# House Number Recognition from Street View Images

**Kevin Yan and Pratik Patil**
Department of Statistics
{kevin.yan, pratik.patil}@mail.utoronto.ca

## Abstract

This project looks at the problem of recognizing house numbers from the photos of house fronts taken from the Street View House Numbers (SVHN) dataset. The SVHN dataset is a collection of real photos of house fronts collected from Google Street View. Although the problem of recognizing digits in document images has been well studied in the literature, recognizing text and numbers in natural images is much harder. Text in photos suffer from the effects of blur, distortion, style and font variations, among others. In order to tackle this problem, we first attack the problem of classifying individual digits on well cropped images of individual digits from the original photos. We experiment with feature extraction techniques based on both hand-crafted features as well as those based on unsupervised learning methods. Specifically we examine feature extraction based on Histogram of Oriented Gradient (HOG) descriptors and that based on K-means clustering. Several preprocessing methods are explored to clean up the dataset first. Different classification algorithms are then applied to the features obtained from the above methods. We observe that the unsupervised feature learning based on K-means gives much better overall results compared with the features obtained from HOG descriptors. This might be because the HOG features are specialized for human detection as they come from computer vision domain with somewhat different application, and are not robust enough to variations that are present in the SVHN dataset. On the other hand, unsupervised features are learnt from the dataset itself and thus are made robust by the learning procedure. Though not experimented, we suggest ways to attempt the problem of detecting digits from the whole photos towards the end to solve the general problem of house number recognition.

## 1 Introduction

Recognizing handwritten digits has received great attention from the machine learning community in recent years. The performance of some of the best classification algorithms for this task on real data sets, such as the MNIST data set [2], even outperform those of real human beings! But the task of recognizing digits from natural photos is much harder. Various effects such blur, distortion, style and font various make the problem challenging. In this project, we study the problem of recognizing house numbers from natural photos of houses fronts. Recently one such data set with 600,000 labeled digits cropped from Google Street View was introduced, called the Street View House Numbers (SVHN) dataset [2]. We perform our experiments on this dataset.

The overall problem of house number recognition can be broken into two parts. One is to detect individual digits in an image of a house number sequence and the other is to classify these digits into one of ten classes. We focus on the second task of classification. The SVHN dataset provides the data in two formats. Format 1 contains original, variable-resolution, color house-number images that have the complete house number. Format 2 contains cropped images of each individual digit, resized to a fixed resolution of $32 \times 32$ pixels, with the digit roughly in the center of the image. In this project we only focus on classification of digits in the images from Format 2. Towards the end, we provide some ways to attempt the problem of detecting digits to perform the complete task of house number recognition on the images from Format 1.

For the problem of classification of individual digits from the Format 2 images, the difficulty lies in getting the right features that can discriminate between different digit classes and which are robust to font and style changes, rotations, size variations, and artifacts of blur and other noises. In this project, we experiment with two different ways of getting features. One is where we obtain specialized hand-crafted features that are designed with particular task and dataset in mind. We examine one such feature extraction method of Histogram of Oriented Gradient (HOG) which was first proposed in [9] for the task of human detection. HOG descriptors have been popular in computer vision community for object detection in the last decade. Second feature extraction method that we study is based on unsupervised feature learning. Specifically we employ a method based on K-means clustering that was described in [4]. In addition, to gauge the difficulty of the problem, we also experiment with the raw features obtained from the images after performing some preprcessing steps to clean the dataset. Once the features are generated, we then train several classification algorithms on these features to get a predictive model to predict labels for unseen images. We experiment with margin based classifiers based on discriminative functions (support vector machines), probabilistic discriminative classifiers (logitistic regression), and probabilistic generative models (naive Bayes). Overall, in this project we evaluate the performance of these standard classification algorithms as we progressively increase the complexity of feature extraction. We move from using the raw images, to adding some preprocessing, to using hand-crafted features of HOG, and finally features learned using an unsupervised algorithm based on K-means clustering algorithm.

We observe that features obtained from unsupervised learning perform much better than those obtain from hand-crafted features of HOG. While the best performance from the former reaches up to about 88%, the later only goes up to 78%, thus about suboptimal by 10% in terms of accuracy. We also observe that without using any sophisticated feature learning, the performance obtained from only the raw features does improve by some preprocessing, but the improvement is not much. This sugguests that features obtained from non-linear processing of the origianl raw feautures are indeed necessary to attain a good performance. Among the classifiers, support vectors machines generally outperform rest of the two classifiers, but however, suffer slightly when there are not enough good features.

This report is oranized as follows. Section 2 talks about some related work on this dataset. In Section 3, we formally define the problem. Section 4 first describes the framework for the classification task and then various models that we have experimented with. In Section 5, we give summary of our results. Section 6 concludes the report with a discussion of various possible extensions, and other models that can be of interest. We also make a few comments about how solving this problem might be useful in general text recognition in natural scene photos.

## 2 Related work

Most recent work on the SVHN data set consists mainly of deep convolutional neural networks. In fact, all the best models on this task are some variation of a deep convolutional neural network. In general, a convolutional neural network will learn features from patches of the original image, and use these learned features to create a new representation of the original image through a process called convolution. Additionally, pooling of this new representation may be done in order to reduce dimensionality of the features, as well as to introduce some variational independence into the model. These two different layers can be repeated multiple times in succession. However, the main idea is to train a neural network that is able to extract and use noise and translation invariant features for the use in tasks such as classification. A popular variation is the use of dropout networks, which are trained by randomly dropping out certain units of the neural network for each new training point, thus each training point will have an effect on only a subset of the parameters. This appears to work because, in a sense, it adds noise to the neural net and keeps the model from over-fitting to the training data, which is a common problem with neural nets [6]. In fact, a drop-out convolutional network has the second best score on the SVHN data with an error rate of 1.94% [7]. However in our paper, we do not discuss these models. Instead we focus more on the effect of preprocessing, and the use of different types of features, while using standard models like logistic regression and linear support vector machines for classification.

