Handwritten Digit Classification

Ian Forbes, Xinchi Chen, Niladri Mohanty

November 4, 2014

1 abstract

We attempt to classify images of handwritten digits written on various textured backgrounds. Using a training set of 50000 images we evaluate the performance of several popular machine learning algorithms.

2 Introduction

Recently, character recognition technology is used in document analysis and recognition community doe to increasing demand of converting an enormous amount of printed or handwritten information to a digital format. Application of character recognition is converting historical, technical and economic printed documents into digital form. There are some successful implementation of optical character recognition (OCR) in digitization of handwritten documents. This field is quite challenging due to the high variability produced by noises, handwriting styles and image quality. Real world applications like bank cheque processing counts for handwriting variability. Also new handwritten digit dataset "CVL dataset" was released [?].

A fast and effective character recognition system required to solve the handwriting recognition problems such as bank cheque, automatic form processing or postal mail sorting [?]. It is necessary to highlight that not only high speed but an accurate system is needed for classification process in real time environments. The main problem in recognizing character is variability. The same class character can be written in different sizes and orientation angles.

For improving the performance in recognition problems, image is divided into local regions. This method is known as image zoning and highly successful in computer vision field [?].

Convolutional algorithms give best accuracy (99.73%) for MNIST database [?]. Ciseran et al. expanded the training and testing database, including elastic distortions. Studies proved that negligence of distortion results into low accuracy rates. Also, it is proved that classifiers which use only digit characteristics not the full image perform poorly [?]. The rest of this document is organized as follows: in section 2, we describe in detail different preprocessing approaches. Section 3 describes feature selection method. Section 4 describes different classification algorithm. Section 5 discusses the experimental evaluation.

3 Preprocessing

The most basic method to reduce noise from any signal is to average it across lot of samples. We tried to average all images for the same class to get rid of noise. As the noise (texture) is random, it is not a great technique to reduce the noise.

We tried many different preprocessing methods in attempt to reduce images noise, sharpen the appearance of the target digit, and most importantly improve our algorithms' accuracy. The first and most basic of theses methods was to normalize all of the images to a $[0, 1]$ scale. The reason for this decision was that while browsing through the images we noticed that many were mostly gray with very little black or white. We believed that by normalizing the images there would be more contrast between the digit and the background which would make it easier to identify the digit.

While normalization proved promising in validation, increasing our validation accuracy by roughly 1%, our actual Kaggle submissions scored lower than the un-normalized submission.

Another method we tried was image sharpening. Image sharpening works by subtracting a blurred copy of the image from a scaled version of the original image. The final image is given by the formula:

$$\text{sharpened} = 1.5 \cdot \text{original} - 0.5 \cdot \text{blurred}$$

Unfortunately this method showed no improvement over the non-sharpened images.

We also tried edge detection via the Laplacian operator which is the sum of the second derivatives. In order to calculate the the Laplacian the original image is convolved with the kernel below.

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

This operation finds area of the image where image intensity changes rapidly. These areas are then assumed to be edges. The problem with this approach however is that the image data becomes very sparse as information

In additional to applying various image filters we also attempted to increase the amount amount of data available us by adding rotated copies of the original data to our dataset. Since the digits from the original test were rotated at random we believed this would increase the amount to useable data and therefore are test results.

4 Feature Selection

We also tried to remove noise by utilizing Fourier Transforms. FFT can help to provide new ways to do familiar processing such as enhancing brightness and contrast, blurring, sharpening and noise removal. Fourier Transforms is the key to remove noise (small features) from an image, while preserving the overall larger aspects of the image. Although after inverse FFT, images are more clearer than the raw images and digits are recognisable by us. But SVM gives very low accuracy on both validation and test dataset.

Another method we tried was image sharpening. Image sharpening works by subtracting a blurred copy of the image from a scaled version of the original image. In particular we tried subtracting

Median filter is used to filter the noise. Again, it did not yield any considerable differences than raw images. Therefore, we did not focused on this technique of preprocessing.

