# Identify Characters from Google Street View Pictures

Liuyi Hu, Lin Su, Alison Wu

## 1 Introduction

Street view recognition is a problem that plays an important role in our life. For example, the ability to automatically transcribe an address number from a geo-located patch of pixels and associate the transcribed number with a known street address helps pinpoint, with a high degree of accuracy, the location of the building it represents. This is particular useful in places where street numbers are otherwise unavailable or places such as Japan and South Korea where streets are rarely numbered in chronological order but in other ways such as the order in which they were constructed, a system that makes many buildings impossibly hard to find, even for locals.

But images obtained from Google Street View pose a formidable challenge for character recognition. Unlike the traditional optical character recognition with pure bit map inputs, the images from street view are presented in many different fonts, styles, colors and orientations, and there are additional environmental factors at play such as lighting and shadows.

The objective of this project is to come up with a good approach to identify characters from Google Street view pictures, which is an active competition at www.kaggle.com. In our work, we used Naive Bayes, Support Vector Machine based on Histogram of Oriented Gradients (HOG), and Random Forest. We evaluated the performance of these three methods on the test dataset on www.kaggle.com, and SVM based on HOG features reached the best performance in terms of prediction accuracy.

## 2 Data Processing

The datasets we use are Google Street View images of characters taken from the Chars74K data [3]. There are 6283 labeled images in the training set and 6220 unlabeled images in the test set. Our goal is to classify images in the test set into 62 categories (0-9, a-z, A-Z) based a predictive model built using the training set.

We decided to pre-process the images to reduce as much noise as possible to increase the accuracy of predictive models. The data processing stage consists of the following steps. (1) Given an original RGB image from the original data set, it was converted to a greyscale image. (2) The greyscale image was then converted into a binary image based on threshold computed by the Otsu's method[8] to minimize the intraclass variance of the black and white pixels. (3) The boundary pixels of binary image was evaluated to reassign the character pixels with 1 and background pixels with 0. (4) The binary image went through repeated process of cropping and removal of connected components that was less than p pixels and connected to the boundary until no such component exists. (5) Finally, the image was cropped and resized to 32*32. Through the image processing step, we were able to increase our prediction accuracy from the benchmark of 0.429 to 0.692 using Julia random forest model and parameters provided by Kaggle[9]. Figure 1 shows some of the images before and after processing.

(a) Before image processing

(b) After image processing

Figure 1: Examples of Google Street View images before and after processing

# 3 Models and Methods

## 3.1 Naive Bayes

We used 32*32 resized data for this model. Each image is a 32*32 bitmap in which each element is a binary value indicating one pixel of black or white.

If each bitmap is divided into blocks of m*m, and the number of white pixels are counted in each block, then for each image, we get a vector $\boldsymbol{x} = (x_1, \ldots, x_d)$, where $d = \frac{32*32}{m^2}$. Possible choice of m are $m = 2$ or $m = 4$. We used the Dirichlet-Multinomial distribution to model the distribution of $\boldsymbol{x}$[10]. That is, suppose $\boldsymbol{x} \sim Mutinomial(\boldsymbol{p})$, where $\boldsymbol{p} = (p_1, \ldots, p_d)$, and $\boldsymbol{p}$ follows a Dirichlet distribution with parameter vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$, where $\alpha_j > 0$. So

the prior density of $\boldsymbol{p}$ is

$$\pi(\boldsymbol{p}) = \frac{\Gamma(|\boldsymbol{\alpha}|)}{\prod_{j=1}^{d}\Gamma(\alpha_j)}\prod_{j=1}^{d} p_j^{\alpha_j - 1}$$

where $|\boldsymbol{\alpha}| = \sum_{j=1}^{d}\alpha_j$. And therefore the Dirichlet-Multinomial distribution of $\boldsymbol{x}$ is

$$f(\boldsymbol{x}|\alpha) = \begin{pmatrix} |\boldsymbol{x}| \\ \boldsymbol{x} \end{pmatrix} \frac{\prod_{j=1}^{d}(\alpha_j)_{x_j}}{(|\boldsymbol{\alpha}|)_{|\boldsymbol{x}|}}$$

where $|\boldsymbol{x}| = \sum_{j=1}^{d} x_j$, and $(a)_k = \prod_{i=0}^{k-1}(a+i) = \frac{\Gamma(a+k)}{\Gamma(a)}$.

