Multi-digit Number Recognition

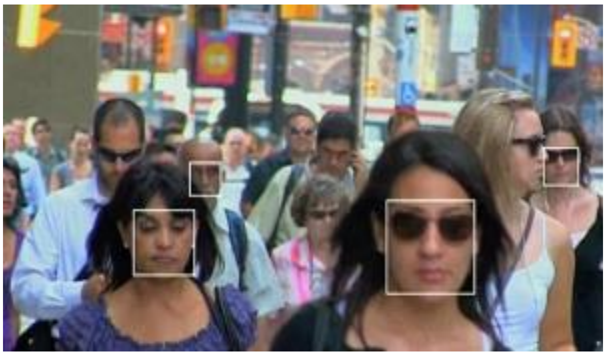
Machine Learning Nanodegree Capstone Project

Petr Shypila
September 18th, 2016

1 Definition:

1.1 Project overview

Nowadays object recognition on images is a very challenging area. While some problems like single character recognition are easy to solve, the others like recognizing particular person in a crowd or recognizing a sequence if characters still might be difficult. Last years huge amount of open data was published which pushed progress in this area forward.



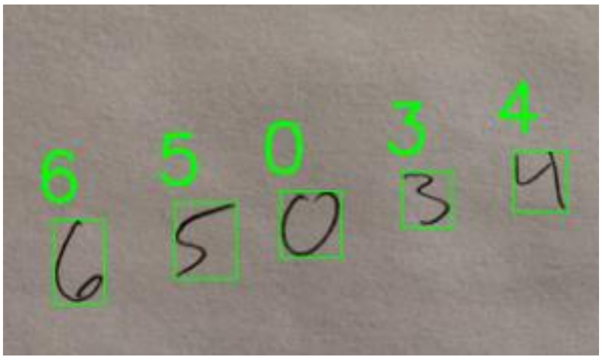


Figure 1 Actual Image Recognition Problems

This project focused on a number recognition problem, which is basically a sequence of symbols. In particular, we are going to implement algorithm which will recognize house number from images. Application will take as input a bunch of images and provide a house number which is presented on each image. One of the difficulties here might be that sometimes house numbers are written in some specific order. It might be written vertical or diagonal or something else. However, at the end solid application which recognizes numbers from images with high accuracy will be presented.

In this project we will use Street View House Numbers(SVHN)[1] dataset to train our application recognize numbers on images. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. The dataset is obtained from house numbers in Google Street View images. Image below represents several samples from this dataset.

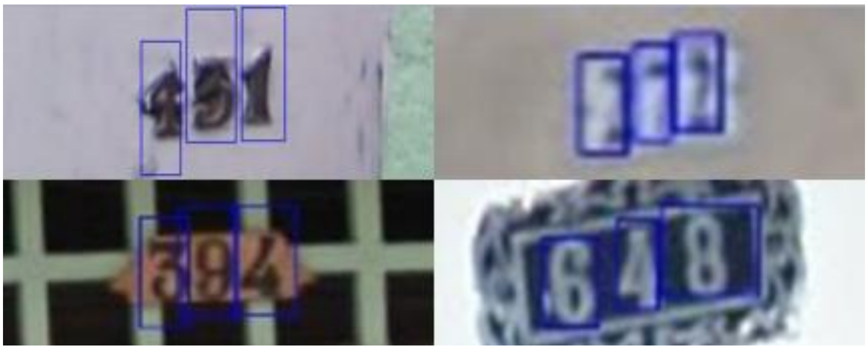


Figure 2 SVHN Dataset Samples

As you can see, some images, like top right have quite bad quality, so even a human sometimes is not able to recognize them. So basically some error can have a place. To get the error rate we are going to calculate relation of incorrectly recognized numbers to the whole amount of numbers to calculate.

1.2 Problem Statement

Recognizing numbers on images might be accomplished with human help if the amount of images to recognize is not big and there is no need to perform these calculations on a daily basis. However, if the amount of images is extremely large using people to solve this problem is not efficient. And this is the place when Machine Learning comes into play. To solve this problem, we are going to use modern deep learning techniques which are provided by the Google's Tensorflow library. In the first part of a project we develop a multilayer convolutional neural network which will be trained to predict just a single digit.

Convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing. They have wide applications in image and video recognition, recommender systems and natural language processing.

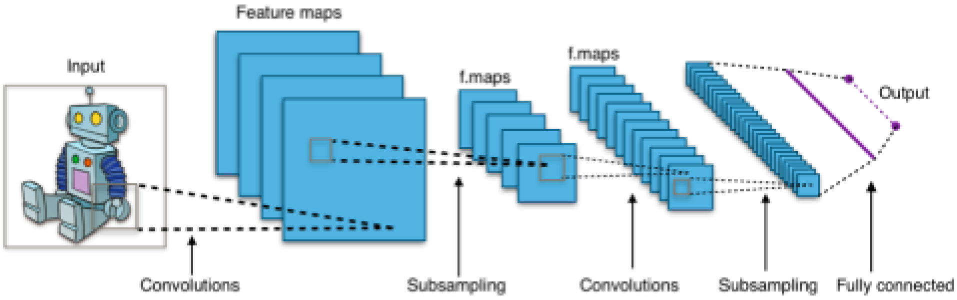


Figure 3 Typical Convolutional Network

A ConvNet model represents a stack of various layers that transform an input to the output. The most common types of layers are:

Convolutional Layer - The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The conv layer parameters consists of a set of learnable filters. And the network will intuitively learn which filters to activate when they see some type of visual feature. The input to the Conv layer usually called input feature map and output is an output feature map.

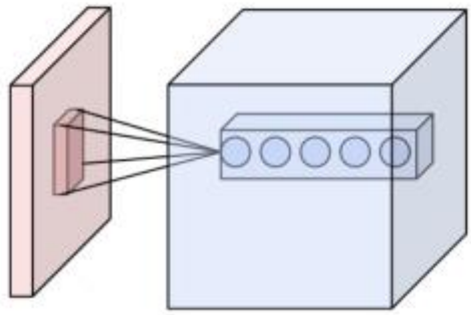


Figure 4 Example input volume(left) and and an example volume of neurons if first conv layer

Pooling Layer - progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, by using some function. In our application MAX operation will be used, as the most common one. Figure 5 illustrates a result of MAX function with 2x2 filter and stride = 2.

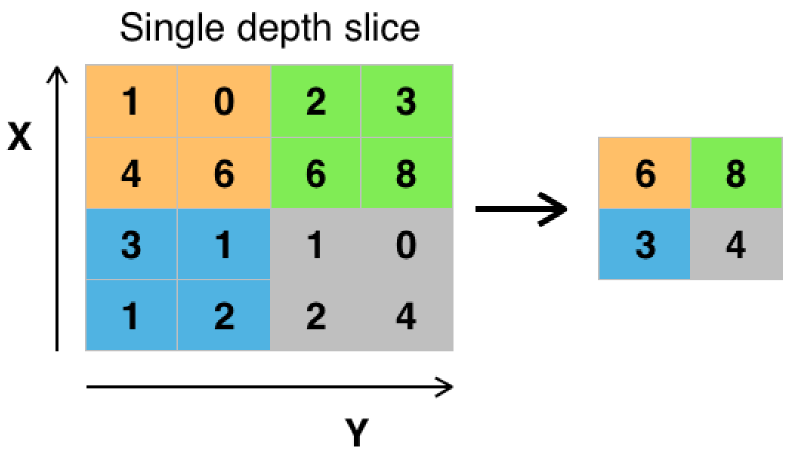


Figure 5 Max pooling with a 2x2 filter and stride = 2

Fully Connected Layer - Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. See the Neural Network section of the notes for more information.

Loss Layer - The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network. Various loss functions appropriate for different tasks may be used there. We will use Softmax loss function for predicting a single class of K mutually exclusive classes.

After basic version will be designed, we will tune network's parameters trying to find configuration with lowest error rate. After that we will adapt this model to a multi-digit problem. This network will be able to recognize numbers with up to five digits with slow error rate. It will be able to take an image with house number presented and return the number which is presented on an image.