5 Feature Selection

Blockwise histograms of local orientations is used to recognize objects [**?**]. This is called as Pyramid Histogram of Oriented Gradients (PHOG). Each pixel in the image is assigned an orientation and magnitude based on the local gradient and histograms are constructed by aggregating the pixel responses within cells of various sizes. The input grayscale image is convolved with filters which respond to horizontal and vertical gradients from which the magnitude and orientation is computed. The orientation could be signed (0360) or unsigned (0180). The signed gradient distinguishes between black to white and white to black transitions which might be useful for digits. PHOG features are tested with Support Vector Machine (SVM) linear and gaussian kernel.

6 Algorithms

6.1 Naive Bayes

Naive Bayes is a basic machine learning algorithm based on Bayes rule that can be used for multiclass classification. It uses Bayes rule along with the assumption that all features are conditionally independent given the class. The decision rule for Bayes rule is given by:

$$\underset{c}{\operatorname{argmax}} \Pr(C = c) \prod_{i=0}^{n} \Pr(F_i = f_i|C = c)$$

We used a variant of Naive Bayes called Multinomial Naive Bayes. This method creates a Multinomial probability distribution for each feature given the class. In technical terms we estimate the distribution $\Pr(F_i = f_i|C = c)$ over each features $f \in F$ and each class $c \in C$ from our training data. Then we use the decision rule to find the class c which maximizes this probability when classifying a test example.

In order to model

6.2 Neural Network

6.3 Support Vector Machine

Support vector machines (SVMs) have been a promising tool for data classification. Its basic idea is to map data into a high dimensional space and find a separating hyperplane with the maximal margin. We already discussed different feature selection strategies. Combination of features and SVM is used to classify using SVM classifier [?]. In all cases, model selection was performed using a validation set. Linear SVM is used to classify test dataset. One vs all classification approach is used for multi class SVM classification. Validation and test dataset accuracy is shown in table 1. Features Validation accuracy Test accuracy PHOG features 14.63Table 1. PHOG features linear SVM result

Linear SVM fails to classify test dataset by PHOG features. It can be inferred that there is no linear separation between different classes. SVM can also perform efficiently for datasets which are not linearly separated. For non-linear classification, it computes a new feature for every given example depending on proximity to landmarks. It is important to note that these landmarks are chosen to be equal to $x_i, i = 1,, m$. Given x, the new features are computed as- $f_i = \text{similarity}(x_i, l_i)$

To test whether SVM is a good classifier for this kind of problem or not, we tried SVM with gaussian kernel. The gaussian kernel function is given below: $f_i = \exp\left((-x - l_i)/(2^2)\right)$ SVM classifier with gaussian kernel is tested with ranging from 10^-7 to 1 and C = 0.1 to 10^5 [?]. Gaussian kernel SVM results for different sigma and C is shown in table 2.

σ	C	Validation	Test Accuracy
10^-7	0.1	31.52%	26.78%
10^-5	10	32.61%	29.82%
10^-3	10^3	29.61%	26.59%
1	10^5	24.94%	17.29%

Table 2. PHOG features gaussian SVM result

The best accuracy by SVM is 32.61By one more way, we tried to test linear SVM classifier. We applied fast fourier transform and then inverse fast fourier transform on training images to remove noise which is already explained in feature selection section. Then we tested these features with SVM linear and gaussian kernel and results is shown in table 3.

Features	Validation	Test accuracy
Linear SVM	10.3%	8.36%
Gaussian SVM	13.96%	9.47%

Table 3. FFT/iFFT features SVM result

We also tested SVMs both kernels for raw data. The accuracy for validation and test datasets are shown in table 4.