Given independent data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, the log-likelihood of $\boldsymbol{\alpha}$ is

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} ln \begin{pmatrix} |\boldsymbol{x_i}| \\ \boldsymbol{x_i} \end{pmatrix} + \sum_{i=1}^{n}\sum_{j=1}^{d}[ln\Gamma(\alpha_j + x_{ij}) - ln\Gamma(\alpha_j)] - \sum_{i=1}^{n}[ln\Gamma(|\boldsymbol{\alpha}| + |\boldsymbol{x}_i|) - ln\Gamma(|\boldsymbol{\alpha}|)]$$

Using Newton's method or EM or MM algorithm, we can calculate the MLE of $\boldsymbol{\alpha}$ given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

In our case, there are 62 categories. Using the training data, we can get an MLE of $\boldsymbol{\alpha}$ for each category. Then, we can construct a simple Bayesian rule to recognize each image in the test set, which is,

$$\boldsymbol{x} \mapsto \underset{k}{argmax}\ \ \hat{\pi}_k f(\boldsymbol{x}|\hat{\boldsymbol{\alpha}}_k)$$

where $\boldsymbol{x} \in$ test set, $k = 1, 2, \ldots, 62$, and prior probability $\hat{\pi}_k$ is the proportion of category $k$ in the training set.

Table 1 shows the accuracy rate from 4-fold cross-validation of the training set. Among all parameters chosen, $m = 2$ has the best performance, therefore, we use it in the final prediction.

| block size(m) | 4-fold CV Accuracy |
|---|---|
| 2 | 0.6069(.0156) |
| 4 | 0.5903(.0091) |

†Standard errors are given in parentheses

Table 1: Summary of accuracy using Dirichlet-Multinoial model

## 3.2 SVM using HOG features

### 3.2.1 HOG

Histogram of oriented gradients (HOG) is a feature descriptor used to detect objects in computer vision and image processing. It was first described by Navneet Dalal and Bill Triggs[4]. The essential thought behind the Histogram of Oriented Gradient descriptors is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. To obtain the HOG features, we first divide the image into small connected regions called cells, and compute a histogram of gradient directions for each cell or edge orientations for the pixels within the cell. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms. And the set of these block histograms represents the descriptor.

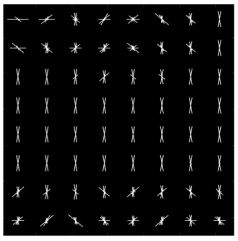
For different picture sizes, the cell size is chosen differently. Figure 2 shows the original picture we are processing and extracted HOG descriptors using different cell sizes. The visualization shows that a cell size of [16 16] does not encode much shape information, while a cell size of [4 4] encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. A good compromise is a cell size of [8 8]. This size setting encodes enough spatial information to visually identify a digit shape while limiting the number of dimensions in the HOG feature vector, which helps speed up training. During training process, we have tried all these three different cell sizes, and cell size of [8 8] performs best in terms of prediction accuracy.

### 3.2.2 SVM

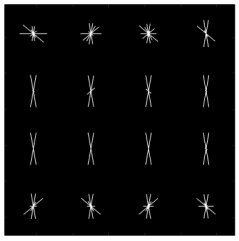
Support Vector Machines (SVM) is a state-of-the-art learning machine based on the structural risk minimization induction principle, and has been widely applied to machine vision fields such as character, handwriting digit and text recognition. SVM functions by projecting the training data $x$ in the input space to a feature space of higher (infinite) dimension by mapping $\phi(x)$. The mapping can be implicitly defined by introducing the so-called kernel function $K(x_i, x_j)$ which computes the inner product of vectors $\phi(x_i)$ and $\phi(x_j)$.The typical kernel function includes the linear function $K(x_i, x_j) = x_i^T x_j$, the radial basis function $K(x_i, x_j) = exp(\frac{||x_i - x_j||^2}{\sigma^2})$ and the polynomial function $K(x_i, x_j) = (x_i^T x_j + 1)^p$. However, SVM classification is essentially a binary (two-class) classification technique, which has to be modified to handle the multi-class tasks. The conventional way is to decompose the M-class problem into a series of two-class problems and construct several binary SVM learners.

