CSC 2515 Project: SVHN

Didier Landry
Department of Electrical and Computer Engineering
University of Toronto
didier.landry@mail.utoronto.ca

Dustin Kut Moy Cheung
Department of Electrical and Computer Engineering
University of Toronto
dustin.kutmoycheung@mail.utoronto.ca

Abstract

Detecting and reading text from photographs is a challenging computer vision problem that has garnered a lot of work in recent years. Being able to accurately localize, and recognize arbitrary digits from natural images is hard due to the complex scenes in those images. In this report, we apply convolutional neural networks to learn a unique set of features optimized for this task, and discuss the evolution of our neural network topology to achieve a high accuracy on the validation set. We use over 500,000 labeled digits obtained from the SVHN[12] dataset for training. Moreover, we also describe our attempt to localize digits from unconstrained images by using image processing techniques and unsupervised learning.

1 Background

Convolutional neural networks[14] are biologically-inspired neural networks that use identical copies of the same neuron for training, allowing the network to have a large number of units while keeping the number of parameters of those neurons small. The convolutional layers are connected to pooling layers to reduce dimensionality. One popular choice is the max-pooling layer, which extracts the maximum of features over small blocks of the previous layer.

2D convolutional neural networks are used in computer vision to learn features for extracting information. The 2D layer will look at patches of images to generate features such as the detection of the presence of an edge, or a particular texture. Those 2D networks are often stacked together with fully-connected neural networks to form the LeNet family of models, which are extremely popular in computer vision.

Recently, convolutional neural networks have been applied with dropout training[15]. Training deep neural networks with a small training set generally leads to overfitting. Dropout is a technique used to prevent overfitting by randomly dropping out units in a neural network. Other popular methods to fix overfitting is to simply stop training when the performance on the validation set starts to get worse, and L2 regularization on the neurons' weights.

All of our training data is provided from the Street View House Numbers(SVHN)[12] dataset. It is a popular training set consisting of Street View images cropped to either show single digits (Format 2) and multiple digits (Format 1). Those images are extracted using a combination of automated algorithms and Amazon Mechanical Turk (AMT) framework.

2 Problem Description

The project consists of classifying digits from street view images. All the images in the Format 2 dataset have a fixed 32x32 resolution with a digit centered at the image. There are ten classes in total. The images show vast intra-class variations due to image distortions that happen in natural scene pictures. Factors that cause those image distortions include lighting, shadows, motion, and focus blurs.

The Format 1 dataset consists of images with different resolutions with multiple digits in each image. The images are not well cropped and not centered. The task is to first detect the digits with a bounding box, then classify each digit. The images in the Format 1 dataset also show the same image variations as in the Format2 dataset.

To facilitate the process of implementing and debugging the digit recognition, the Format 2 data was used to test the digit recognition neural network. The digit segmentation was treated as a different problem because it (obviously) degrades the quality of the input images fed to the neural network. However, the digit segmentation algorithm produces 32x32 images that could, then, be fed to the neural network for recognition.

3 Process

3.1 Digit Recognition

3.1.1 Frameworks Used

Throughout the project, we use the popular Theano project[5], together with Lasagne[3], numpy, scipy[16], and NoLearn[4] libraries. Most of our experiments are executed using an EC2 machine with a GPU to exploit the speed gains of training with the CUDA framework.

3.1.2 Basic Neural Network

Our first approach is to train a basic Neural Network with two hidden layers with the grayscale images of the train dataset. The nonlinearity activation function used for our hidden layers is the Rectifier[11], whereas the output layer uses a softmax layer. The grayscale images are normalized before being used as input. The output layer consists of 10 units, each representing a particular digit. The guessed digit from the Neural Network is determined by the unit with the largest value.

3.1.3 Convolutional Neural Network (CNN)

We also attempted to add convolutional neural networks to our basic neural network. We added two convolutional and max-pooling layers in between the inputs and the rectifier hidden layers.

3.1.4 Improvement to CNN

One of the issues that can be experienced when using deep neural networks is overfitting, where the neural networks overtrain on the training set, while the validation set gets worse. One of the methods to rectify overlearning is by the use of Dropout layers.

Another improvement to our Convolutional Neural Network is to employ pre-processing on our images to boost some features that could help our network train faster and better. One of the most used techniques is the Local Contrast Normalization, which is applied to each of the three color channels of the images.