## 3 Problem description

We first describe the task of digit recognition on the dataset in Format 2. This format has fixed resolution $32 \times 32$ color images which contain a digit in the center. Informally the task in this case is to label an unseen image with one of the ten digits. More precisely, there are ten classes, one for each digit, $\{1, \ldots, 10\}$. Digit $i$ corresponds to class $i$, for $i = 1, \ldots, 9$. Digit 0 corresponds to the 10-th class. Given any unseen image, the system has to predict its class. To accomplish this, we have at our disposal a set of manually labeled images, where the ground truth is known. Each image has only a single correct label. This is the training data. We have a set of images which we want to predict the classes for. This is the testing data. In short, the task is a supervised multiclass single-label classification. The overall process in the learning system is shown in Figure 1. Some examples of the images from the training data are shown in Figure 2a and their corresponding true labels. Figure 2b similarly shows some examples from the testing data set along with their true labels. As can be seen from the figures the digits vary in size, location, shape (width, fonts), and they are also not always isolated, there may be other digits in the image. This task is similar to digit classification in the MNIST dataset [2]; however, there are much more variations in images, as mentioned above, that make it a much more difficult task than that of classification on MNIST.

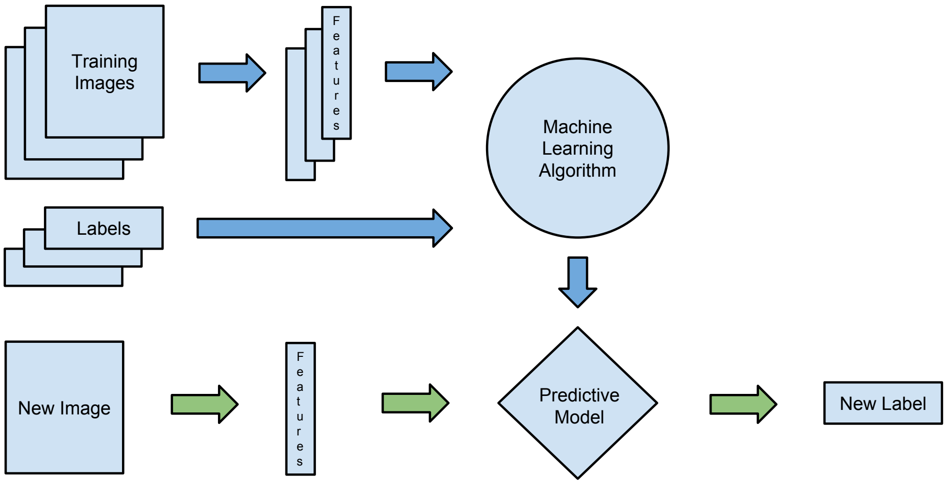
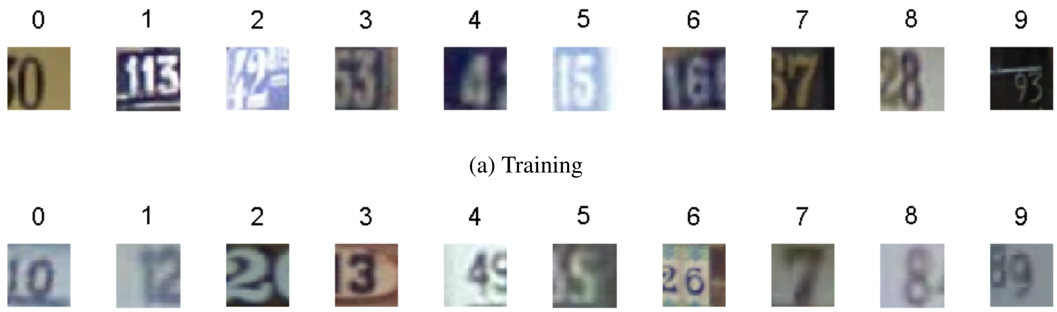


Figure 1: Pictorial representation of the digit classification process



(a) Training

(b) Testing

Figure 2: Training and testing examples for each class

# 4 Classification framework and models

We begin by describing the classification framework we have adopted as shown in Figure 3. After reading the image, we first preprocess it to remove some noise and to enhance certain features. Then features are extracted from the proprocessed image that are functions of the original pixels (which we call as raw features). Our goal in the feature extraction is to try to get features that distinguish the classes and which are robust to variations in the dataset. These features are then used by classification algorithms to build a predictive model which when applied to the features of the test images will produce predictions. The predictions are then compared with the true labels and the performance is evaluated based on how well the predictions match to the true labels. Below we describe each block in detail and various methods that we experimented on.