The earliest and one of the most widely used implementations is the "one-vs-all" method[1],

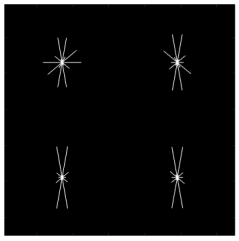
(a) Original picture



(b) CellSize = [4 4]



(c) CellSize = [8 8]



(d) CellSize = [16 16]

Figure 2: HOG Features

which constructs M SVM classifiers with the $i$th one separating class $i$ from all the remaining classes, i.e., the $i$th SVM is trained with all of the examples in the $i$th class with positive labels and all the other examples with negative labels. Mathematically the $i$th SVM solves the following problem that yields the $i$th decision function $f_i(x) = w_i^T \phi(x) + b_i$:

$$\begin{aligned} &minimize : \frac{1}{2}||w_i||^2 + C\sum_{l=1}^{N} \epsilon_l^i \\ &subject\ \ to : y_j^*(w_i\phi(x_j) + b_i) \geq 1 - \epsilon_l^i, \epsilon_l^i \geq 0 \end{aligned}$$

where $y_j^* = 1$ if $y_j$ is in class $i$ and $y_j^* = -1$ otherwise.

At the classification phase, a sample $x$ is classified as in the class $k$ whose $f_k(x)$ produces the largest value, i.e.

$$k = \underset{i=1,\cdots,M}{argmax}\ \ f_i(x) = \underset{i=1,\cdots,M}{argmax}\ \ w_i^T\phi(x) + b_i$$

Another major method is called the "one-vs-one" method. This idea was first introduced in[6] and the first use of this strategy on SVM was by Krebel[7]. The method constructs $\frac{M(M-1)}{2}$ binary SVM classifiers, each for every distinct pair of classes. Each of binary clas-

sifiers takes sample from one class as positive and samples from another class as negative. Mathematically the binary SVM between class $i$ and class $j$ solves the following problem:

$$minimize : \frac{1}{2}||w_{ij}||^2 + C\sum_{k=1}^{N}\epsilon_k$$
$$subject\ \ to : y_k^*(w_{ij}\phi(x_k) + b_{ij}) \geq 1 - \epsilon^k, \epsilon_k \geq 0$$

where $x_k$ are the sample from class $i$ or class $j$, and $y_k^* = 1$ if $y_k$ is from class $i$ and $y_k^* = -1$ if $y_k$ is from class $j$.

For prediction at a point, each classifier is queried once and issues a vote. The class with the maximum number of votes is the winner.

In order to evaluate the performance of these two methods, we implemented three different kernel functions: Linear, Polynomial and Radio Basis Function(RBF). Each of them were employed to carry out "one-vs-one" method and "one-vs-all" method. Table 2 gives a summary of the prediction accuracy based on a 4-fold cross validation. It is evident that the "one-vs-all" approach to multiclass classification has exhibited a better prediction result than "one-vs-one" approach. And among all the combinations, the RBF SVM learner with "one-vs-all" methods has the highest prediction accuracy.

| SVM Kernel | one-vs-one | one-vs-all |
|---|---|---|
| Linear | 0.7134(.0069) | 0.7254(.0138) |
| RBF | 0.6040(.0130) | 0.7457(.0072) |
| Polynomial | 0.7161(.0109) | 0.7313(.0099) |

†Standard errors are given in parentheses

Table 2: Summary of accuracy using different SVM learners

## 3.3 Random Forest

Random forest classification is an ensemble learning method that uses bootstrap aggregation techniques and random selection of features. The algorithm was first developed by Leo Breiman and Adele Cutler. In a random forest model, classification trees are fitted repeatedly on bootstrap samples, the final classification decision is the majority votes of the trees. Random forest is effective on large data set with thousands of variables. It is robust to outliers and missing values in the data[2]. Unlike decision trees, random forests do not

overfit as number of trees increases. Predictions can be made quickly once the model is built.

Given a training set of predictors $X = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\}$ with labels $Y = \{\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}\}$, we construct a random forest of B trees, $\{T_1, \ldots, T_B\}$, by building tree $T_b$ with the following steps:

(1) Draw a bootstrap sample $X_b^*, Y_b^*$ of size N from the training set.
(2) Grow a random-forest tree $T_b$ by repeatedly select the best variable among the $m \ll d$ input variables to split the terminal nodes until the minimum node size is reached.

For a new predictor $\boldsymbol{x}$, the classification is the majority vote across all B trees [5].