3.1.5 Augmenting the training set

Another way to decrease overfitting in our neural network is to simply augment our training set. To achieve this, we use the extra set ($\sim$ 500k images) available from the SVHN dataset to augment our initial training set of $\sim$ 73k images. We also tried to increase our dataset by adding noise and slightly rotating our existing training set to produce more data.

3.2 Image Segmentation

The unsegmented data set contains images of houses' address plates at different levels of zoom of zoom and isolation. Moreover, the number of digits that a given image contains is also unknown. Consequently, separating each individual digit is far from trivial.

Here, the problems of segmentation and recognition were considered independently. That is, for simplicity, the recognition algorithm was developped using Format 2 data (32x32 images, single digits). The segmentation algorithm simply generates Format 2 data from the Format 1 data (png images, multiple digits).

3.2.1 Process

For this project, an algorithm loosely inspired on MATLAB's OCR example[1] was written. Unfortunately, all attempts to leverage machine learning techniques in this part of this algorithm has performed worse than hardcoding decisions (more on that in the results section).

3.2.2 MATLAB Algorithm

First, the images are preprocessed: they are resized to a height of 200 pixels while keeping the aspect ratio the same and converted to grayscale.

Figure 1: Original Image (2200.png in training set)

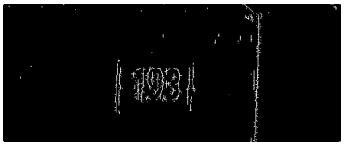

Figure 2: Vertical Edges (Sobel vert. edge detection)

Figure 3: Mask obtained by morphological closing, after filtering

The second step is to find the areas of the image that have a high probability of containing text (an approach similar to that of A. Coates et. al.'s "detection"[6]). For that, we apply a sobel vertical edge detection algorithm and, then, morphological closing on the found edges (which are very scattered) to create connected regions. These connected regions are then enclosed in the smallest box that can contain them completely. This worked well because the vertical boundaries in numbers tend to create a rectangular blob of points that, when morphologically closed, completely contains the digit (Figure 2 & 3). Vertical edges such as the wall, in this example, get filtered easily because, after closing, they create really narrow regions.

Depending on the image, this preliminary analysis can produce a lot of possibilities. To solve this problem, the candidate connected zones are filtered by size, aspect ratio and solidity. Areas that are too narrow, fat or have a low solidity (level of vertical gradients in a box containing the connected zone) are discarded. The zone that has the greatest probability of containing a digit is then passed to the next step. In other words, this second step finds the scale of the numbers in the image because it effectively does the vertical cropping.

The third step is to detect the number of digits present in the previously found area and split them (basically, do the horizontal cropping). For that, the image is turned into binary (black and white) using Otsus thresholding method[13, 2] . Then, after inverting the image if the background is white, the number of white pixels in each column (assumed to be the digit) is counted. If it falls below a threshold (that is a function of the maximum number of white pixels in the images column), a split point is saved (see Figure 4 & 5).

Finally, the split zones are, again, filtered by width and cropped into 32x32 pixels images (extending laterally to get a 1:1 aspect ratio, if necessary) to match the conventions of Format 2 (Figure 5 & 6).

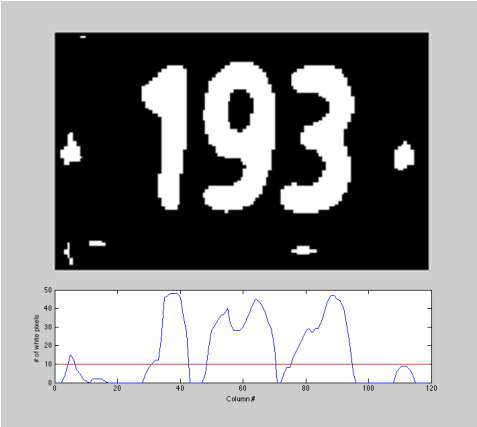

Figure 4: Digit splitting based on of white pixels per column. The red line is the threshold.