2 Analysis

2.1 Data exploration

SVHN dataset consists of 3 groups. Training, testing and extra data. Training set has 33402 samples, training set has 13068 and extra set has about 230000 entries. Extra set will be used to extend our training set and perform validation during model training. Speaking about possible labels, basically there might be an infinite amount of labels, since there are no limitations about data presented. However, since we are going to work with images which have up to five digits between 1 and 99.999 inclusive, this is the amount of all possible labels. Samples which have numbers with more than 5 digits will be removed from the dataset during preprocessing step. Beside the images, dataset contains also a digitStruct structure which contains some meta information about each image. This structure has same length as amount of images in a dataset (one entry per image). Here each entry has two fields: **name** which is a string containing the filename of the corresponding image and **bbox** which is a struct array that contains the position, size and label of each digit bounding box in the image. For instance, *digitStruct(300).bbox(2).height* gives height of the 2nd digit bounding box in the 300th image.

2.2 Exploratory Visualization

Figure 5 illustrates class lengths distribution across several datasets. As we can see all datasets have Gaussian distributions. The relation of 1, 4 and 5 digits classes to the whole dataset almost equal for training, test and validation datasets. On the other hand, Test dataset has more classes with two digits while extra set has more classes with

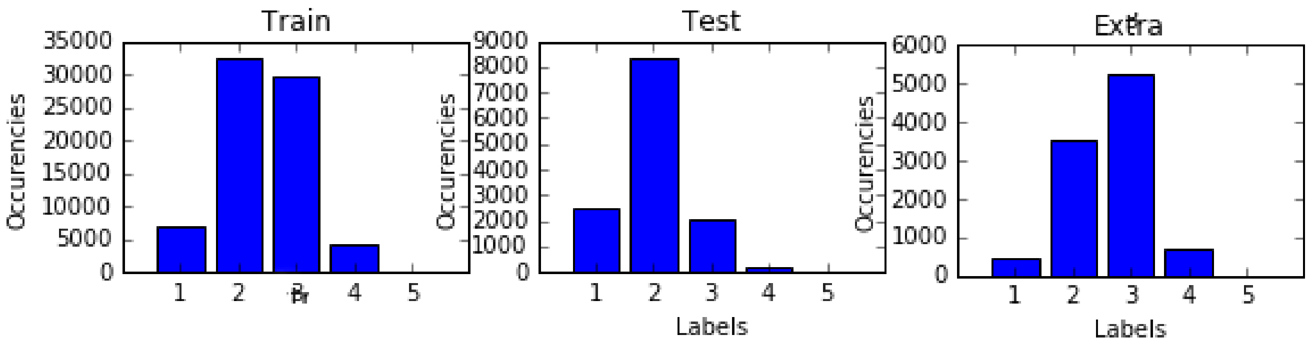


Figure 5 Number length distribution

2.3 Metrics

For this problem we calculate accuracy by counting how many digits were correctly recognized in a number. Then we sum up all this percentage and divide it by a size of a dataset. Here is the formula:

$$\frac{\sum\left(\frac{Correctly\ classified\ digits}{\#\ of\ digits\ in\ number}\right)}{Size\ of\ dataset}$$

3 Methodology

3.1 Data Preprocessing

During data preprocessing we implemented these steps:

1. We each image just near the numbers. By doing so we simplify work for neural network.
2. Then we scale cropped images to 32x32 size.
3. After that mean value was extracted from each pixel and divided by standard deviation.
4. We converted all input images into a grayscale. It is a very common technique in image preprocessing. It helps classifier to recognize digits because normally digit's color does not matter. To do that we need to multiply each pixel level by some multiplier. This approach described in details in this [2] paper. In general, we need to multiply red channel by a factor of 0.2989, green by 0.5870 and a blue one by 0.1140. Figure 8 provides several images which were processed by this algorithm:

