

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 of characters still might be difficult. Last years huge amount of open data was published which pushed progress in this area forward.

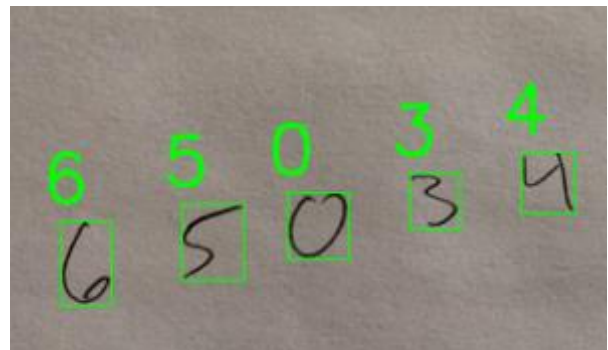
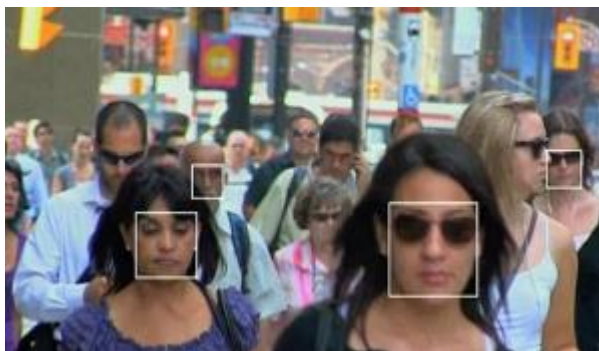


Figure 1 Actual Image Recognition Problems

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.

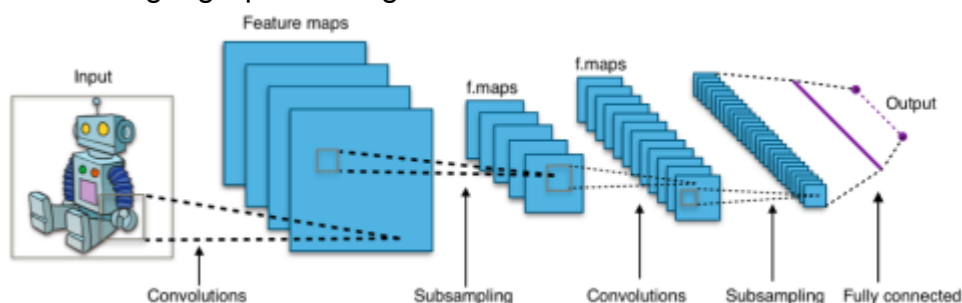


Figure 3 Typical Convolutional Network

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.

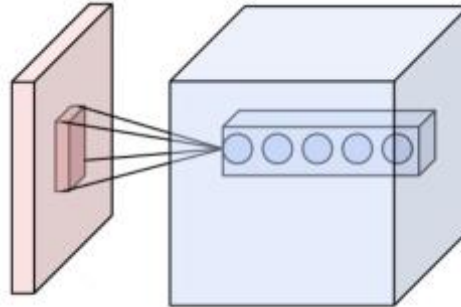


Figure 4 Example input volume(left) and an example volume of neurons if first conv layer

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.

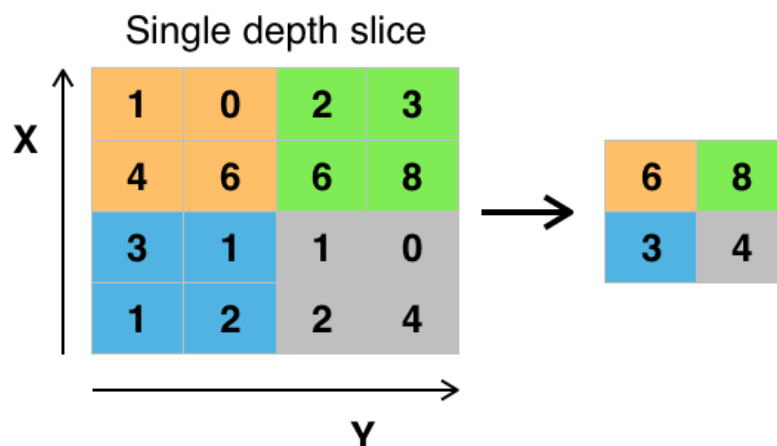


Figure 5 Max pooling with a 2x2 filter and stride = 2

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. This network will be able to recognize numbers with up to five digits with low error rate. It will be able to take an image with house number presented and return the number which is presented on an image. We evaluate this approach on test dataset and achieve over 90% accuracy in recognizing complete street numbers. The reason of choosing

90% as a threshold is quite simple. SVHN dataset contains many samples which is hard to recognize even for a human. Figure 6 contains several of those.