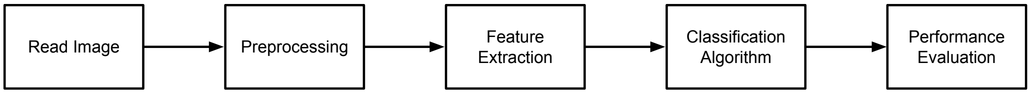


Figure 3: Digit classification framework

## 4.1 Preprocessing

To prepare images for use in feature extraction, the original colour images were first converted into gray-scale images. This is a reasonable pre-processing step because the colours do not offer much information in the recognition of digits. For digits, whats important are the actual shapes, edges and curves of the digits, as well as the contrast between the actual digits and the background in which they are printed. We then try to clean up the images. The 32 × 32 images in Format 2 of the dataset are actually prepared by using the original color house-number images in Format 1, with character level bounding boxes. These original character bounding boxes are then extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. This leads to some distracting digits to the sides of the central digit of interest. We first try to eliminate the side noise by first just cropping some side columns on both right and left sides. We observe most digits are in the center of the 32 × 32 box and by eliminating 4 columns on the both the sides we do not lose any information about the digit we want to classify. Although this step is not essential, because if these side pixels really do not provide any information about the digit, the classification algorithm will/should put small and zero weights on features obtained from those pixels. But we observe by eliminating these noisy pixels, we can reduce the feature length for HOG features and thus make the classifier run faster. For K-means based feature learning, we keep the original size images 32 × 32 as it is easier to work with square images in that case. In addition, in order to have a uniformity among the images and to improve on the contrast, we perform normalization. There are two ways in which we do normalization for images. One in which we rescale range of pixel intensities of images to a uniform range [10], and the other in which we standardize each image to have zero mean and unit variance [11]. This second method is done by removing the average pixel intensity of the image, and dividing by the standard deviation of pixel intensity of the image, for each pixel. By doing so, we will be able to see more contrast and detail in images that are uniformly dark, or uniformly light because of the scaling done by normalization. We will also compare these two normalizations. Additionally, whitening will also be performed on the image data prior to their use in the K-means algorithm, this is because the K-means algorithm does not handle large amounts of the correlation in the data well, and will generate centroids that lie close to each other rather than spanning the entire space [2]. Thus in order to remove some of the correlation in images, we can whiten the images, which removes the correlation between different features/pixels of the image. Moreover, specifically for HOG features, we also binarized the images so they they were strictly black and white. The reason for doing so is that HOG features outline the shapes found in an image, and this process would be greatly enhanced when provided black and white images. We compare two different thresholding methods for binarization, Otsu's method [12] and Sauvola method [13].

### 4.2 Feature extraction

Provided the preprocessed images, which are 32x32 matrices consisting of pixel intensities, there may possibly be better representations of this data. In fact, there is a lot of noise in the images, as well as some obstructions like extra digits, that could throw off the classifier. Thus, by identifying the useful characteristics of the image via feature extraction, we are hoping to find representations that provide key information on the digit present, while filtering out the noise. We will attempt to do so using both handcrafted features, HOG, as well as learned features using the K-Means algorithm.

#### 4.2.1 Hand-crafted features

Handcrafted features are obtained by carefully observing the dataset and defining sets of operations that are fine tuned to the dataset under consideration for the particular task at hand. These are very specialized features that are created specifically with the dataset and the desired ouput in mind. We experiment with one such feature representation method called Histogram of Oriented Gradients (HOG). HOG are popular feature descriptors that are used in the computer vision community for the purpose of object detection in an image. It was first described by Dalal and Triggs [9] for the problem of pedestrain detection in static images.

The basic idea behind the HOG descriptors is that local shape and appearance of an object can be described by the distribution of gradient directions (edges). In order to accomplish this, the algorithm divides the given image into small connected regions, called cells with some number of pixels in it. For each cell, histogram of gradient directions is complied for pixels within that cell. The number of histogram channels, or number of bins, are evenly spread between 0 to 180 (unsigned gradient) degrees or 0 to 360 (signed gradient). The combination of this histograms then represents the descriptor. To further robustness to changes in illumination and shadowing, the local histograms are contrast-normalized by calculating a measure of the intensity across a group of cells called a block, and this measure of intensity value will then be used to normalize cells within in each block. Notice that each cell will appear multiple times in the final descriptor, but normalized by a different set of neighboring cells. The number of feature vectors are then number of blocks multiplied by number of features in each block, which is just number of bins multiplied by number of cells per block. The process for HOG descriptors is shown in Figure 4. In our experiments, we use unsigned gradient with 9 bins (as done in [9]) with cell size of $2 \times 2$ and block size of $2 \times 2$ as we are interested in small scale spatial informations and we do not want to miss any small scale details.

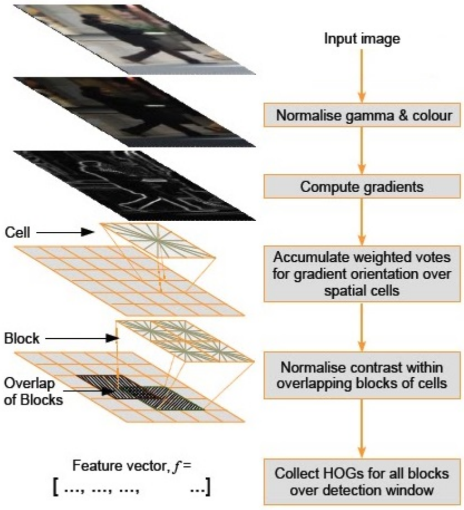

Input image
Normalise gamma & colour
Compute gradients
Accumulate weighted votes for gradient orientation over spatial cells
Normalise contrast within overlapping blocks of cells
Collect HOGs for all blocks over detection window
Cell
Block
Overlap of Blocks
Feature vector, $f$=
[ ..., ..., ..., ...]