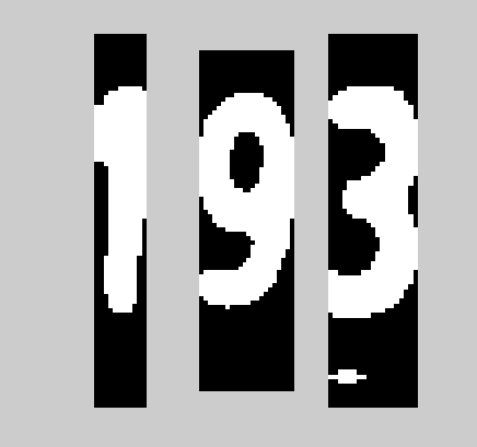

Figure 5: Split Zones. Leftmost zone was discarded because of its small width

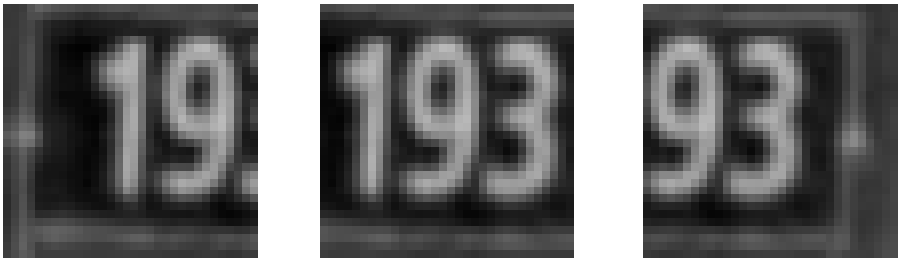

Figure 6: Final Output

3.2.3 Sliding Window Approach

We also attempted to use a sliding window approach, where a window of size 32x32 would move around the picture with a stride of 16 pixels. For each of these windows, we note whether there is a digit found inside the box or not.

After running the sliding window technique, each window is fed to a convolutional neural network with the output as a list indicating if that window has a digit in it, basically creating a binary classifier.

However, due to the huge number of ”non-digit” windows compared to ”have-a-digit” window, it proved to be hard to effectively train our network to effectively classify whether a window contains a digit or not.

Moreover we suffered from scaling window issues where sometimes the window size we chose was too big for a particular image and captures several digits, or the window size was too small and only a tiny section of the digit was obtained.

4 Experimental Results

4.1 Digit Recognition

As shown in Table 1, the greatest accuracy jump came from using convolutional layers in the neural network. Indeed, accuracy jumped by almost 10%, which was an even more dramatic effect than adding 500k extra training examples (6% improvement).

This is not a surprising result since convolutional layers are known to be good at image processing. The layers typically learn features such as contrast and edges. Strangely, however, adding a third or

Algorithm	Validation Accuracy %
NN with Regular Hidden layers (1 hidden layer, 300 units trained on complete training set)	79.43
NN with Convolutional Layers (2 convolutional layers, 24 and 32 filters and max-pooling 1 hidden layer, 300 units trained on complete training set)	89.37
NN with Convolutional Layers and Contrast Normalization (2 convolutional layers, 24 and 32 filters and max-pooling 1 hidden layer, 300 units trained on complete training set)	90.15
NN with Convolutional Layers and Contrast Normalization and Dropout (2 convolutional layers, 24 and 32 filters and max-pooling 1 hidden layer, 300 units, dropout layer trained on complete training set)	90.89

Table 1: Algorithm and Accuracy

fourth convolutional layer degraded the results. This might be because, given the size of the images, adding extra layers was superfluous.

On the other hand, adding contrast normalization had very little effect on the accuracy. Its possible that the lack of contrast in the original image was just compensated by some weights being higher (more sensitive contrast thresholds). When processing the images with improved contrast, the network would just have learned to be less sensitive while learning the same features. This makes sense; increasing contrast only boosts information already present in the image at the first place[10].

# of training examples	Accuracy %
73,257 (just the training set)	89.28
100,000	93.74
200,000	95.06
300,000	95.69
400,000	96.00
500,000	96.13
604,388 (all of training + all of extras)	96.50

Table 2: Accuracy vs Training Examples

The next thing that greatly improved the results was to augment the training set with some examples taken from the extras set. This confirms the general intuition that more data is better. Figure 7 and Table 2 show the accuracies obtained with different numbers of training examples after 10 epochs.

4.2 Image Segmentation

It is important to point out that the digit segmentation algorithm is mostly an engineered solution (in that it doesnt learn from the data); it just applies decisions based on hard-coded thresholds, gradients and masking.

This is because this engineered solution systematically performed better than the machine learning techniques we used and that any attempt to integrate machine learning algorithms degraded the results. Below are a few of the techniques that were tried but later abandoned.