Figure 6 Possible outliers

As you can see images above are very different. They have different size or quality or all together, they can contain some additional symbols nearby. Digits could be written diagonally. All such data is hard to recognize even for a human, so we expect that neural network may also face with some problems here. Moreover, due to a slow hardware, the neural network which we build is going to be quite simple and we don't use all available data, since it is hard to load all it to the RAM. Based on that we expect that our solution will be able to recognize numbers correctly in at least 90% of images from testing set.

2 Analysis

2.1 Data exploration

SVHN dataset consists of 3 groups. Training, testing and extra data. Training set has 33402 samples, testing set has 13068 and extra set has about 230000 samples. 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 7 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. About 90% of training data has 2-3 digit labels. About 80% of test data has 2 digit labels and about 10% of tests data are numbers with 1 or 3 digits each. The majority of extra data are numbers with 2 and 3 digits in a number.

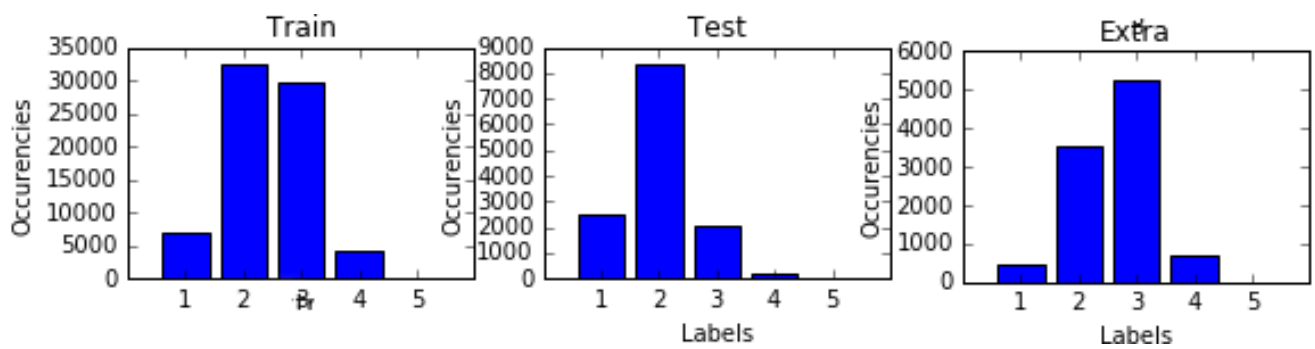


Figure 7 Number length distribution

One more important information about data is a digit distribution across datasets. Figure 8 illustrates how digits are distributed in train, test and extra data. As you can see digits 1-3 are most popular across all datasets.

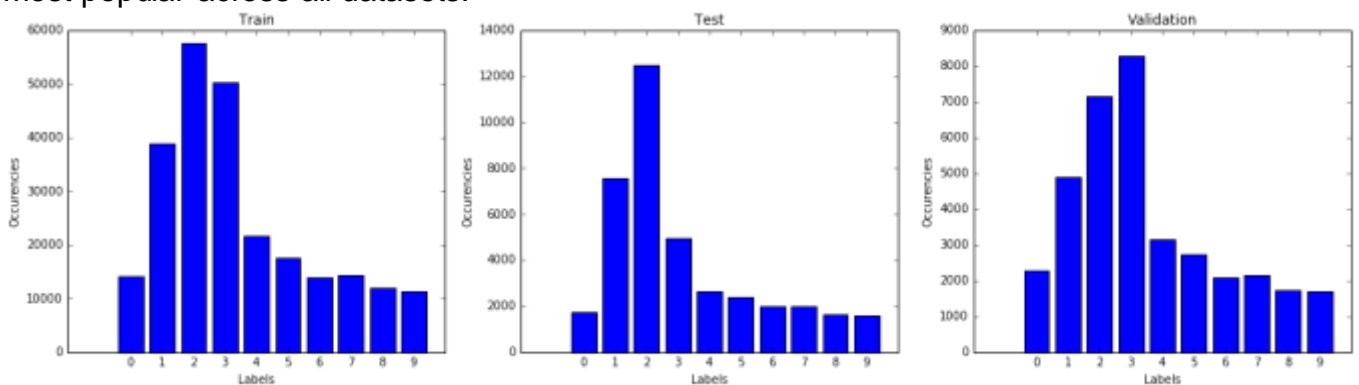


Figure 8 Digit distribution across train, test and validation datasets

Now let's take a look on images sizes. First for, during preprocessing we are going to crop images just near the numbers to make our neural network easier to train. So original image size does not matter. Below are presented min, max and mean sizes of images height and width of train, validation and test datasets after they were cropped just near the numbers.