Figure 4: Process for computing HOG descriptors (image adapted from slides of Dalal [14])

### 4.2.2 K-Means feature extraction

The second method for classifying the SVHN digits is going to be with a convolutional k-means filter, and a linear SVM. Initially we will use this k-means filter in order to extract features from the original image, and these features will then be passed onto a multi-class linear SVM in order to complete the classification.

To capture the features from the image, we first sample $W \times W$ patches from the original image; for this project we chose to sample 200,000 patches from the training data set. These patches are then fed into the K-Means algorithm in order to generate K cluster centres(centroids), which are learned without supervision, meaning no there was no feedback in the training process. These cluster centres will represent certain features that are found in size $W \times W$ patches (windows) of the SVHN images. In particular, the images end up detecting edges as can be seen in figure 9. These edges are very similar to features that are extracted by auto-encoders [5]. Extracting features from $W \times W$ sized patches of the image is done for a couple of reasons; firstly, its possible to reduce the number of parameters we have to learn when finding the reduced representation of the original image. Secondly, theres believed to be a stationary representation of images, meaning features collected by learning from one part of an image may be relevant in other parts of the image as well. The next step involves convolving the original image with size $W \times W$ windows of the learned centroids/features, which will lead to a new representation of the image, with a size of $(32 - W + 1)$ x $(32 - W + 1)$ x $K$, given that we use a stride size of 1. To expand, each step in the convolution (of which there will be $(32 - W + 1)^2$), will result in a $1 \times K$ vector, [ $f_1(x), ..., f_K(x)$ ], where each $f_k(x)$ = max { 0, average distance of patch $x$ to all centroids - distance of patch $x$ to the k-th centroid}. This can be considered a sort of soft-linear assignment, basically when the patch is too different from centroid k, then we will assign $f_k(x) = 0$, meaning it doesnt belong to the cluster. While if patch $x$ is relatively closer to centroid k compared to all other centroids, then we assign a positive value to indicate such relationship. Following the convolution, in order to reduce dimensionality for the SVM classifier, we will then perform pooling on this representation to end up with $4 \times K$ features for each original image. Pooling is done in order to reduce the dimensionality of the features, as well as to introduce some translational invariance in the model. Here we split the convolved image feature into four quadrants;we will then represent each quadrant using as a single feature (per channel) by taking the sum of all the values in that quadrant. Thus, pooling will result in $4 \times K$ features to be used in the multi-class linear SVM. The process can be seen clearly in figure 5 below.

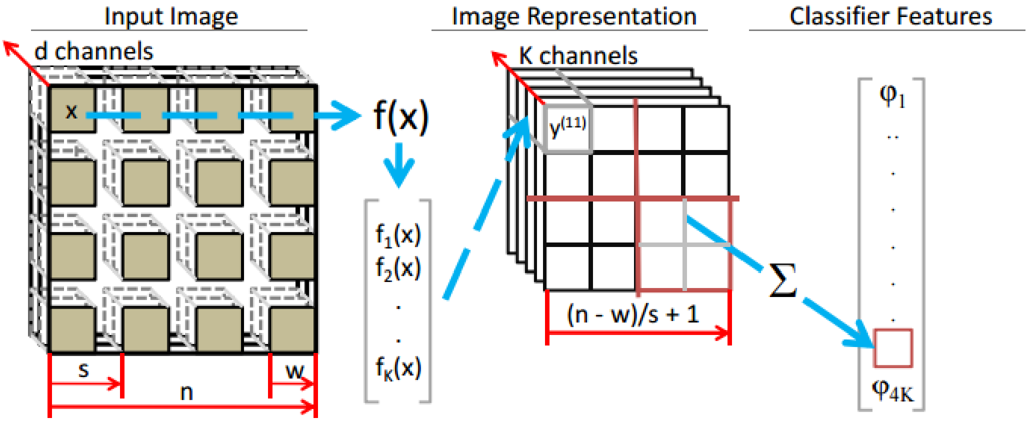


Figure 5: Visual representation of the process in which we extract features using the K-Means Algorithm (image taken from [4])

### 4.3 Classification algorithms and performance metric

In order to perform classification after we have extracted the features, we will mainly be using linear support vector machines, logistic regression, and to a lesser extent, gaussian naive bayes classifiers. However, the linear SVM will be our main classification algorithm. In all these classifiers, the decision boundaries are linear. However, they differ in the sense that GNB is a probabilistic generative classifier, logistic regression is a probabilistic discriminant classifier, and SVMs do not make any probabilistic assumptions on the variables. Thus this provides a good variety of possible classification models for which we can test our features on. In addition, because we need a multiclass SVMs for the SVHN problem, 10 SVMs will be trained, and the class for an image will be chosen using a one-vs-rest scheme (we will be using linearSVC from the sci-kit learn package for this). Finally, in terms of measuring the performance of each classifier, we will be looking at the training and testing accuracy, where the accuracy is the percentage of correct classifications out of all images considered.

## 5 Experiments and analysis

### 5.1 Effect of preprocessing

#### 5.1.1 Comparison of different normalizations and sharpening

We first compare between the normalization procedures of range expansion and standardization (zero-mean, standard deviation 1 images). Figure 6 shows effects on different preprocessing operations and their corresponding HOG descriptors. Comparing among the two normalization methods, we see that both of the methods increase the contrast of the image as dark points becomes darker and white points becomes whiter. The effect of contrast is more prominent in the standardization type of normalization. It seems that in the standardization approach, points near dark pixels get more darker as compared to stretching. But this introduces more connections and in this case the HOG boundary is not as clear compared with normalization that does range expansion. Sharpening seems to not improve the overall effectiveness, but not by much. Lastly as can be seen from the Figure, performing the HOG operation on binarized images results in more informative features than that with gray-scale or color images. This is because with binarized images, the curves are better outlined.