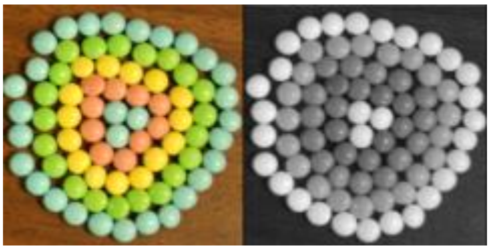


Figure 8 Grayscale algorithm result

5. After putting image into gray scale we apply local contrast normalization on images to improve recognition. More details about local contrast normalization could be found in this [3] paperwork.
6. '0' digit class replaced from 10 to 0 to be consistent with other classes. 10 class indicates that no digit for the layer presented.
7. Each sample class split by digit. If number has size less than 5 digits we add labels 10 at the end. It means that number 567 will have following set of subclasses: ['5', '6', '7','10','10'].

3.2 Implementation

As it was written previously, to recognize numbers on images we are going to build a convolutional neural network(ConvNet). It has been shown many times that ConvNets outperform many [4] other common approaches in image recognition problems.

For our problem we build a ConvNet which has a set of layers S of size N = 5 to recognize separately each digit in a number of maximum length N. On the output we receive a set of logits of size N which we pass to softmax function to get probabilities of belonging digit to some particular class.

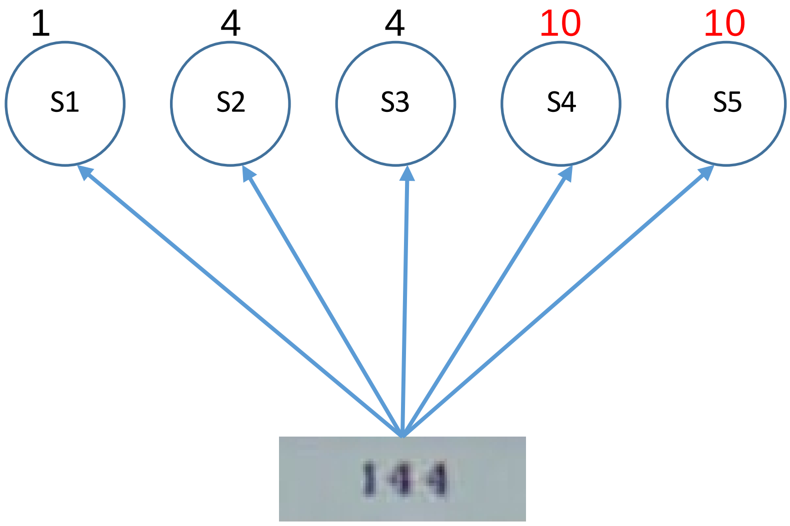


Figure 9 Number recognition architecture

Softmax function returns probability distribution across all classes which we use for calculating accuracy. We use same architecture to recognize all digits in a class. Figure 10 illustrates the architecture scheme

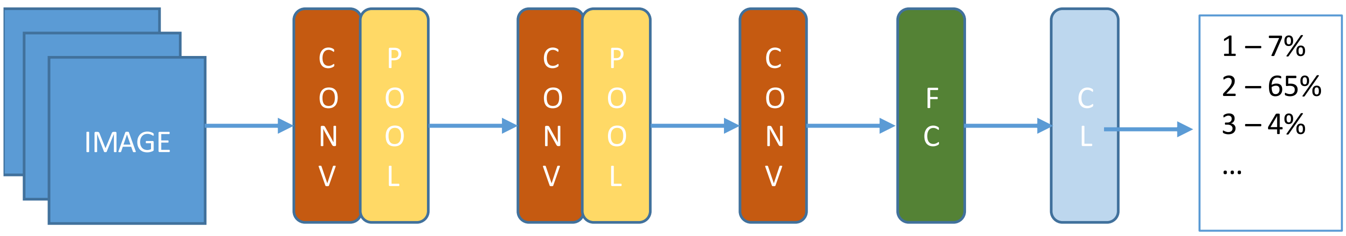


Figure 10 Single digit recognition ConvNet architecture

Here first ConvNet receives a batch of 32x32 images. First conv layer applies has convolution size 5 x 5 x 1 x 16. The second conv layer has convolution size of 5 x 5 x16 x 32. And the third one 5 x 5 x 16 x 32. During training we also apply dropout normalization and at the end we have one fully-connected layer with weight size 64 x 11.