4.2.1 Better Boundaries

An attempt to use a more advanced edge detection algorithm, P. Isola et. al.s crisp boundary detection[9], was made without success. This algorithm, by leveraging the mutual information given by the pixels in the image, performs really well when segmenting complex images. However, the

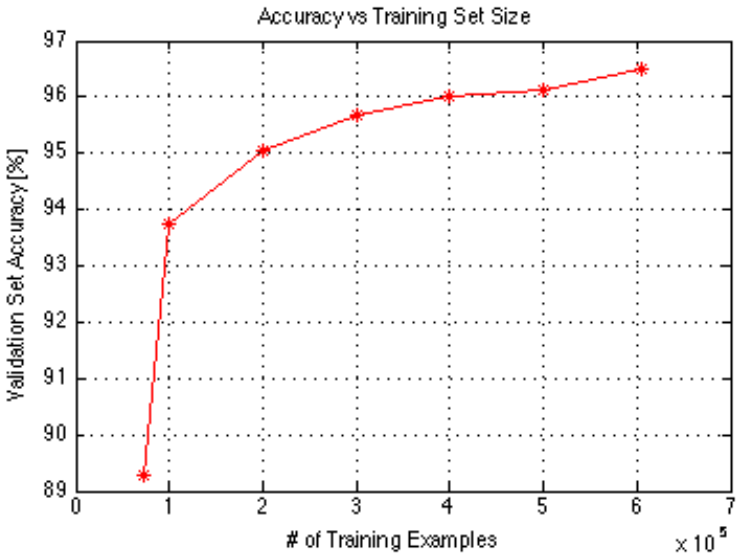

Figure 7: Accuracy vs. Training Examples. (2 convolutional layers, 24 and 32 filters and max-pooling 1 hidden layer, 300 units)

results obtained here, on simple lower resolution images were not usable for our purposed. Indeed, with low contrast pictures, it would detect tiles due to colour imperfections in the image files encoding while, in images where a strong colour contrast that is not due to the address digits is present (such as the edge of a wall), it would completely disregard the lower contrast changes due to the digits.

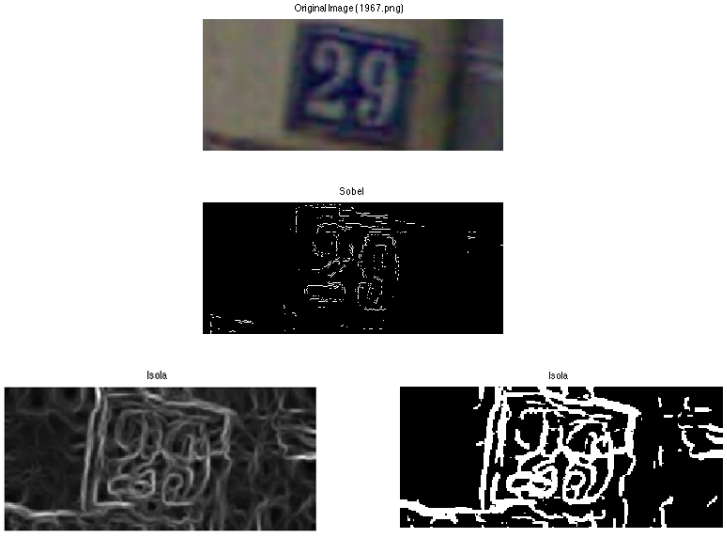

Figure 8: Sobel Thresholding vs. P.Isola's Crisp Boundary Detection Algorithm (before and after thresholding)

4.2.2 PCA for Segmentation

In order to make the separation of the digits easier, an attempt was made to use PCA to generate the histogram in Fig. 4 (and to rectify the image when the address was diagonal). This, for instance, would have made it easier to separate numbers that are placed in diagonal because it would have been rotated to an horizontal orientation (the current algorithm tends to fail at this task because of the vertical overlap between the numbers). However, this approach was abandoned because it also had major adverse effects, such as a tendency to flip the image vertically or horizontally that made it perform worse than a simple count of white pixels.