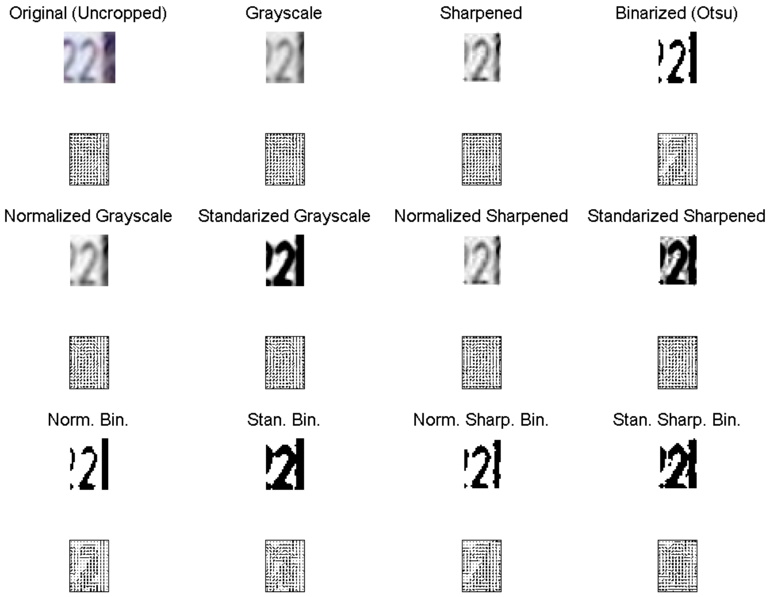


Figure 6: Comparison of the two normalization methods and effect of binarization

### 5.1.2 Comparison of different thresholding for binarization

Next we experiment with different thresholding methods for binarization. One is based on Otsu's method [12]. The other one is an adaptive thresholding technique, called Sauvola's method [13]. We observe that Otsu's method works much better than Sauvola. An example, a comparison of the two methods is show in 7. Binarization based on Sauvola is much more sparse than that with Otsu, and sometimes we lose important information as seen in the second row of the figure. Thus, we will choose to use this type of binarization on our images prior to HOG feature extraction.

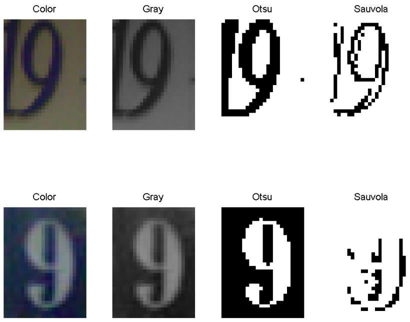


Figure 7: Effects of binarization based on Otsu's method and Sauvola method

### 5.1.3 Experiments on border noise removal

Finally we report some of our experiments with border noise removal. We tried using image processing techniques to attempt to remove the side digits by removing connected structures to the border that are lighter than the background. It worked really well on some of the examples, but unfortunately removed completely all the digits in some cases. For example, as shown in Figure 8, the top row show perfect removal of the side digit, but on the bottom we have an example where the main digit 3 is connected to the side digit and thus the algorithm deleted that digits as well. More sophisticated methods could tried where we could detect the middle digit and mask it so that it does not get deleted but we decided to not pursue those directions. It is because if we can get good features, our classifier will automatically learn to ignore the side noise. So rather than relying on manual image processing, we decided to focus more on machine learning. Let our data guide us! And we have lots it!

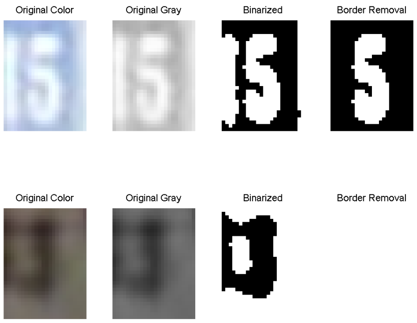


Figure 8: Image processing experiments to remove side noise

### 5.2 Raw feautures

In this section we perform classification using only the grayscaled versions of the original images as input. In addition, we also normalized these grayscale images, and compared classification performance between both forms of input using algorithms mentioned in section 4.4. In general we will notice that performance using the normalized version of the images is always better than the un-normalized version, thus presenting evidence for the use of normalization in digit recognition tasks. The use of normalization is to enhance the contrast present in images, and as we can see from figure 2, some of the images tend to blend in with their backgrounds, thus making normalization valid on the SVHN images. Also, overall it appears that the logistic regression model has some of the best results on the classification task with these features. It edges out both SVMs and perhaps hinting that including some uncertainy/probabilities in the model may be correct for digit classification. Moreover, we can also attribute some of the success of LR over SVMs to the use of discriminant functions in LR via the softmax, compared to the one-vs-rest system employed by multi-class SVMs.

| | Gray-scale | | Normalized Gray-scale | |
|---|---|---|---|---|
| Method | Training Accuracy | Test Accuracy | Training Accuracy | Test Accuracy |
| Gaussian NB | 12.389% | 12.465% | 32.529% | 34.93% |
| Logistic Regression | 27.647% | 23.383% | 30.69% | 25.626% |
| SVM C=0.1 | 26.76% | 22.43% | 28.80% | 24.52% |
| SVM C=1 | 24.2475% | 18.36% | 18% | 16.21% |

