UNIVERSITY OF CALIFORNIA, SAN DIEGO

Task Specific Image Text Recognition

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Nadav Ben-Haim

Committee in charge:

David J. Kriegman, Chair
Serge Belongie
Yoav Freund

2008

The thesis of Nadav Ben-Haim is approved and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2008

DEDICATION

To my grandparents Lea Kigler, Marku Kigler, Nurit Ben-Haim, and Michael Ben-Haim, who risked their lives to create better lives for their children and grandchildren. Also to my parents and siblings, who have been supportive of all my pursuits. Last but not least, to America, Israel, and all those who fight for freedom, allowing me to comfortably sit here and write this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I cannot thank Professor Serge Belongie enough for his consistent help throughout the entire 2 years I have spent doing graduate work. Also thanks to Boris Babenko and Dr. Piotr Dollar for very useful suggestions regarding the boosting framework I wrote. My graduate work started as a result of my interest in Louka Dlagnekov's Masters thesis, ”Recognizing Cars” (Dlagnekov and Belongie, 2005). Louka provided me with helpful resources related to my work. This work was partially funded by NSF Career Grant #0448615.

ABSTRACT OF THE THESIS

Task Specific Image Text Recognition

by

Nadav Ben-Haim
Master of Science in Computer Science

University of California, San Diego, 2008

David J. Kriegman, Chair

This thesis addresses the problem of reading *image text*, which we define here as a digital image of machine printed text. Images of license plates, signs, and scanned documents fall into this category, whereas images of handwriting do not. Automatically reading image text is a very well researched problem, which falls into the broader category of Optical Character Recognition (OCR). Virtually all work in this domain begins by segmenting characters from the image and proceeds with a classification stage to identify each character. This conventional approach is not best suited for task specific recognition such as reading license plates, scanned documents, or freeway signs, which can often be blurry and poor quality. In this thesis, we apply a boosting framework to the character recognition problem, which allows us to avoid character segmentation altogether. This approach allows us to read blurry, poor quality images that are difficult to segment. When there is a constrained domain, there is generally a large amount of training images available. Our approach benefits from this since it is entirely based on machine learning. We perform experiments on hand labeled datasets of low resolu-

tion license plate images and demonstrate highly encouraging results. In addition, we show that if enough domain knowledge is available, we can avoid the arduous task of hand-labeling examples by automatically synthesizing training data.

Chapter 1

Introduction

As images and video become more abundant on the internet, there is a growing desire to automatically analyze these vast amount of data. In our "post 9-11" world, it is clear that automatically analyzing surveillance footage is a high priority for government agencies and security companies. One way to analyze images and video is to automatically read the image text contained in them (See Figure 1.1 for examples of image text). This has proven to be valuable in vehicle surveillance with technologies such as License Plate Recognition which are in operation globally. Furthermore, reading image text such as signs and product labels on the internet can translate into profits via targeted advertising for search companies. With variability in fonts and image quality, accurately reading image text is an open research problem. Even in the constrained domain of License Plate Recognition, companies report accuracy levels of 95%, which implies there is room for improvement (Neubauer and Tyan, 2003).

1.1 Problem Statement

There is a consensus in most of the literature that image text recognition is divided into two steps - character segmentation, and character classification (Naito et al., 2000; Yu and Kim, 2000; Chen and Yuille, 2004). Character segmentation can fail un-

Figure 1.1: Examples of image text. From license plates, to signs, to scanned documents, and in varying degrees of quality and resolution, image text spans many domains.

der many circumstances such as poor lighting, motion blur, small character spacing, low resolution, and a high degree of background noise. Likewise, character classification is difficult due to the potentially large variety of fonts and natural variations in the appearance of characters. Although this two step approach works decently for reading generic text of all fonts in different environments, many times we wish to read text within a specific domain. Machine learning fits well for task specific applications since there is generally a large amount of training images available. We use a boosting framework based on Viola and Jones (2001) to learn character classifiers, allowing us to do task specific image text recognition. In order for our system to learn, a large amount of positives and negatives is required, and hand labeling images is one way to gather these

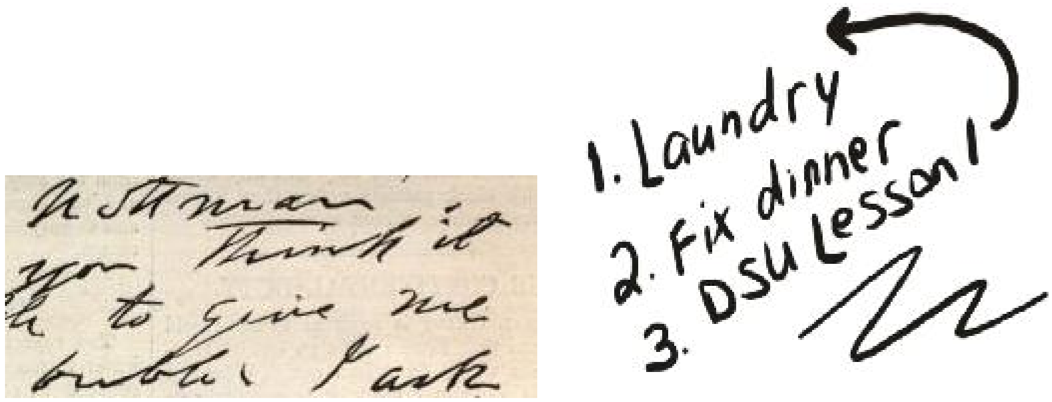

Figure 1.2: Handwriting is not considered image text. The difference between handwriting and image text quite distinct.

examples. Hand labeling can be very time consuming, so to alleviate the burden, we show that it is also possible to synthesize training data.

1.2 Thesis Structure

Chapter 2 discusses previous methods employed to read image text. In Chapter 3, we present our approach, and discuss its advantages and disadvantages. Results on experiments and comparison to a baseline algorithm are shown in Chapter 4. Finally, Chapter 5 discusses methods for efficiency improvements and suggestions for further research.