During training phase, we use Adagrad optimization algorithm. Adagrad [5] is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. Adagrad greatly improved the robustness of SGD.

The final version of our model is able to correctly recognize numbers on images with accuracy equal 0.9. The figure 11 illustrates result produced by our trained model.

Input images

Output classes for those images

Figure 11 ConvNet input and output results

3.3 Optimization

During the work on a project I applied such techniques: Also I added local response normalization layer. Although it is labeled like fallen out of favor [8], in my case local normalization helped dramatically. Without local response normalization layer model was not able to learn anything. The next parameter I played with was dropout rate. In my case, dropout rate 0.9375 showed best results. With dropout rate 0.5 test accuracy was about 0.86, while with dropout rate 0.9375 test accuracy is about 0.9. Figure 11 illustrates minimization of loss value with dropout 0.5(left) and 0.9375(right).

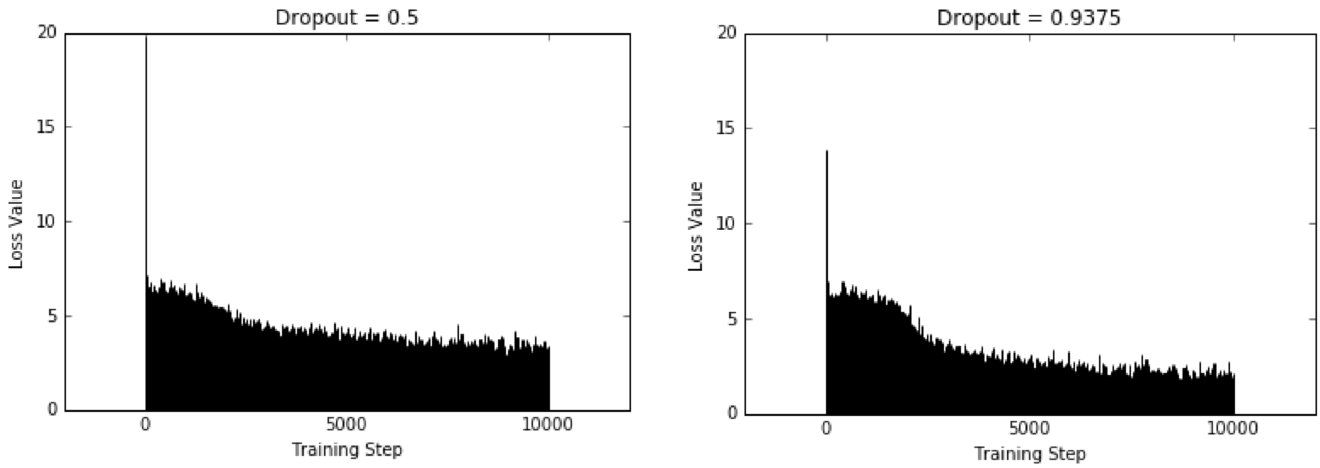


Figure 11 Loss rates with different dropouts

4 Conclusion

In this project we trained a simple convolutional neural network which is trained to recognize numbers on images from SVHN dataset. The accuracy rate of this model is above 90% which is very good. Moreover, we should also take into account that there are a lot of images which is hard to recognize even for a human. Based on that from my perspective the result is very good and even exceed my expectations.

On the other hand, there is still a place for improvement. Neural Network which we trained could be trained with even higher amount of the training data which is able in extra dataset. Also, the architecture of neural network is very simple. There is also a place for improvements, which could be performed with more powerful hardware with GPU, since GPU processors better handle matrix calculations.

5 References

[1] http://ufldl.stanford.edu/housenumbers

[2] http://www.eyemaginary.com/Rendering/TurnColorsGray.pdf

[3] http://yann.lecun.com/exdb/publis/pdf/jarrett-iccv-09.pdf

[4] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] http://jmlr.org/papers/v12/duchi11a.html

[6] https://en.wikipedia.org/wiki/Backpropagation

[7] http://doi.org/10.1109/ICDAR.2011.95

[8] http://cs231n.github.io/convolutional-networks/#norm