Table 1: Accuracy of both normalized, and un-normalized grayscale images using multiple classifiers

### 5.3 Hand-crafted features

We used 3 different sets of data in order to generate the HOG features which were normalized gray-scale images, black and white images, as well as normalized black and white images (normalized prior to binarization). Only normalized gray-scale images were considered because of their performance in the previous section. Note that we used all images in the training set for training, and all images in the test set for testing. Additionally we used a linear SVM in order to do classification, and varied over values of C (penalty for error) in this experiment. Overall, we see that using HOG features generated from black and white images result in higher test accuracy than the use of gray-scale images. Black and white images are usually better able to isolate and outline the digit, which really helps the HOG feature extractor focus on the digit as seen in Figures 6 and 7. Additionally, there does not appear to be any difference in test accuracies when performing normalization before binarization; Although, the highest accuracy is 78.02% using normalized BW images, compared to the 77.99% using normal BW images, this doesn't seem to be too significant. Because we are binarizing the images, the threshold will also adjust with normalization, thus resulting in little to no difference between these two types of images. Finally, we notice that as the value of C for the SVM decreases (i.e. punishing errors less), we tend to get better test accuracies for all cases. This leads us to believe that HOG features generated aren't really that linearly separable, and may contain a lot of overlap. Also by reducing C, we end up with larger margins in our SVMs which allows for more confident answers from the one-vs-rest classifier. We also decided to train a logistic regression, multinomial naive bayes, as well as a Gaussian Naive Bayes classifer for on the HOG features generated from normalized black and white images. The testing accuracies are as follows: 77.25% for LR, 71.72% for GNB, and 71.63% for the MNB. The performances using these models fall below that of the use of the linear SVM, which is no surprise given the reasons we mentioned prior. Overall, it appears that HOG features provide a substantial boost in performance compared to the previous features we were using (raw gray-scale images) in the previous section.

| C | Training Accuracy | Test Accuracy |
|---|---|---|
| 0.01 | 85.33% | 75.43% |
| 0.1 | 87.12% | 73.75% |
| 1 | 87.72% | 72.86% |
| 10 | 89.016% | 70.255% |
| 100 | 89.54% | 68.277% |

Table 2: Grayscale Normalized HOG

| C | Training Accuracy | Testing Accuracy |
|---|---|---|
| 0.01 | 82.46% | 77.99% |
| 0.1 | 84.79% | 77.25% |
| 1 | 86.33% | 76.21% |
| 10 | 86.82% | 74.99% |
| 100 | 86.89% | 74.58% |

Table 3: BW Unnormalized HOG

| C | Training Accuracy | Test Accuracy |
|---|---|---|
| 0.01 | 82.45% | 78.02% |
| 0.1 | 84.74% | 77.24% |
| 1 | 86.33% | 75.95% |
| 10 | 86.72% | 74.99% |
| 100 | 86.82% | 74.58% |

Table 4: BW Normalized HOG

## 5.4 Learned features

### 5.4.1 Dataset preparation

First, both a training and validation set were created for cross validation. The training set consisted of 45000 data points of which two thirds were randomly chosen from the original training set and one-third was randomly chosen from the extra set. The validation set consisted of 6000 data points as suggested by LeCun [8], which were chosen randomly with the same 2:1 ratio between the original and extra sets.

### 5.4.2 Results and analysis

Using data we then iterated through window sizes of {4,6,8} and cluster sizes of {250,500,750}, for a total of 9 combinations for use in cross validation. The penalty of the linear SVM was set at 100, and the stride was set to 1 for the cross validation process. The best results came from W=8, and K=750, the training accuracy and validation accuracy (for the CV data), were 92.6956%, and 90.10% respectively. These settings were then used on the actual training and test data set, and we received a training accuracy and testing accuracy of 90.656183%, 88.241395% respectively. As suggested by Coates et al., its generally possible to increase the performance of the model by increasing the number of centroids (K). By increasing the number of centroids to 1000, we were able to achieve 91.487503% accuracy on the training set, and 88.594806% on the test set. Through cross-validation we were able to see that as we increased the window size up to 8, we were able to see better performance on the validation set. This seems to indicate that larger windows sizes up to a certain extent will allow us to capture more complex features in the image [4]. For example, if we have small window sizes we may be missing certain characteristics of digits, like the arches and curves found in 2s,8s that can be captured when we use larger patches. The features that were extracted by our model with K=1000, and W=8 are shown in figure 9, where we are able to see many edges and curves that would correspond to parts of the digits.