Chapter 2

Related Work

Automatically reading machine printed text predates computers, back to 1935 when Gustav Tauschek first obtained a patent for a machine that uses template matching with a photodetector (Tauschek, 1935). Since then, along with the development of computers, scanners, and cameras, a vast amount of work has been done on this problem. This section covers popular methods for image text recognition.

2.1 Conventional Approach

Throughout the decades, at the core of most image text reading algorithms, has been character segmentation, and character classification. They have largely been treated as separate problems, and are independently discussed below.

2.1.1 Character Segmentation

Character segmentation is the process of identifying regions in an image that represent characters (See Figure 2.1). One of the older methods for character segmentation is Projection Analysis, surveyed in (Casey and Lecolinet, 1996), and applied to license plates in (Zhang and Zhang, 2003). This process consists of summing pixel values across one dimension, where peaks and valleys of the projection give insight to

Figure 2.1: Character segmentation is the first step in most text reading algorithms

Figure 2.2: Taken from Chen and Yuille (2004), when images are too blurry, character segmentation fails

character locations. However, problems arise when a large amount of background noise and blur is present, or if there is little spacing between characters. More sophisticated segmentation techniques incorporate a form of binarization followed by connected component analysis to separate characters. An adaptive binarization technique introduced in (Niblack, 1986) is used by (Chen and Yuille, 2004) to suppress background noise in signs. The adaptive technique picks a threshold T for each pixel by looking at intensity statistics of a local window. The result is a binary image which may contain borders and other background noise as well, so connected component analysis is then used to identify characters. However, as Chen and Yuille (2004) show, this method can fail when image quality is poor or characters are very blurry (See Figure 2.2). These segmentation methods are chosen in a somewhat ad-hoc manner and do not learn segmentations from training data. As a result, these approaches may be good for reading a huge variety of image text domains, but for task specific purposes, they are not optimal.

2.1.2 Character Classification

Although methods in character segmentation rarely incorporate machine learning[1], there has been a strong trend in recent years toward its use in character classification. When a single font is present in the data, as is the case with reading Korean license plates in (Yu and Kim, 2000), template matching can be accurate and fast. However, when there is a large amount of noise, occlusion, and different fonts, machine learning can be very powerful. Methods incorporating neural networks have been successful on license plates (Nijhuis et al., 1995; Chang et al., 2004) and on degraded characters (Namane et al., 2005). Graphical models have been used more recently, incorporating image similarity among characters to boost recognition accuracy (Weinman and Learned Miller, 2006). These learning methods all require an initial step of feature extraction on the training images. A method for character feature extraction based on Gabor filters is successfully used in (Wang et al., 2005; Weinman and Learned Miller, 2006). However, the process of feature extraction is not obvious and generally independent of the learning algorithm used. As a result, the extra burden of choosing a feature extraction method for the learning algorithm exists.

Character classification can also be seen as a shape matching problem. Belongie et al. (2001) use shape contexts to describe interest points on a character and use these descriptors to find correspondences in other characters, and then estimate a transformation between the images. This work was extended to work on reading visual CAPTCHAs in (Mori and Malik, 2003). Such methods can be slow, and for blurry images, it is difficult to describe local regions by shape.

2.2 Other Approaches

Although most research has followed the convention described above, there has been work that more closely relates to our approach. An object detection method using

[1] We did find learning based segmentation methods for handwritten text, but this thesis does not focus on that domain.

Figure 2.3: Character classification is the process of assigning a character label to an image region

templates is taken in (Takahashi et al., 2007; Dlagnekov and Belongie, 2005), where individual characters are searched for in the image using template matching methods such as Normalized Cross Correlation (NCC). This approach is most similar to ours and is used as a baseline for our method.

Chapter 3

Our Approach

In our approach to image text recognition, we search the image for all characters in the alphabet. This can be thought of as exhaustively sliding a small window across the image and at each increment asking what character exists in that window, if any. We employ a boosting framework for object detection to do this (Viola and Jones, 2001). The framework, based on Adaboost, was made popular due to its accuracy and speed on face detection. Adaboost (Freund and Schapire, 1997) is a machine learning algorithm that learns a strong classifier by combining many weak classifiers (See Algorithm 1). The Viola-Jones learning framework has been extended to work on other classes of objects such as license plates (Dlagnekov and Belongie, 2005) and natural scene text (Chen and Yuille, 2004)[1]. While Freund and Schapire (1996) apply Adaboost to recognizing characters, their experiments are performed on clean, segmented data. We have not found any work which applies a Viola-Jones framework to reading text. Below, we discuss the assumptions required by our approach, present the boosting framework in detail, and give strengths and drawbacks of the approach.

[1] Note that detecting license plates and natural scene text is different than actually reading the text itself

Algorithm 1 Discrete Adaboost with Prior

Input: Number of rounds T, Prior p, Dataset $(x_1, y_1), ..., (x_n, y_n)$ where $y_i = -1, +1$ for negative and positive examples, respectively.

1: Initialize a distribution $D_1(i)$ where $D_1(i) = \frac{p}{m}$ for positives, and $D_1(i) = \frac{(1-p)}{l}$ where m and l are the number of positives and negatives, respectively.

2: **for** $t = 1$ to T **do**

3: Find the classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ that minimizes the error with respect to the distribution D_t:

$h_t = \underset{h_j}{\text{argmin}} \epsilon_j$, where $\epsilon_j = \sum_{i=1}^{m+l} D_t(i)[y_i \neq h_j(x_i)]$

4: if $\epsilon_t >= 0.5$, stop.