	Train	Test	Validation
Max Width	876	1083	668
Max Height	501	516	356
Min Width	25	31	22
Min Height	12	13	13
Mean Width	128.29	172.58	100.04
Mean Height	57.21	71.56	60.72

Table 1 Images minimum and maximum width and height

Since images sizes are different we are going to resize all of them to same size during preprocessing step which is described below.

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:

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

3 Methodology

3.1 Data Preprocessing

During data preprocessing we implemented these steps:

1. We cropped each image just near the numbers. By doing so we simplified work for neural network.
2. Then we scale cropped images to 32x32 size since after cropping all images have different sizes.
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 multiplied each pixel level by some value. 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 9 provides several images which were processed by this algorithm:

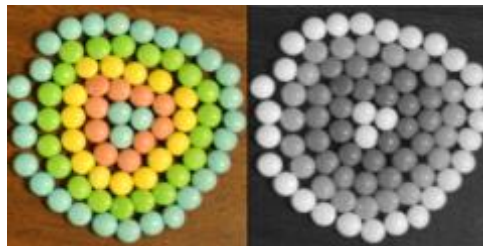


Figure 9 Grayscale algorithm result

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] work.
6. '0' digit class replaced from 10 to 0 to be consistent with other digits and classes. Now we use class 10 to label not-existing digit. It means that altogether we have 11 labels.
7. Then we split each sample class by digit. If number has size less than 5 digits, we add labels 10 at the end. It means that number 567 will consist of a list 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 10 illustrates this approach.

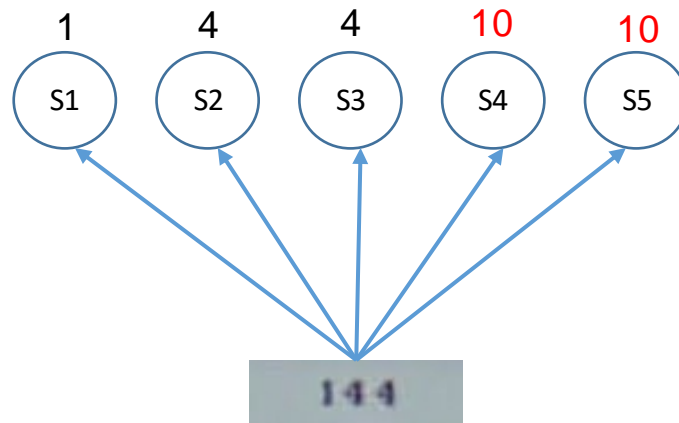


Figure 10 Number recognition architecture

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 11 illustrates the architecture of any single set of layers which is responsible for recognizing a digit (S1, ... , S5).

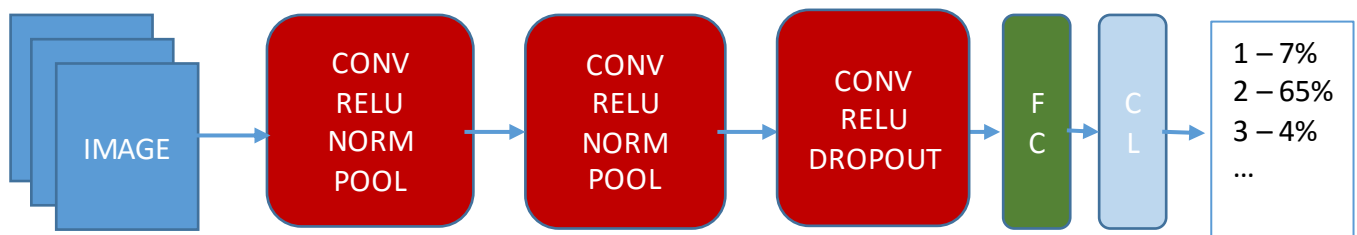


Figure 11 Single digit recognition ConvNet architecture

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 x 16 x 32. And the third one 5 x 5 x 16 x 32. On the output of first, second and third convolutional layer we send data to ReLU layer which doesn't change input size, but applies elementwise activation function $\text{MAX}(X, 0)$ thresholding at zero. Then after first and second ReLU layer we send data to local response normalization layer. Although this kind of normalization layer marked as one, which has a minimal practical impact, for our problem it showed very good results in image classification. At the end, before we send data to a single fully-connected layer we apply dropout to our data with probability 0.9375 and reshape the data into a 2D matrix to feed it to the fully connected layer. Fully connected layer has weight 64 x 11. The received result we pass to classification layer which applies softmax function and gives us information probability about belonging each digit to some class.