We fit random forest model on processed Street View images vectors and the corresponding HOG feature vectors respectively using the DecisonTree package in Julia. The parameters in the model are the number of trees B, number of selected feature m, and subsample size N. Using 4 fold cross validation on the training set. Table 3presents the best prediction result using 50 features, 90 trees and 70 % of data as bootstrap subsample.

| Data | 4fold CV Accuracy |
|---|---|
| Processed Images | 0.7075(.0176) |
| HOG Features | 0.7142(.0104) |

†Standard errors are given in parentheses

Table 3: Summary of accuracy using Random Forest

# 4 Results

Table 4 shows the best prediction accuracy using 49 % of the test data (results are provided by www.kaggle.com). SVM using HOG with RBF kernel and one-vs-all method has the best performance. Figure 3 shows the best ranking we have using SVM as of April 27, 2015 (our account name is Brian).

| Method | Accuracy | Ranking by Apr.27, 2015 |
|---|---|---|
| Naive Bayes | 0.6057 | 26 |
| Random Forest | 0.7327 | 7 |
| **SVM*** | **0.7660** | **3** |
| Julia Random Forest Benchmark | 0.4293 | 57 |
| Julia kNN Benchmark | 0.4058 | 62 |

Table 4: Summary of accuracy on test set

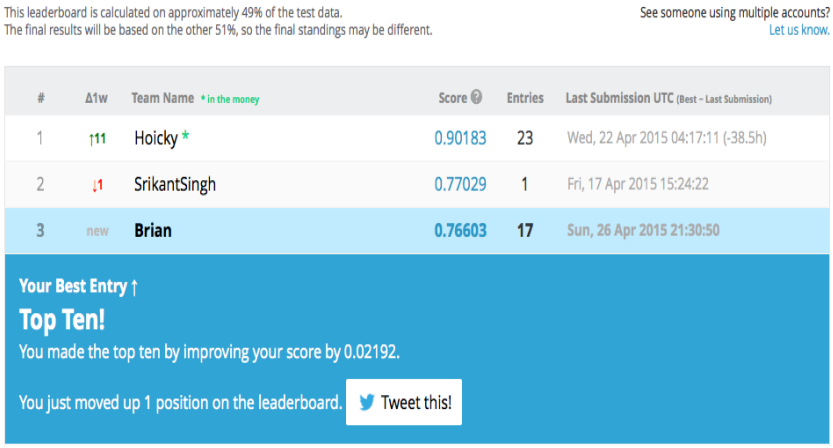



Figure 3: Best Ranking using SVM

# 5 Conclusions

Overall, SVM classification using RBF kernel and HOG features gives the best result among all models. Image processing and HOG feature are also key steps to increase prediction accuracy. By examine the confusion matrix, we noticed that most of the classification errors were due to misclassification of characters with similar shape like "1"(one) and "l" (lower case L), or confusion between lower and upper case letters such as "s" and "S", "c" and "C". Table 5 gives the top five misclassified errors and their share of classification errors. Most of these pairs were even impossible to distinguish by human eyes. We could further increase our prediction accuracy if the characters were given in their original context. In the future, we would also like to try to use the popular Neural Network Model for this classification problem.

| True Value | Prediction | Misclassification Rate |
|---|---|---|
| o | O | 0.0121 |
| s | S | 0.0110 |
| i | I | 0.0105 |
| 0(zero) | O | 0.0080 |
| l | I | 0.0080 |

Table 5: Top 5 Misclassification from Random Forest

# References


[1] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem, pages 77-87, IEEE Computer Society Press, 1994.

[2] L. Breiman. Random Forests Machine Learning 45.1 (2001): 5-32.

[3] T. E. de Campos, B. R. Babu and M. Varma. Character recognition in natural images, Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal, February 2009.

[4] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In CVPR, pages 886-893, 2005.

[5] T. Hastie, R. Tibshirani,and J. H. Friedman. Random Forests. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2009. 587-89.

[6] S. Knerr, L. Personnaz, G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman, editor, Neurocomputing: Algorithm, Architectures and Applications. Springer-Velag, 1990.

[7] Ulrich H.-G. Kreel. Pairwise classification and support vector machines, Advances in kernel methods: support vector learning, MIT Press, Cambridge, MA, 1999.

[8] Nobuyuki Otsu. A threshold selection method from gray-level histograms. IEEE Trans. Sys., Man., Cyber. 9 (1): 6266, 1979

[9] L. Tandalla First Steps With Julia. (n.d.). Retrieved April 26, 2015, from https://www.kaggle.com/c/street-view-getting-started-with-julia/details/julia-tutorial.

[10] H. Zhou, http://hua-zhou.github.io/teaching/st758-2014fall/ST758-2014-HW6.pdf.