5: Compute $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ where ϵ_t is the weighted error rate of classifier h_t

6: Update:

$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$ where Z_t is a normalization factor so that $\sum_{i=1}^{m+l} D_{t+1}(i) = 1$

7: **end for**

8: Output the final classifier:

$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

3.1 Assumption

Since our approach is task specific, we assume that certain domain knowledge exists about the image text to be recognized. Domain knowledge can be represented implicitly in a set of training images, or it can be in the form of explicit image characteristics. Examples of domain knowledge can be environment parameters such as contrast, illumination, and noise. Additionally, knowledge of fonts is important because of the visual differences among characters across different fonts (See Figure 3.1). Obviously, the more constrained the domain is, the better the results will be. However, our approach is robust enough to handle a high degree of variation in image text if good training data is used.

Figure 3.1: The letter ”G” shown in a variety of fonts.

3.2 Learning Framework

The boosting framework learns an individual classifier for every character in the alphabet. Once the classifiers are learned, reading text can be done by running all character classifiers on a new image. Since character segmentation is avoided in this approach, it is unclear how to pick the correct size window to scan the image for characters. We call this process scale selection and leave it to be addressed within the context of the specific image text domain. It is assumed that the reader is familiar with the framework introduced in (Viola and Jones, 2001). Below we discuss the proposed flow of creating a text recognition system, which consists of hand labeling or image synthesis, training, and detection.

3.2.1 Hand Labeling and Image Synthesis

Since the process of training requires many labeled examples of positives and negatives, hand labeling examples to identify the regions where individual characters exist is one option. For each character in the alphabet, a human must iterate through a set of image text and draw a bounding box around every instance of that character. Since this can be very time consuming, another option is to automatically synthesize training data. In order to synthesize training data, plenty of domain knowledge must exist. In this thesis, we synthesized images of license plates, and discuss this further in Chapter 4. Once hand labeling or image synthesis is complete, we are ready to train our system to classify characters.

3.2.2 Training

An individual character classifier is learned by training a cascade of Adaboost classifiers (Viola and Jones, 2001). Three aspects of the framework we chose to emphasize are the use of priors, bootstrapping, and randomized features because they contributed to achieving good performance.

In the original Adaboost algorithm, equal weights are assigned to all examples. However, if we are training with more negatives than positives, it makes sense to assign weights proportional to the number of positives and negatives (Viola and Jones, 2001). Furthermore, in the first few stages of the cascade, we want Adaboost to focus on classifying positives correctly, versus classifying negatives correctly. As a result, Adaboost now has two parameters, the number of rounds, and a prior. The prior is a value between 0 and 1 which denotes what fraction of the weights should be assigned to the positive examples. For example, a prior of 0.9 tells us that all m positives will be initialized with a weight $\frac{0.9}{m}$, while all l negatives will be initialized with a weight of $\frac{0.1}{l}$ (See Algorithm 1). This additional parameter gives us more flexibility in the training process and allows us to achieve better performance.

Another key aspect to achieving good accuracy is bootstrapping in the cascade. After training a stage n, the negatives used to train stage n are discarded and the partial classifier (stages $1...n$) is used to classify unseen negatives in the training set. Only the misclassified negatives are added to the set of negatives which will be used in the following stage. The process of bootstrapping allows us to look at more difficult examples as the stage level increases in the cascade.

Each Adaboost classifier in the cascade is composed of decision stump classifiers based on randomized features. The boosting framework we use is designed to easily learn objects of many classes, not just individual characters. It may be argued that certain object classes require specially crafted feature sets. Chen and Yuille (2004) argue that the "choice of feature set is critical to the success and transparency of the algorithm." However, since Adaboost picks the best classifiers from a large set of weak classifiers, the learning process also does feature selection. We take the approach suggested by (Dollar et al., 2007), that the feature space we are exploring is potentially infinite, and that rather than hand crafting special features for a given task, we generate a large set of randomized features and let Adaboost do the work of feature selection. We define a feature $F_{C,H}(I)$ as a function where I is the image patch to be processed, C is a selected image channel of I, and H is a randomized Haar-like filter. H defines n random sub-regions as boxes with randomly chosen dimensions and locations on the scanning window (See Figure 3.2). A box indexed by i has a sign value $y_i \in \{-1, +1\}$. $\bar{c}_i$ denotes the the mean of the pixel values in C defined by box i. The feature is then computed as:

$$F_{C,H}(I) = \sum_{i=1}^{n} y_i \bar{c}_i$$

A large set of features is generated and passed to the Adaboost algorithm, where decision stumps are trained and weights are adjusted. Finally, Adaboost picks the best features and the process is repeated in the next stage of the cascade. Note that in this process, we not only learn a character classifier, we also use Adaboost to select our features. The process of generating a large set of randomized features is only done during training. Once a classifier is learned, only the features selected by Adaboost are computed over unseen data.

3.2.3 Detection

Once we have trained a classifier for each character in the alphabet, we are ready to run them on new images to read text. The process of running a classifier over an entire image is character detection. Each character detector is run independently on the image. A window of specified dimensions scans the entire image. At each increment, the region defined by the scanning window is classified as either a positive or negative. If classified as positive, it outputs a confidence level based on (Friedman et al., 2000).

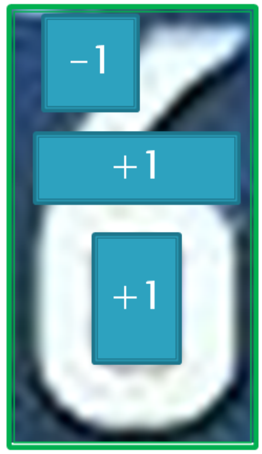

Figure 3.2: The filter is passed over an image patch as shown, but can be computed over a variety of image channels (X and Y gradients, grayscale, etc.).