During training phase, we use stochastic gradient descent optimization algorithm. Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning. It uses learning rate parameter to determine how fast will it converge. High learning rate helps to converge faster but with higher error rate, while gradient descent with low learning rate learns slower but produces more accurate results.

On the output of our trained model we have 5 vectors of size 11. Each entry contains a probability of belonging digit to some particular class. I-th vector contains probability distribution of i-th digit in a number. If number has length which is less than 5 digits, these vectors should have highest probability for class 10. Figure 12 provides examples of input and output data.

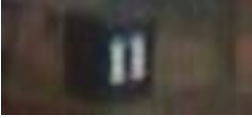
Input	Output
	Softmax(y0)=[0.1, 0.7, 0.01, 0.01, 0.01, 0.02, 0.02, 0.03, 0.05, 0.03, 0.02] Softmax(y1)=[0.02, 0.6, 0.01, 0.01, 0.01, 0.1, 0.12, 0.03, 0.05, 0.03, 0.02] Softmax(y2)=[0.1, 0.01, 0.01, 0.01, 0.02, 0.02, 0.03, 0.05, 0.03, 0.02, 0.7] Softmax(y3)=[0.1, 0.07, 0.01, 0.01, 0.01, 0.02, 0.02, 0.03, 0.05, 0.03, 0.65] Softmax(y4)=[0.1, 0.12, 0.01, 0.01, 0.01, 0.02, 0.02, 0.03, 0.05, 0.03, 0.6]

Figure 12 Input which ConvNet receives and output which it produces

Then we create array Z of same size as produced output and initialize it with following function: $Z[i] = \text{argmax}(\text{softmax}(y_i))$. By using example provided in Figure 11 the array body will be following: $Z = [1, 1, 10, 10, 10]$. Finally, at the end we concatenate all elements from Z which are not equal 10. This our predicted number. Described model recognizes numbers with accuracy 0.92. Figure 13 shows pictures with their actual numbers and those which are recognized by our convolutional neural network.



Figure 13 ConvNet input and output results

3.3 Optimization

During the work on a project I tried many different techniques. Not all of them were useful, here presented those of them which improved overall accuracy: First for all, localization layers helped our model to start train on a data. Although it is labeled like fallen out of favor [8], in my case local normalization helped dramatically. 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 0.92. Figure 14 illustrates minimization of loss value with dropout 0.5(left) and 0.9375(right).

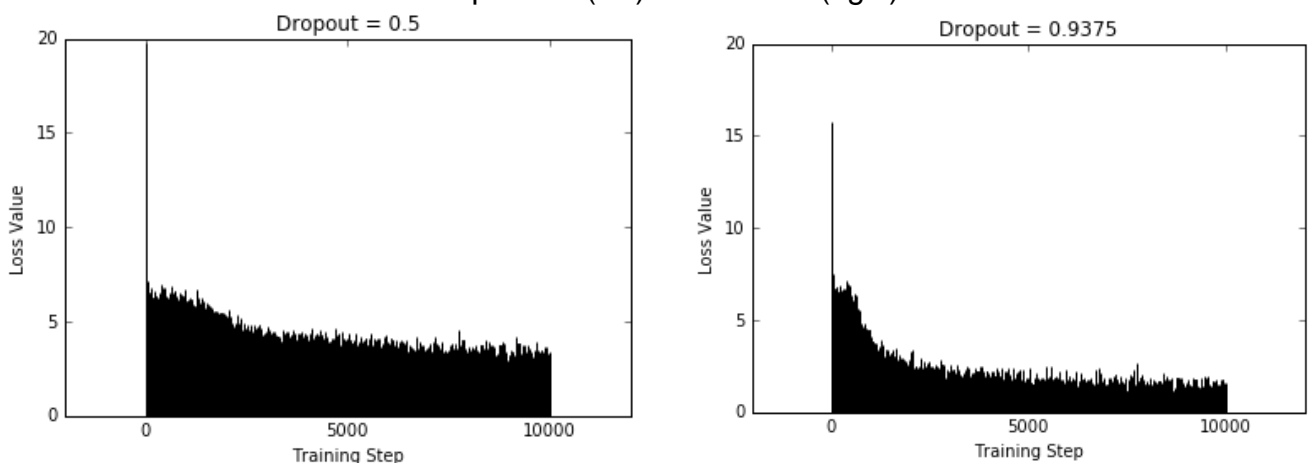


Figure 14 Loss rates with different dropouts

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 about 92% 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. During development I noticed that ConvNet architecture is one of the fundamental key to solve the problem and even small changes produces big impact. One of those examples is a local response optimization layer which labeled as one, which is not very profitable. These layers helped model to start learn from the data. Model without local response optimizer started to learn data, reached accuracy 0.54 on first 1000 steps and stuck here. Figure 15 illustrates loss value for this model.

