-text (and fail) in an image than it is to completely fail to let the letter classification even look at an important edge. Henceforth, the SVM approach is much better than the naive approach despite the lesser overall success rate (which can also be attributed to the disproportionate sample sizes of text edges vs. non-text edges).

5 Conclusion and Future Research

Improvements could likely be made in the order in which the connected edges are traced, ensuring that angular transitions are more coherent and continuous. Furthermore as Gradient Edge Tracing is (as far as we know) a new feature, there are still questions to be answered in terms of how much information can be extracted form it. Using FFTs or other transforms it might be possible to do character recognition in addition to just binary text classification.

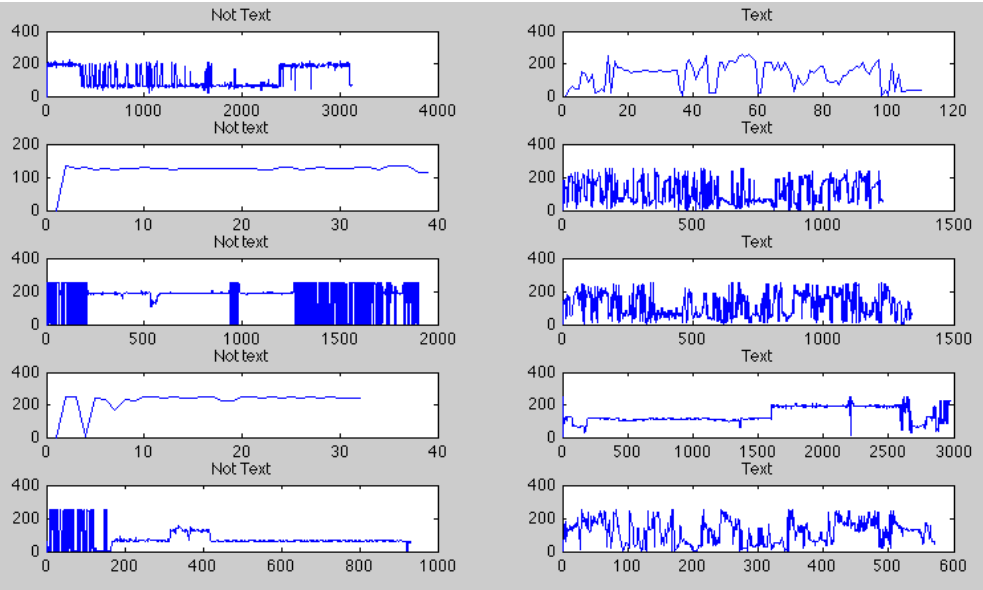
In conclusion, it was demonstrated that an SVM could be applied to "Gradient Edge Traces," to classify edges as text or non-text, with an average accuracy of 76.9%. Furthhermore, this algorithm is computationally efficient and translation, scale and rotation invariant.

6 References and Special Thanks

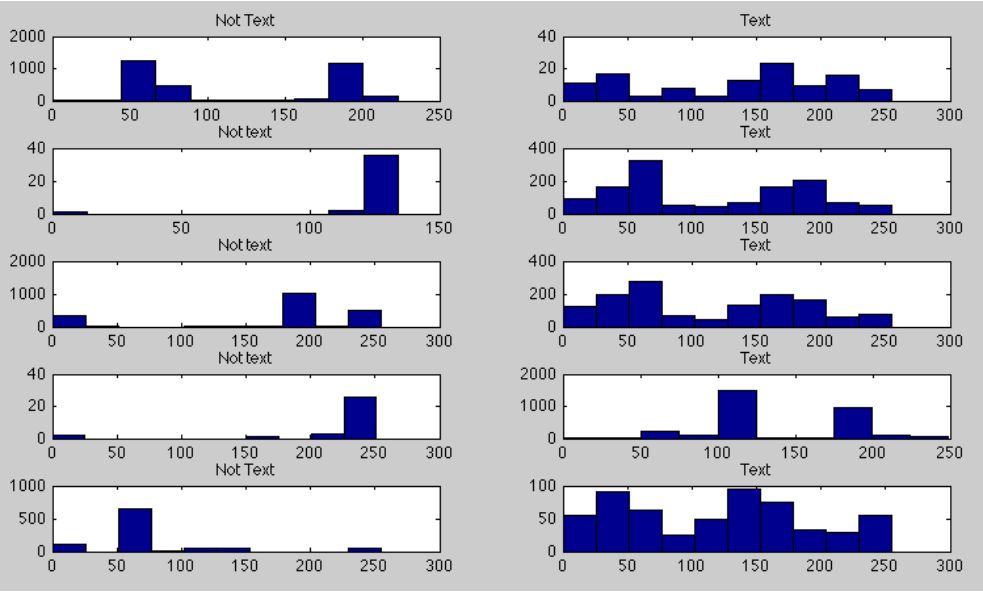
Special Thanks to Otavio Good for helping us figure out that the iPhone framebuffer is YCbCr encoded.

7 Appendix A

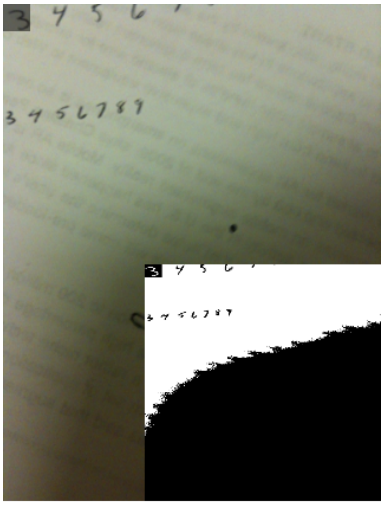
7.1 Edge Angle Sequences



7.2 Histogram of Angles Per Edge

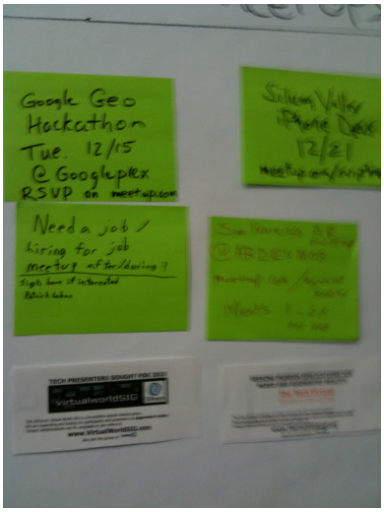


7.3 Neural Net OCR Running on the iPhone

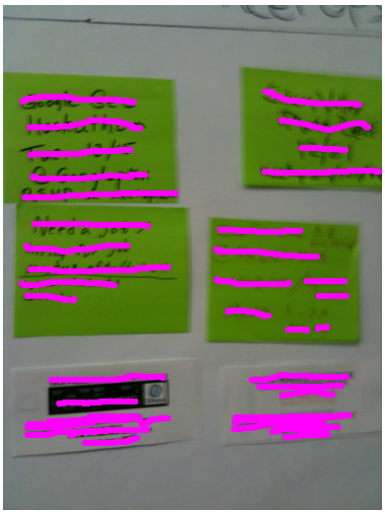


-5x0, Number: 0)0.0 1)0.0 2)0.0 3)2.0
4)0.0 5)0.0 6)0.0 7)0.0 8)0.0 9)0.0

7.4 Example Original Image



7.5 Example Training Image



7.6 Example Edge Filtered Image