The confidence level is computed as:

$$P(y = 1|x) = \frac{e^{F(x)}}{e^{-F(x)} + e^{F(x)}}$$

Where $F(x)$ is the weighted sum of the weak classifiers in the last stage of the cascade, and x is the image region being classified. Since the confidence is only computed for positives, $F(x)$ will be in the range [0.5 1.0]. We normalize this value so that it will be in the range [0 1]. After all detectors are passed through the image along with some scale selection process, any post-processing can be done based on domain knowledge. For example, it is known that for license plates, the characters lie on the same baseline, which can be used to weed out false positives.

3.3 Strengths and Drawbacks

Applying a boosting framework to recognizing characters has both strengths and drawbacks. The process of explicitly segmenting characters from the image is avoided, and instead incorporated into the learning process. Patches of the image background are treated as negatives during training and therefore weeded out during classification. This allows us to read images that would otherwise be difficult to segment. However, since segmentation is avoided, there is no clear way to determine the scale at which to search for characters. It is possible to search several scales exhaustively but in many cases there is domain knowledge about the scale, so we leave scale selection to be addressed within the desired domain. We feel that the drawbacks of character segmentation are greater than the the drawbacks of scale selection for task specific text recognition. Our solution is also more elegant than previous methods, since recognizing characters is entirely embedded in the learning process. Additionally, the learning process implicitly does feature selection from a large set of randomly generated features. These features are simple, and fast to compute, allowing for fast detection. Even though detection is fast, the approach is exhaustive, and is based on searching the image for all possible characters, at a variety of scales. This is far more time consuming than first segmenting characters, and then classifying them. However, with recent advances in efficiency of multi-class boosting and hardware improvements, performance can be improved dramatically (See Chapter 5).

Chapter 4

Results

a b

Figure 4.1: a) Examples from Single Font dataset. b) Examples from the Multiple Font dataset.

Experiments were performed on two separate test sets of license plate images. The Single Font test set contains blurry, low resolution California license plates. The Multiple Font test set contains license plates from every state in the country spanning a wide variety of fonts (See Figure 4.1 for example images from each dataset). We chose to measure the accuracy of individual character classifiers, since this is the core performance metric in our approach to text recognition. Results were compared to simple template classifiers based on Normalized Cross Correlation. Generic OCR software such as Tesseract (Smith, 2007) and ABBYY FineReader™ showed irrelevant results on our low resolution datasets. Consequently, we did not compare our results to these OCR packages.

4.1 Notes on Experiments and Viewing Results

4.1.1 Viewing Datasets and Results

Training and Test sets are available online along with classifier output and performance statistics on the test sets. Please visit the following link here[1] to access this data.

4.1.2 Visualization of Classifier Output

Throughout this section, there will be many examples of a character classifier's output on a test image shown below in Figure 4.2.

Figure 4.2: Blue box is ground truth. All other boxes are classifier output.

Generally, you will see a blue bounding box, specifying the labeled ground truth for a certain character classifier. The other colored boxes are output of the character classifier. The color of the boxes correspond to the the confidence level of the image patch. Green represents a high confidence, while red represents a low confidence (red is not shown in the figure). For example, since the digit "1" is enclosed in a blue box like in Figure 4.2, we know that high confidence green boxes are output of a "1" classifier.

4.1.3 Scale Selection

Since we do not focus on scale selection in this thesis, all images were resized such that the characters in the images were of a canonical width and height (10×20)

[1] http://vision.ucsd.edu/∼nbenhaim

prior to training and testing. All training and testing used a scanning window of 10×20. The image resizing is done as part of the training process, and so the images remain in their original resolution on disk. Test images showing output of classifiers have been resized to this canonical scale, and therefore some images look distorted.

4.2 Experiments on Single Font Dataset

We used a low cost monochrome CCD camera to capture frames at 320×240 resolution in a public parking lot. A license plate detector with a fixed-size scanning window of 34×68 was run on the recorded footage, and cropped out all license plates found in the frames. Approximately 50,000 plates[2] were cropped from the video footage and saved as separate images. Using a subset of these images, test sets were labeled only for digits 1-9 because creating test sets for every character would have been too time consuming (See Table A.1). These test sets allow us to quantify the performance of character classifiers. Below, we elaborate on template matching classifiers that were used as a baseline, followed by the boosted classifiers that were trained with both real and synthesized images. Finally, we discuss results and compare all three methods.

4.2.1 Template Matching Classifiers

In evaluating performance of boosted character classifiers, it is important to compare the results to a baseline approach. An ideal approach to compare to would be a text reading system that first does character segmentation, followed by character classification. We tried feeding images from the dataset to ABBYY FineReader™and Tesseract, but both systems couldn't identify text in many of the images. This is not surprising because these engines are meant to read higher quality scanned documents. Since segmentation is so difficult on this dataset, a good baseline can consist of replacing the boosted classifier with a template matching based classifier. This way the notion of

[2]This does not imply 50,000 unique plates and also includes false positives.

scanning a window across the image remains, but the classifier is treated as a black box. The baseline classifier consists of a character template and a threshold. At each window location, the Normalized Cross Correlation(NCC) is computed between the image patch defined by the window, and the character template. The NCC between two patches I_1 and I_2 is defined as:

$$NCC(I_1, I_2) = \frac{\sum_x (I_1(x) - \bar{I}_1)(I_2(x) - \bar{I}_2)}{\sqrt{\sum_x (I_1(x) - \bar{I}_1)^2 \sum_x (I_2(x) - \bar{I}_2)^2}}$$

where

$$\bar{I}_1 = \frac{1}{N} \sum_x I_1(x), \bar{I}_2 = \frac{1}{N} \sum_x I_2(x)$$

are the means of I_1 and I_2 respectively.

If the score between the a template and an image patch is above the specified threshold, the image patch is classified as true, which means the character is present. This classifier can be seen as nearest neighbor classification because we are computing the normalized dot product between an image patch and a prototype font template, and then assigning a character label based on a threshold. Since California license plates use Penitentiary Gothic font, the templates used are 10×20 images of Penitentiary Gothic characters (See Figure 4.3c for examples of templates). Template classifiers were evaluated for digits 1-9 on the test set using threshold values of 0.5, 0.6, 0.7, and 0.8. Template classifiers with a threshold of 0.7 performed quite well on the test sets, but did not out perform the boosted classifiers.