It also appears that as we increase the number of centroids, performance on the validation set increases as well. By increasing the number of centroids, we are intuitively looking for more features in the images, and thus were better able to capture certain nuances that would be missed with fewer

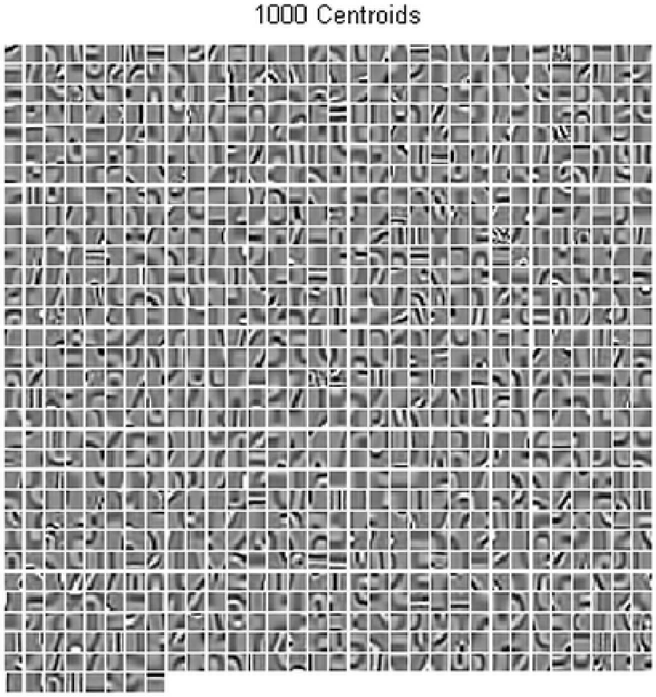


Figure 9: Images of the 1000 centroids discovered by K-Means with Window Size=8

centroids; as an extreme, having 2 centroids for example, would only allow us to capture very general features that apply to the majority of digits. Through further experiments, we were also able to determine that the higher the penalty term for our SVM was, the better the model performed on the test set (note: this analysis is based on peformance on the original training and test data). This experiment was conducted using a window size of 8 and a cluster size of 500 for speed purposes. It appeared that as we increased C, the performance consistently increased from 71.16% accuracy when C=0.1, to 88.11% when C=200. These results may be possible because the features learned by the model may be relatively good, and hence the feature space may separable to a certain extent, and as we increase the cost of penalties, we get closer to this boundary. In fact, we tried increasing C from 100 to 250, on a model with 1000 centroids, and saw the same increase in performance, this time from 88.594806% to 88.875230%. This seems to suggest the representations discovered are quite good at representing the original image.

| W | K | Training Accuracy | Test Accuracy |
|---|---|---|---|
| 4 | 250 | 81.133% | 79.95% |
| 4 | 500 | 83.922% | 82.35% |
| 4 | 750 | 85.2667% | 83.2833% |
| 6 | 250 | 86.5822% | 85.75% |
| 6 | 500 | 89.24% | 87.05% |
| 6 | 750 | 90.42% | 88.133% |
| 8 | 250 | 89.38% | 88.233% |
| 8 | 500 | 91.5289% | 89.6167% |
| 8 | 750 | 92.6956% | 90.1% |

Table 5: Results for Cross-Validation

Figure 11 contains confusion matrices for both the training and test data. A confusion matrix indicates how often digits were correctly, and incorrectly classified, in addition we can also see which digits an image may've been mistaken for. We see that the confusion matrices for both training and testing data are fairly similar, with very slight deviations. These results confirm that the features generated by K-Means are relevant for both the observed digits, as well as new, unobserved digits.

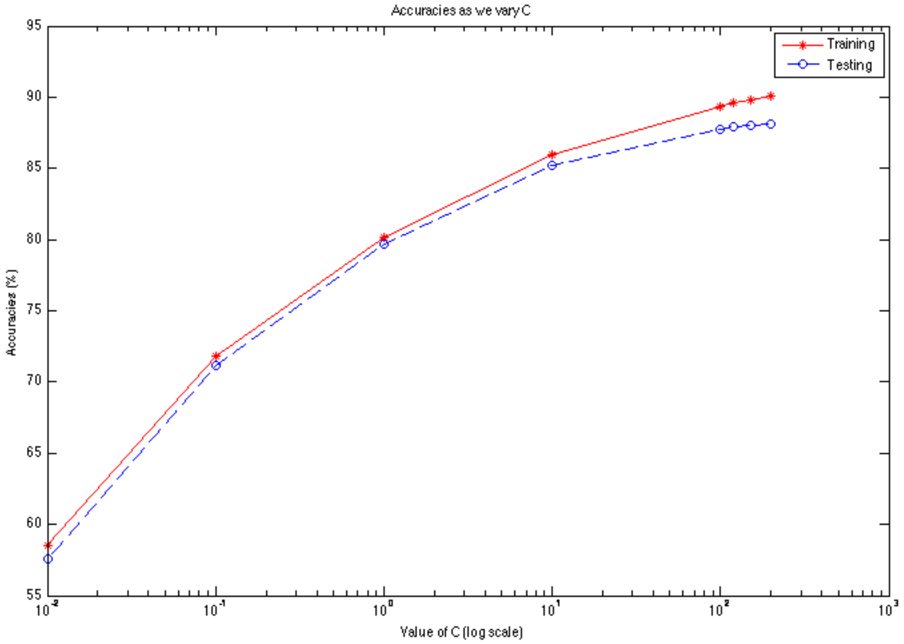
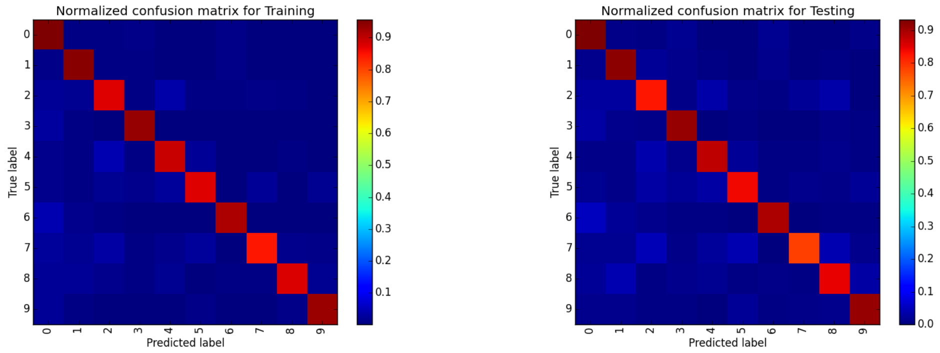


Figure 10: The effect of accuracy with varying C, where W=8, K=500

Thus, the algorithm was able to extract useful and relevant features from the digits, which can also be confirmed by the training and test scores mentioned previously which are in general, very similar.