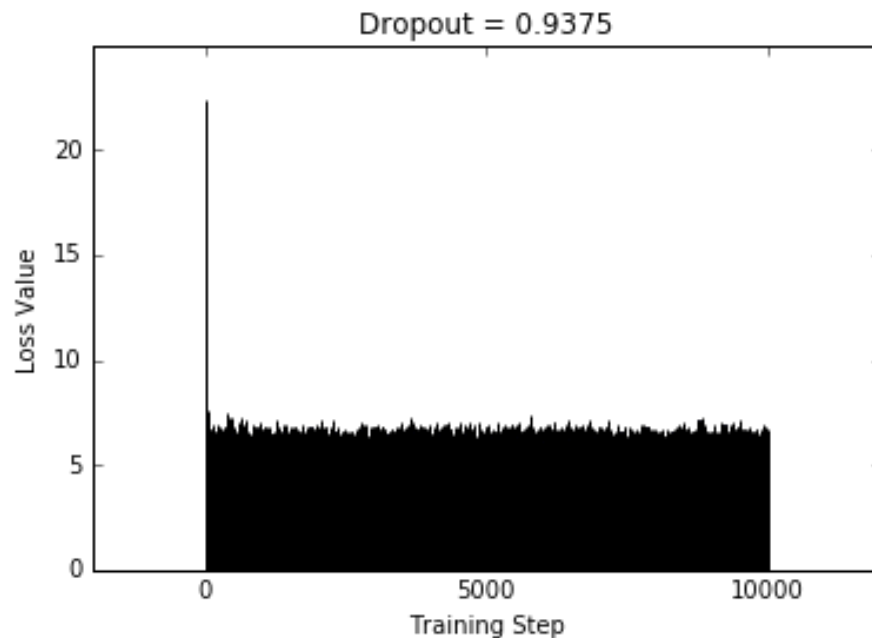


Figure 15 Loss value without local response normalization

Also I tried to apply Abandon [5] optimizer, which is an optimization of SGD. However, with this optimizer our model produced 3% less accurate results.

The Figure 16 shows accuracy of training and validation sets during network training. As we can see, network starts learning very fast at the beginning and almost stops at the end. What is important here is that accuracy values are almost same for training and validation sets. It means that our model is not over fitted.

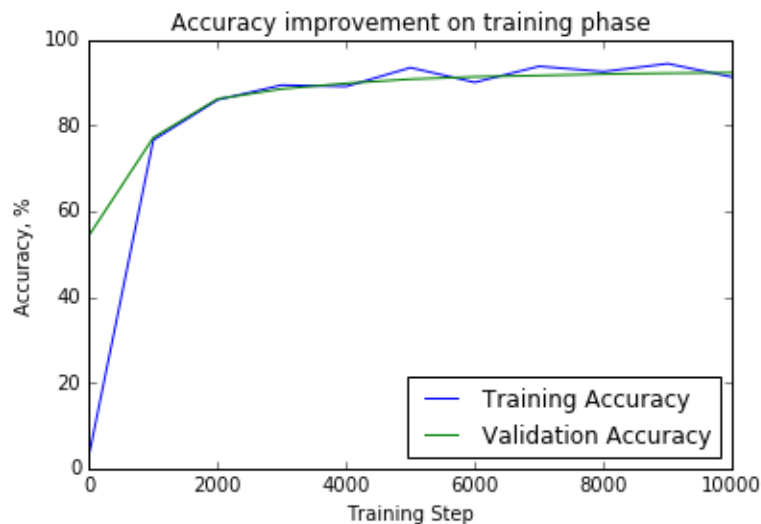


Figure 16 Training and Testing accuracy

Also we need to compare training and validation accuracy according to testing set which neural network was not able to see during training phase. Table below contains accuracy for training and validation datasets on the last training step and accuracy of testing dataset.

Training	Validation	Testing
92.5%	92.4%	92.3%

Table 2 Accuracy for training, validation and testing datasets

Above we see that our model is not over fitted and robust to unseen data. The fact that all datasets have very close accuracy value means that our model is ready to deal with any values from the SVHN dataset and not only those which we have used for training.

By using our final solution, we received 92.3% accuracy on testing dataset which is above a 90% threshold. That means that our goal is achieved and we have good stable solution which recognizes multi-digit numbers with accuracy higher than 90%.

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/house-numbers>
- [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>