4.2.2 Classifiers Trained with Real Images

A training set was hand labeled for digits 1-9 using the captured license plate data in the same way that the test sets were labeled (See Table A.2). A separate classifier was trained for each digit using the boosting framework. Each classifier required minor parameter tweaking to achieve the best performance on the test set. See Table A.4 for the

Figure 4.3: a&b) Examples of backgrounds used to synthesize plates. c) Penitentiary Gothic templates used to synthesize Single Font set. d) Character templates used to synthesize Multiple Font set

parameters used to train each digit. Parameters were chosen to optimize classification accuracy, and detection speed was not taken into account when designing the cascade. As a result, character detectors were extremely slow on the test sets(See Table A.7). However, training a classifier took only a few minutes. Accuracy for these classifiers was very good, even though we used a small feature set, and only a few hundred positives for training a classifier. See Figures 4.4 and 4.5 for for good and bad output of digit classifiers on test images.

4.2.3 Classifiers Trained with Synthesized Images

One of our goals is to show that with enough domain knowledge, we can avoid hand labeling a training set by synthesizing images and automatically generating labels. For this data set, there are generally dark Penitentiary Font digits against a light background. Therefore, we placed randomly chosen font templates on top of randomly chosen backgrounds (See Figure 4.3). Then, we tried to capture some of the real data's properties by applying slight rotation on each axis, adding motion blur, varying the image resolution, and mimicking shadow lines (See Figure 4.6 for synthesized images).

Figure 4.4: These are examples for which digit classifiers performed well on the test images.

Figure 4.5: These are examples for which digit classifiers performed poorly on the test images. Top row shows false negatives. Bottom row shows false positives.

Thousands of images were synthesized along with the location and dimensions of each character on the plate. The data generated by the synthesizer was then used to train a new set of digit classifiers. Training good classifiers with synthetic data required thousands of examples as opposed to a few hundred needed to train with real data (See Table A.3). As a result, training took about 15-20 minutes, significantly longer than training with the hand labeled set. Training parameters were initially optimized only for classification accuracy. However, we then retrained all the classifiers and increased the number of stages in the cascade (See Table A.5). Consequently, the accuracy remained very good but the detection time went down significantly. In this thesis, we only present results of

Figure 4.6: Examples of synthesized license plates. Notice how shadow lines and brightness level of the plate vary.

the classifiers optimized for both speed and accuracy.

4.2.4 Comparison

Both classifiers trained with real images and synthesized images performed very well, but those trained with real images performed best. Template matching classifiers did not achieve accuracy as good as any of the boosted classifiers. Figure 4.7 shows a visual comparison between all three kinds of classifiers passed on the test images. For each type of classifier (trained with real, trained with synthesized, and template), an average ROC curve (See Figure 4.8) was generated by evaluating each digit classifier over a series of confidence thresholds, and then averaging the true positive rates, and false positive rates over all digits. Notice that the same templates used to synthesize license plate images were used in the template classifiers. Even though the same templates were used, the boosted classifiers trained with synthesized images performed much better than the template classifiers. It is clear, however, that training a learning algorithm with real images yields the best performance. Additionally, training with real images only required a small number of hand labeled positives. We believe that if a large amount of data is available, then it is better to spend time hand labeling training data, than trying to make a good synthesizer. Also, notice how detection time is affected by the parameters of the cascade (See Table A.7).

Figure 4.7: Each row shows ouput of the different classifiers on the same test image. Notice the improvement moving from left to right. In some cases all classifiers perform equally well, as is the case in the fourth row.

4.3 Experiments on Multiple Font Dataset

California parking lots are mostly occupied by cars with California license plates. To get a multiple font dataset, we went to http://www.worldlicenseplates.com and collected 313 different license plates containing a variety of fonts and backgrounds. The set contained several license plates from every state in America. Test sets were hand labeled for characters “1”, “4”, “D”, “P” (See Table A.8). In choosing characters to test, we focused both on characters that varied across different fonts, and characters that were similar across different fonts. “4” seemed to vary the most, while “D” and “1” had minor variations, and “P” had few variations. Although the letter “D” doesn't vary all that much across fonts, it can get confused with “O” and “0”. Since the dataset of plates is so small, we were not able to hand label training sets, so we only trained classifiers with synthesized images. Below, we go into further detail on the baseline tests

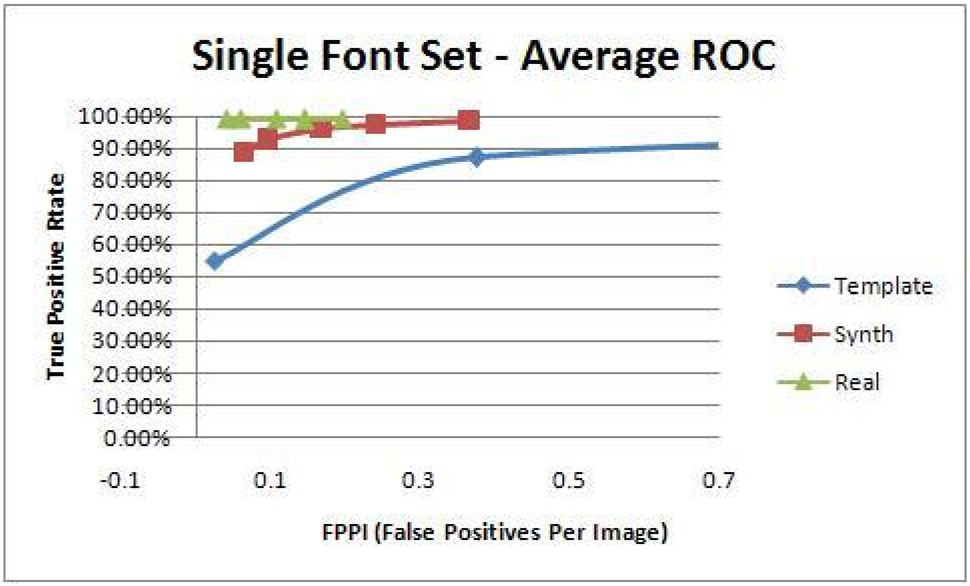

Figure 4.8: Average ROC curve for template classifiers, classifiers trained with real data, and classifiers trained with synthesized data. Two data points on the template classifiers were left out for the purposes of comparison. It is clear though, that the boosted classifiers outperform template classifiers.