4.2.3 k-Means for Segmentation

Further, k-Means was tried as a way to locate the centers of the digits. The white pixel count was be used to guess a value for k and, then, the image was split using the centroids found by k-Means on the thresholded image. This caused more errors than a simple split however, because, even if the number of digits found was correct, the centroids were sometimes distributed vertically (i.e. two centroids on the same number) instead of all horizontally. Forcing the k-Means to be only horizontal produced slightly better results but was still not as good as a simple split because it was sensitive to noise (address plate edges) and weight distributions in the digits (a 5 typically has a centroid in its upper-left portion, for instance).

4.2.4 Results

The segmentation results are reported in Table 3. Please note that the accuracy is based on a success being defined as properly isolating and centering ALL the digits in an image (not more or less). For instance, the case where all the digits of an address were found properly where but the edge of the number plate was mistaken for an 1 would be considered a fail.

Also, since this definition of success is too variable for a computer to properly evaluate it, the successes had to be counted manually. This, unfortunately, introduces a bias in the numbers below as it limits the number of examples that can be used to compile the accuracy statistics.

Examples (treated with this algorithm)	Accuracy (hand-counted) %
100 examples from training set (2200 to 2300.png)	36
100 examples from extras set (3000 to 3100.png)	37

Table 3: Examples vs Accuracy for Image Segmentation

5 Discussion and Conclusion

5.1 Scaling with Millions of Examples

Scaling our current convolutional neural network to millions of examples is incredibly challenging. Luckily, in Lasagne, a lot of optimizations happen in the background. For instance, the batch size is selected automatically depending on the number of examples.

Even though Lasagne and Theano ultimately produces CUDA code that can be run on a GPU, there are further optimizations that could be achieved by using a pure CUDA approach and getting rid of Python dependencies along the way.

One way to scale our training is to simply use a distributed neural network spanning across several machines. With this amount of data, the overhead involved with communication between the different machines become negligible compared to the speed gain in training in parallel. An example of such a distributed network is DistBelief[7], developed at Google.

Another way is to simply decrease the pixels of each image used to a smaller size. By having less pixels to train on, the training speed will increase, albeit at the expense of losing some information.

5.2 Digit Recognition

The input representation of the images into the Convolutional Neural Network consists of grayscale images obtained from the original RGB images. In future work, this could further be improved by using the YUV representation of those images. YUV representations tend to be less sensitive to different lighting conditions and may help reduce the intra-class variations in the SVHN dataset.

The number of layers in our convolutional neural network are relatively small compared to what some of the current solutions are using. It will be interesting to see how our results are affected by deeper networks, more filters in our convolutional layer and even more training data.

Alternate pooling layers could also be explored. Some research points to the use of Lp-Pooling[14] as a superior pooling layer to the max-pooling we are currently using.

We could also explore the use of Maxout Networks[8] with our current network to see if it helps with delaying overfitting.

5.3 Image Segmentation

The image segmentation algorithm could be improved to allow for more than one zone of high probability to contain a digit. Currently, if the numbers of the address are too far apart in the image, they get boxed separately and, in the end, only one number gets isolated properly. This is because, after the morphological closing of the vertical gradients, all the numbers are assumed to be in the same connected zone (this was a simplifying assumption that proved to be correct in most cases).

It would also be interesting to continue to investigate to find machine learning techniques that could exploit some information that comes with the Format 1 images; the boxes. Indeed, each image comes with the location and size of boxes that surround the each digit. Perhaps, using the number of boxes, a classifier could be trained in a supervised manner to find the number of digits in each image.

References

[1] Automatically detect and recognize text in natural images. `http://www.mathworks.com/help/images/ref/graythresh.html`.

[2] Graythresh, matlab documentation. `http://www.mathworks.com/help/images/ref/graythresh.html`.

[3] Neural network tools for theano. `https://github.com/Lasagne/Lasagne`.

[4] scikit-learn compatible wrappers for neural net libraries. `http://pythonhosted.org/nolearn/`.

[5] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[6] Adam Coates, Blake Carpenter, Carl Case, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J Wu, and Andrew Y Ng. Text detection and character recognition in scene images with unsupervised feature learning. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 440–445. IEEE, 2011.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[8] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[9] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Crisp boundary detection using pointwise mutual information. In *Computer Vision–ECCV 2014*, pages 799–814. Springer, 2014.

[10] Yann Lecun. Lenet-5, convolutional neural networks. `http://yann.lecun.com/exdb/lenet/`.

[11] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[12] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5. Granada, Spain, 2011.

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

[14] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3288–3291. IEEE, 2012.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.