(a) Training Data

(b) Test Data

Figure 11: Confusion Matrices

## 6 Conclusion and discussion

For the majority of the project, we only considered the use of the training and test sets from the SVHN data set. However, given that they included the extra data, we could've randomly split the extra set in order to generate training and test sets that involved all digit images collected. As the authors of SVHN indicated, the extra set contained much more easier examples, with fewer obstructions. Therefore it may've been beneficial to have more of these in training set so that we could capture better features from the digits themselves, instead of perhaps capturing noise in the images with our features. Additionally if we were to use much more data, it may make more sense to train the K-Means clustering algorithm in an on-line fashion. Also, we may also decide to move to logistic regression as our main classifier as it can benefit from online-SGD when we're using a larger data set. It may've also been viable to train a lot of weak learners using the large data set available to us, and used an ensemble of them to make digit classifications. For the weak learners the NB classifiers can be considered because of the speed of which they can be trained, and each will be trained on a different subset of available training and extra data. This idea may be considered, and experimented with in the future because of the performance enhancements that can be offered via these ensemble methods.

With respect to the results in project, overall we saw as we increased the complexity in our feature extraction methods, that we received better accuracies on the test set. Initially, we saw rather poor performances when we used the the raw gray-scale images as inputs. We were then able to achieve mid-70 accuracies using variations of HOG features. Finally, using a K-Means, we were able to get about 88% accuracy on the test set. It appears that learning features from smaller patches of the original image,and using the features to create new representations through convolution has had a significant impact on the ability for the basic SVM classifier, which was used throughout the paper, to perform much better classifications. This idea of feature extraction is in fact incredibly powerful, and is basically the backbone for all the top models used for the SVHN classification task. Thus, it would be interesting to explore the use of convolutional neural nets for this classification problem because they both expand on the use of this idea of feature extraction, as well as excel at it in a state-of-the-art manner exceeding that of human performance on the SVHN task. Additionally, it may be better to move in this direction when provided more data because these nets can be trained using SGD in an on-line fashion, which can take advantage of large data sets as not all of the data will be needed at once. We can also alternatively try the text detection and recognition framework of [15] based on Hough Forests that has shown better performance than K-means based method that was experimented in this project.

Finally a few comments regarding the usefulness and applicability. In general, by studying this task of classification on SVHN, the community can discover better and more useful features, or featuring learning algorithms which can be generalized can possibly be generalized to any character or object recognition task. Under this new digit data set, we introduced obstructions and variabilities in the digits unseen in MNIST. In general, digits are a intuitive starting point for object/character recognition as their are only 10 classes; Thus, by developing good features and feature extraction techniques on the SVHN data set, we will ultimately aid the construction of algorithms and techniques for other object recognition tasks that may contain obstructions like that of SVHN.

## References

[1] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011) Reading digits in natural images with unsupervised feature learning. *NIPS workshop on deep learning and unsupervised feature learning* Vol. 2011, No. 2, p. 5. Granada, Spain.

[2] MNIST. http://yann.lecun.com/exdb/mnist/.

[3] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint*. arXiv:1312.6082.

[4] Coates, A., Ng, A. Y., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. *In International Conference on Artificial Intelligence and Statistics* (pp. 215-223).

[5] Coates, A., Ng, A. Y. (2012). Learning feature representations with k-means. *In Neural Networks: Tricks of the Trade* (pp. 561-580). Springer Berlin Heidelberg.

[6] Srivastava, N. (2013). Improving neural networks with dropout (Doctoral dissertation, University of Toronto).

[7] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., Fergus, R. (2013). Regularization of neural networks using dropconnect.*In Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (pp. 1058-1066).

[8] Sermanet, P., Chintala, S., LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. *In Pattern Recognition (ICPR), 2012 21st International Conference* on (pp. 3288-3291). IEEE.

[9] Dalal, N., and Triggs, B. (2005). Histograms of oriented gradients for human detection. *In Computer Vision and Pattern Recognition*, 2005. CVPR 2005. IEEE Computer Society Conference on (Vol. 1, pp. 886-893). IEEE.

[10] http://en.wikipedia.org/wiki/Normalization_%28image_processing%29

[11] http://ufldl.stanford.edu/wiki/index.php/Data_Preprocessing#Natural_Greyscale_Images

[12] Otsu, N. (1975). A threshold selection method from gray-level histograms. *Automatica*, 11(285-296), 23-27.

[13] Shafait, F., Keysers, D., and Breuel, T. M. (2008). Efficient implementation of local adaptive thresholding techniques using integral images. *In Electronic Imaging 2008* (pp. 681510-681510). International Society for Optics and Photonics.

[14] pascallin.ecs.soton.ac.uk/challenges/VOC/voc2006/slides/dalal.ppt

[15] Yildirim, G., Achanta, R., and Ssstrunk, S. (2013). Text Recognition in Natural Images using Multiclass Hough Forests. *In VISAPP (1)* (pp. 737-741).