using template matching, and the tests using boosted classifiers trained with synthesized images. Finally, we show and compare results, and provide further analysis.

4.3.1 Template Matching Classifiers

Since this dataset contains many fonts, a template classifier becomes more exhaustive as it needs to have a template corresponding to each font for a given character. However, we did not extend the template classifiers in such a way when doing these tests. The templates used were the same Penitentiary Gothic font images used in the Single Font experiments. Rather than spending time to create good template classifiers for these tests, we focused our attention on the boosted classifiers. We can infer from the Single Font experiments that even if there is only one font in the dataset, classifiers trained with a boosting framework outperform template classifiers. As expected, using

Figure 4.9: Examples of multiple font synthesized license plates.

templates resulted in poor accuracy on the Multiple Font set.

4.3.2 Classifiers Trained with Synthesized Images

The Multiple Font set differs from Single Font set in the number of fonts, backgrounds, and contrast reversal (light characters on a dark background). In order to synthesize these characteristics, we simply extended the Single Font plate synthesizer to include more fonts, and randomly apply contrast reversal. However, additional templates were only added for the characters we tested on [3]. Additional backgrounds were not added to the synthesizer because adding backgrounds did not improve performance. See Figure 4.3d for examples of font templates used for each test character, and Figure 4.9 for synthesized images of multiple font license plates. Once again, thousands of positive examples were needed for training (See Table A.6). For good and bad examples of classifier output on test images, see Figures 4.10 and 4.11. It is especially interesting to see a classifier's output on an image where both dark and light letters exist (See Figure 4.12). Applying a character segmentation to such an image would be quite difficult.

[3]Font templates were not added for "P" since there was almost no font variation for this letter.

Figure 4.10: Examples where the classifiers performed well.

Figure 4.11: Examples where the classifiers failed or performed poorly.

4.3.3 Comparison

Boosted classifiers significantly outperform template classifiers as expected. For a visual comparison of the two methods on test images, see Figure 4.13. Classifiers

Figure 4.12: Example of a "D" Multiple Font classifier on a test image. Notice the fact that it detects "D" with different contrast in the same image and does not detect the "0".

were evaluated at different thresholds and an average ROC curve was plotted for both template and boosted classifiers (See Figure 4.14). In both cases, there are significantly more false positives than there were on the Single Font test sets. One of the main reasons is that the synthesized plates did not capture the characteristics of the dataset very well. Another is the small size of the test data available. Additionally, the more variations in the object class being learned, the more difficult it will be for Adaboost to discriminate between positives and negatives. This can best be demonstrated by observing the top features picked for a Single Font "4" boosted classifier (See Figure 4.15). The top two features picked represent the regions where the diagonal bar on the four exists, and doesn't exist. Now let's observe the top features picked for a Multiple Font "4" classifier (See Figure 4.16). Most of the top features picked here are in the bottom half of the four since there are variations in the top half of Multiple Font fours. Therefore, as structural variations in a character increase, the harder it will be to train a good classifier. In conclusion, results on multiple font sets using synthesized images are good, and we believe that training with real images will yield far better performance.

Figure 4.13: Visual comparison of template classifiers versus boosted classifiers on test images.

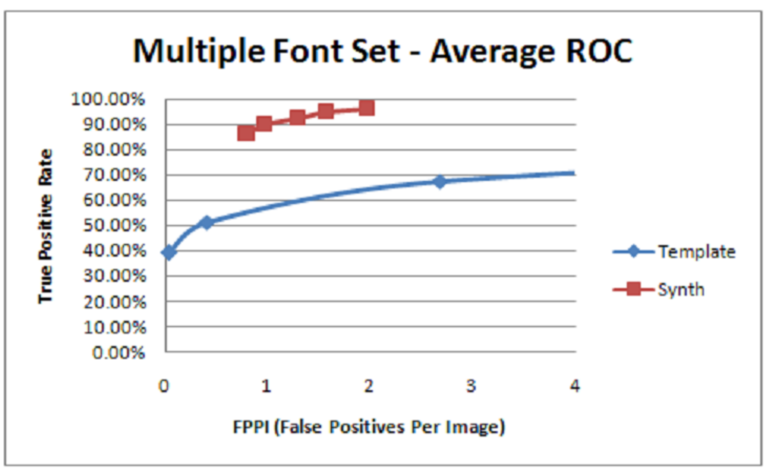

Figure 4.14: Average ROC curves for template classifiers and boosted classifiers.

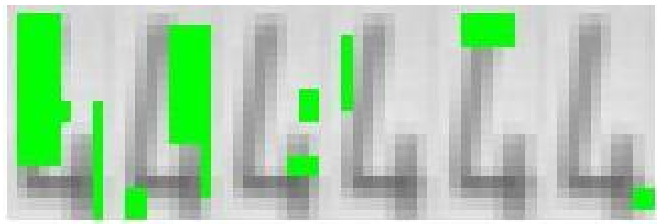

Figure 4.15: These are the top five features picked for a Single Font “4”.