Kernel	Validation accuracy	Test accuracy
Linear	42.06%	39.48%
Gaussian	40.24%	39.21%

6.4 Convolutional Neural Network

Convolution Neural Networks take inspiration from nature. In the 1960's Hubel & Weisel conducted a wealth research on cats and monkeys in order to try to understand the biological vision systems of the brain. Their research showed that the vision system consisted of a number of 'receptive fields' which were layered on top each other. They also showed that vision at least in the first steps of the vision system where highly localized. Convolutional Neural use this notion of locality when building their neurons. Where as Fully Connected NNs accept inputs from all other neurons CNNs attempt to model the biological vision system by taking locality into account in the first layers of the network. The locals layers are then feed into a FNN at the end of the network to compute the final output. This has allowed CNNs to achieve remarkably good results in the area of image recognition.

6.4.1 Methodology

In order to build our CNN we used the Caffe [**?**] CNN framework. Caffe allows one to create a CNN using Google Protocol Buffers, a structured data definition language similar to XML and JSON, to define the structure of the network including convolution, pooling, and fully connected layers as well as output layers such as loss, argmax, and accuracy. Using Caffe one simply has to define their NN no coding required. Caffe will then compile this the definition file and build the network from it.

We based our final network off of the LeNet architecture [**?**]. This architecture was recommend by the MNIST tutorial provided on Caffe website. Since our dataset is very similar to the MNIST dataset we believed this would be a good place to start.

In order to train and validate the performance of our our network we segmented the dataset into 5 parts. We used the first 4 segments to train network and the last segment to validate against. We did not perform cross validation as the cost of train the network was quite high.

6.4.2 Network Architecture

Using the MNIST architecture as a our starting point we managed achieved achieve 86.7% accuracy on the public Kaggle test set. By adding an additional convolutional and pooling layer we managed to increase this to 91.5%. Next we increased the initial learning rate from 0.1 to 0.5 (a learning rate of 1 lead to NaN errors) in hopes that the previous network had fallen into a local minimum. After increasing the initial learning rate to 0.5 we scored 92.1%.

Adding a fourth layer posed a slight problem as each convolutional + pooling layer reduces the original input of 48×48 by slightly more than half when using a 5×5 kernel. In fact the output of the 3 layer network using 5,5,5 for kernel was a single scalar value. Because of this a fourth layer could not added. In order to remedy this we had to reduce the kernel size, and therefore the rate of input size reduction, in the proceeding layers. In the end the 4 layer network used 5,5,3,3 for its kernel sizes. Unfortunately the 4 layer network proved to be significantly worse than the 3 layer network, only scoring in the 75% range. Because of this, and the fact that the extra layer increased the training time, we choose no to train any additional 4 layer networks.

6.4.3 GPU Training

In order to train our network we used a single nVidia GTX 860m GPU with 2GB of memory we were able to train and validate our networks to convergence in roughly 20 minutes. This allowed us to easily tune parameters and add remove layers from the network inorder to see their effect on the validation score.

Graphics Processing Units (GPUs) allow for massively parallel operations to be performed on matrices and vectors. Traditionally used for graphics processing and rendering GPUs have recently found use in the fields of image processing and recognition.

The main utility behind using GPUs is that they employ the use of thousands of lightweight threads that run in parallel. This allows for many operations on images to be processed at the same time. GPUs usually have on the order 10^4 threads.

In stark contrast to CPU programming GPU programming rarely uses common programming constructs such as loops and conditionals. nVidia®'s CUDA uses a system of threads, block, and grids to execute operations in parallel. In particular each block can be assigned a number threads to operate on it. Each block can be be a 1,2,or 3 dimensional matrix. Blocks are grouped together in grids. In order to manipulate the data users apply a kernel to each grid. In order of apply a convolution to an image would be mapped to a single block. A grid would consist of a batch of images. In the context of our digit recognition problem we could assign $48 \times 48 = 2304$ threads to each block corresponding to 1 thread per pixel. The kernel (function) we would apply to each block would be a very basic function that computes the weighted sum of a given pixel and its neighbours based on the convolution kernel (image).

7 Discussion

References

Appendix