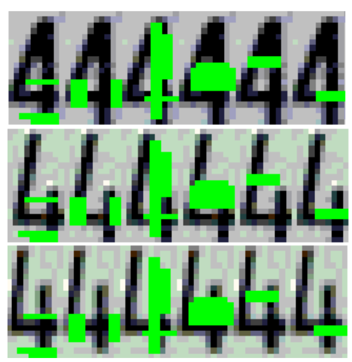

Figure 4.16: These are the top five features picked for a Multiple Font “4”. Each row shows the same features on top of “4”s of different fonts

Chapter 5

Conclusions and Future Work

In this thesis, we have proposed a method for image text recognition using a boosting framework which allows us to avoid explicit character segmentation. The method is entirely based on machine learning and is therefore best suited for task specific purposes. As a result, characters can be recognized in blurry images, and with varying fonts. We also show that training images can be synthesized which allows us to avoid hand labeling. However, the best performance is achieved when real images are used to train character classifiers. Although our results are very encouraging, several challenges and limitations still exist. Below, we discuss recent work that addresses many of these challenges.

5.1 Labeling

In our experiments, training classifiers with only a few hundred real examples performed better than training with thousands of synthesized examples. It is therefore clear that training with real images is preferable to create the most accurate classifiers. Since data is so easy to collect and cheap to store, there is a strong desire to quickly label it for machine learning algorithms. Semi-automatic Visual Learning (SEVILLE) is a labeling framework which uses an initial classifier to help reduce the number of images that the user has to sift through to label (Abramson and Freund, 2006). After enough new examples are labeled, a new detector is trained and is replaced with the old detector. SEVILLE reduces the amount of time required to label a large set of training images. However, even this process can be time consuming if many different object classes need to be labeled. The Soylent Grid is an infrastructure designed to distribute the labeling task to a large number of people over a network (Steinbach et al., 2007). This framework would plug into the visual CAPTCHA component of participating web sites. Instead of seeing a visual CAPTCHA while trying to access a web site, the user would be given two labeling tasks: one for which the ground truth is known, and another for which the ground truth is unknown. Considering the amount of visual CAPTCHAs filled out every day, replacing these by labeling tasks from the Soylent Grid would allow for vast amounts of data to be labeled quickly. Additionally, human labeling can be turned into games such as the Google™Image Labeler [1]. However, it is difficult to create games which ask for specific tasks to be completed, such as cropping out regions, and answering specific questions about an image. It is these cases for which the Soylent Grid can be superior.

The fastest way to label image text is to type in the phrase, word, or number embedded in the image. However, simply knowing whether a character or word exists in an image is not enough information for the Viola Jones framework to learn a classifier. The exact bounding box of each character needs to be known. Algorithms have been proposed which only need an image, and a binary label as input. These algorithms fall into the category of Multiple Instance Learning (MIL). MIL algorithms were created because the task of specifying a bounding box is much more time consuming than marking whether the object of interest exists in the image or not. Integrating the Viola-Jones detection framework into a MIL algorithm was done in (Viola et al., 2005), which is the most closely related MIL work to this thesis.

Active labeling, harnessing the power of the internet, and new developments in machine learning, are all helping in the ability to label huge amounts of data that would

[1] http://images.google.com/imagelabeler

otherwise cost unreasonable amounts of time and money.

5.2 Efficiency

The most computationally expensive part of an object detector is computing the many haar features over each location of the scanning window. In this thesis, we trained individual character classifiers and proposed that each classifier be run over an image independently. However, there are features that could be shared across many different characters, and therefore reduce the number of features that need to be computed (See Figure 5.1). A new variant of Adaboost is presented in (Torralba et al., 2004), which picks the weak learner in each round that minimizes the error across multiple classes of objects. This allows features to be picked which are shared among many classifiers. The number of features is observed to grow almost logarithmic in the number of object classes. This results in having to compute far less features and therefore improving efficiency. There are also lower level optimizations that can be done to greatly speed up detection rates. With advances in parallel computing, the prospect of assigning each detector a CPU does not seem unreasonable. In conclusion, multi-class object detection will become much faster with better algorithms, and advances in hardware.

5.3 Generalizing to Many Fonts

Although we show classifiers can recognize multiple fonts for certain characters, we do not know whether the features used are powerful enough to generalize to a large number of fonts. Some fonts may have characters that look so different, they would require an object class of their own. It may be that for generic image text recognition, a different approach would be required, such as shape matching as discussed earlier. We leave these questions to be answered in future work.

Figure 5.1: Shown above are the top features picked for a “3”, and a “8”. Notice that the fifth feature on the “8” is very similar to the third feature on the “3”

Appendix A

Table Appendix

Table A.1: Number of instances and examples for hand labeled Single Font test sets of each digit.

Digit	# Instances	# Images
1	259	212
2	161	153
3	209	181
4	303	293
5	160	149
6	207	188
7	218	168
8	130	115
9	235	303
Mean	209	185

Table A.2: Number of instances and examples for hand labeled Single Font training sets of each digit.

Digit	# Instances	# Images
1	144	1091
2	249	715
3	203	1187
4	271	897
5	249	443
6	227	1023
7	230	927
8	287	1137
9	200	1058
Mean	229	942

Table A.3: Number of instances and examples for synthesized Single Font training sets of each digit.

Digit	# Instances	# Images
1	3229	2778
2	3195	2745
3	3223	2765
4	3286	2816
5	3262	2818
6	3375	2877
7	3164	2733
8	3227	2785
9	3329	2872
Mean	3254	2799

Table A.4: Training parameters for Single Font classifiers trained w/real images

	#Rounds per Stage	Prior per Stage	Features Per Stage	Negatives Per Stage
1	30,60,100,150	0.7,0.3,0.2,0.1	500	3000
2	12,30,60,100	0.9,0.3,0.2,0.1	2000	1000
3	12,30,60,100	0.8,0.2,0.1,0.1	500	3000
4	12,30,60,100	0.9,0.7,0.7,0.7	500	3000
5	12,30,60,100	0.9,0.9,0.9,0.9	500	3000
6	12,30,60,100	0.8,0.3,0.2,0.1	300	3000
7	12,30,60,100	0.9,0.5,0.5,0.5	500	3000
8	12,30,60,100	0.9,0.5,0.5,0.5	500	3000
9	30,60,100	0.5,0.5,0.5	500	3000

Table A.5: Training parameters for Multiple Font classifiers trained w/synthesized data

	#Rounds per Stage	Prior per Stage	Ftrs/Stage	Negatives/Stage
1	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.9,0.8	300	3000
2	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.8,0.5	300	3000
3	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.8,0.8	300	3000
4	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.9,0.9	300	3000
5	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.8,0.7	300	3000
6	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.8,0.7	300	3000
7	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.9,0.9	300	3000
8	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.9,0.9	300	3000
9	1,3,6,12,30,60,100	0.99,0.9,0.9,0.9,0.9,0.9,0.9	300	3000

Table A.6: Number of instances and images for each character's synthesized Multiple Font training set.

Character	# Instances	# Images
1	3326	2888
4	3264	2784
D	895	867
P	918	889
Mean	2101	1857

Table A.7: Average Detection Times Per Image. These numbers represent the average amount of time to run a classifier over an image.

Classifiers	Mean Detect Time (ms)	Standard Deviation (ms)
Template Matching using NCC	0.56	0.22
Trained with Real Data	50.8	27.3
Trained with Synthesized Data	13.7	3.4

Table A.8: Number of instances and examples for each character in the multiple font test set

Character	# Instances	# Images
1	137	310
4	114	310
D	33	310
P	27	310
Mean	78	310

References

Abramson, Y., and Freund, Y., 2006: Active learning for visual object recognition. *UCSD Report*, **1**.

Belongie, S., Malik, J., and Puzicha, J., 2001: Matching shapes. In *International Conference on Computer Vision*, I: 454–461.

Casey, R., and Lecolinet, E., 1996: A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **18**(7), 690–706.

Chang, S., Chen, L., Chung, Y., and Chen, S., 2004: Automatic license plate recognition. *IEEE Trans. Intelligent Transportation Systems*, **5**(1), 42–53.

Chen, X., and Yuille, A., 2004: Detecting and reading text in natural scenes. In *IEEE Computer Vision and Pattern Recognition or CVPR*, II: 366–373.

Dlagnekov, L., and Belongie, S., 2005: Recognizing Cars. Technical report, Technical Report CS2005-0833, UCSD CSE, 2005.

Dollar, P., Tu, Z., Tao, H., and Belongie, S., 2007: Feature mining for image classification. In *IEEE Computer Vision and Pattern Recognition or CVPR*, 1–8.

Freund, Y., and Schapire, R., 1996: Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, **148**, 156.

Freund, Y., and Schapire, R., 1997: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, **55**(1), 119–139.

Friedman, J., Hastie, T., and Tibshirani, R., 2000: Additive logistic regression: a statistical view of boosting. *Ann. Statist*, **28**(2), 337–407.

Mori, G., and Malik, J., 2003: Recognizing objects in adversarial clutter: Breaking a visual captcha. In *IEEE Computer Vision and Pattern Recognition or CVPR*, I: 134–141.

Naito, T., Tsukada, T., Yamada, K., Kozuka, K., and Yamamoto, S., 2000: Robust license-plate recognition method for passing vehicles underoutside environment. *Vehicular Technology, IEEE Transactions on*, **49**(6), 2309–2319.

Namane, A., Arezki, M., Guessoum, A., Soubari, E., Meyrueis, P., and Bruynooghe, M., 2005: Sequential neural network combination for degraded machine-printed character recognition. *Proceedings of SPIE*, **5676**, 101.

Neubauer, C., and Tyan, J., 2003: US Patent 6,553,131: License plate recognition with an intelligent camera.

Niblack, W., 1986: *An Introduction to Digital Image Processing*. Prentice Hall.

Nijhuis, J., Ter Brugge, M., Helmholt, K., Pluim, J., Spaanenburg, L., Venema, R., and Westenberg, M., 1995: Car license plate recognition with neural networks and fuzzy logic. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, **5**.

Smith, R., 2007: An overview of the tesseract ocr engine. In *International Conference on Document Analysis and Recognition*, 629–633.

Steinbach, S., Rabaud, V., and Belongie, S., 2007: Soylent grid: It's made of people. In *International Conference on Computer Vision*, 1–7.

Takahashi, Y., Tanaka, H., Suzuki, A., Shio, A., and Ohtsuka, S., 2007: License plate recognition using gray image template matching with noise reduction filters and character alignment. *Systems and Computers in Japan*, **38**(3), 49–61.

Tauschek, G., 1935: US Patent 2,026,329: Reading Machine.

Torralba, A., Murphy, K., and Freeman, W., 2004: Sharing features: Efficient boosting procedures for multiclass object detection. In *IEEE Computer Vision and Pattern Recognition or CVPR*, II: 762–769.

Viola, P., and Jones, M., 2001: Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Vision and Pattern Recognition or CVPR*, I:511–518.

Viola, P., Platt, J., and Zhang, C., 2005: Multiple instance boosting for object detection. *Proceedings of NIPS*.

Wang, X., Ding, X., and Liu, C., 2005: Gabor filters-based feature extraction for character recognition. *Pattern Recognition*, **38**(3), 369–379.

Weinman, J., and Learned Miller, E., 2006: Improving recognition of novel input with similarity. In *IEEE Computer Vision and Pattern Recognition or CVPR*, I: 308–315.

Yu, M., and Kim, Y., 2000: An approach to Korean license plate recognition based on vertical edge matching. *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, **4**.

Zhang, Y., and Zhang, C., 2003: A new algorithm for character segmentation of license plate. *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, 106–109.