Machine Learning Engineer Nanodegree Alexandru Korotcov
Capstone Project: Deep Learning January 10th, 2017

Recognizing digits and numbers in natural scene images

1. Definition

1.1. Project Overview

The human brain visual system is very advanced and it doesn't take any effort for humans to tell apart a lion and a jaguar, read a sign, or recognize a human's face [1]. The major problems in computer vision are still related to object recognition and scene "perception". Many problems need to be solved in order to read text in natural images. A number of factors that introduce challenges to the task are [2]:

- variations in font style and thickness;
- different background as well as foreground color and texture;
- camera position and geometric distortions;
- illumination, lighting, shadows, specularities, and occlusions;
- character arrangements;
- image acquisition factors such as resolution, motion, and focus blurs;

The success of "AlexNet" [3], a convolutional network and winning entry to the 2012 ImageNet [4] competition [5], inspired many computer vision researchers to apply "AlexNet" to a larger variety of computer vision tasks [6] such as object-detection, segmentation, human pose estimation, video classification, object tracking, superresolution, etc.

Reading text from photographs is one of many difficult computer vision problems that are important for a range of real world applications [7]. Moreover, recognizing multi-digit numbers in images could be an important component of modern-day map making or car plate recognition task. The Street View House Numbers (SVHN) is a real-world image dataset [8] for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

In this project I have built and trained a couple of different multilayered convolutional neural network (ConvNet) models to recognize single digits and multi-digit numbers in street captured level images using the SVHN dataset (total 46470 train, validation, and test images have been used in this project). All images were cropped and

converted to grayscale prior to model training. A high degree accuracy of single digit recognition in multi-digit numbers of 91.2% and whole number of 74.8% have been achieved. Moreover, this model could be used as a base model for developing deep learning models for recognizing digits and numbers in natural scene images.

1.2. Problem Statement

This project is focused on the recognition of single digits and whole numbers (sequence of digits) in multi-digit numbers from SVHN images (Figure 1) with the use of a deep convolutional neural network, which operates directly on image pixel intensities without extra steps of localization and segmentation of each digit from an image. The images shown in Figure 1 are the original house-number images that clearly represent all of the possible challenges associated with recognizing numbers such as variable-resolution, color, illumination, camera position, shadows, etc.

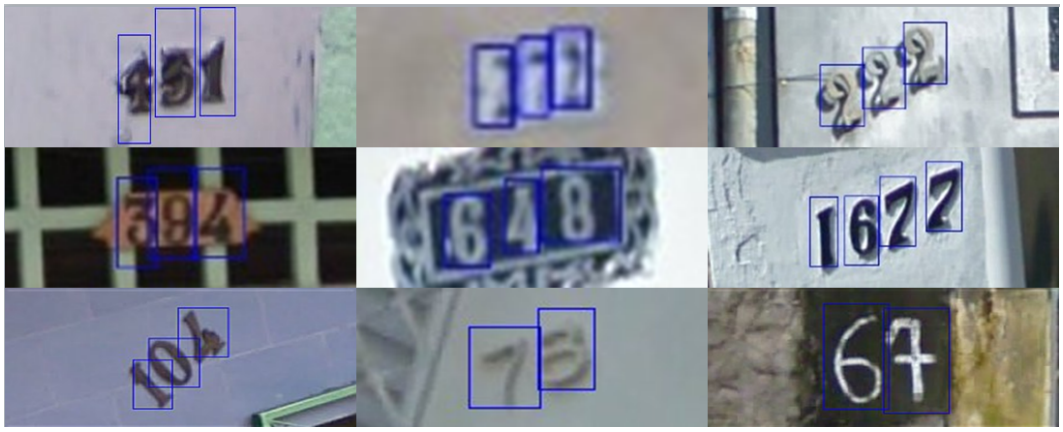

Figure 1. Samples from SVHN dataset with character level bounding boxes (the blue bounding boxes here are just for illustration purposes but have been used in the pre-processing step to crop whole numbers from images).

My approach to solve the number recognition problem is to build a couple of different convolutional deep neural networks sequence transcription models of different complexity which will probabilistically (softmax function) classify each digit from the whole number and then combine them into a sequence of digits. Thus, the whole number will be identified as a sequence of digits, $\boldsymbol{S} = s_1, s_2, \ldots s_n$, where n defines the order of the whole number, and s_n is the argmax of each digit of this particular order. The loss will be computed separately for each digit and then combined, and then estimated. Finally, if $\boldsymbol{S}$ represents the output sequence and $\boldsymbol{X}$ represents the input image, then, the goal is to learn a model $P(\boldsymbol{S}|\boldsymbol{X})$ by maximizing $log\ P(\boldsymbol{S}|\boldsymbol{X})$.

I am planning to use the deep ConvNet similar to AlexNet [6] with 3 and 4 hidden convolutional layers followed by a rectified linear unit layer and max polling procedures finalized with 1 and/or 2 fully connected layers with 75% dropout, followed by a final

digits classification sequence transcription, which is composed of 4 locally connected parallel layers (each one per one digit in the whole number). The final prediction is a sequence of digits, $s_1\ s_2\ s_3\ s_4$, where s_n is an actual digit from 0 to 10 (10 means 0 and 0 means no digit in this particular position, for example, 201 will be represented as a sequence of 2, 10, 1, 0).

1.3. Metrics

Since the purpose of this project is to recognize digit/sequence of digits, the accuracies will be a sufficient performance metric. There are two metrics I used to measure the performance of my sequence of digits recognition problem, which are *single digit recognition accuracy* and *whole number (sequence of digits) recognition accuracy*.

Single digit recognition accuracy is defined as a number of correctly predicted single digits in multi-digit image divided by the total number of digits in all images in the dataset, multiplied by 100.

Sequence of digits recognition accuracy is defined as a number of correctly predicted whole number (sequence of digits) divided by the total number of images in the dataset, multiplied by 100.

2. Analysis

2.1. Data Exploration and Visualization

The SVHN dataset is a real-world image dataset [8] obtained from house numbers in Google Street View images (Figure 2) along with bounding boxes for individual digits. The datasets are about 73257 digits for training (train.tar.gz), 26032 digits for testing (test.tar.gz), and 531131 additional, somewhat less difficult samples, to use as extra training data. The labels are 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10. The bounding box information are stored in digitStruct.mat. Each tar.gz file contains the original images in png format, together with a digitStruct.mat file. The digitStruct.mat file contains a struct called digitStruct with the same length as the number of original images. Each element in digitStruct has the following fields: name, which is a string containing the filename of the corresponding image, bbox, which is a struct array that contains the position, size and label of each digit bounding box in the image, and height, which gives the height of the 2^{nd} digit bounding box in the 300^{th} image (Eg: digitStruct(300).bbox(2).height).

There are a total of 33402 and 13068 images in downloaded and extracted train and test datasets, respectively. The digits distributions in the datasets are shown in

Figure 3. There are 9 images with 5-digit numbers and one image with a 6-digit number in the train dataset. I will exclude those images in the future model building because they are not representative of the entire dataset and could be treated as outliers. Similarly, I will exclude 5-digit images from the test dataset (there are two of them in the test dataset).

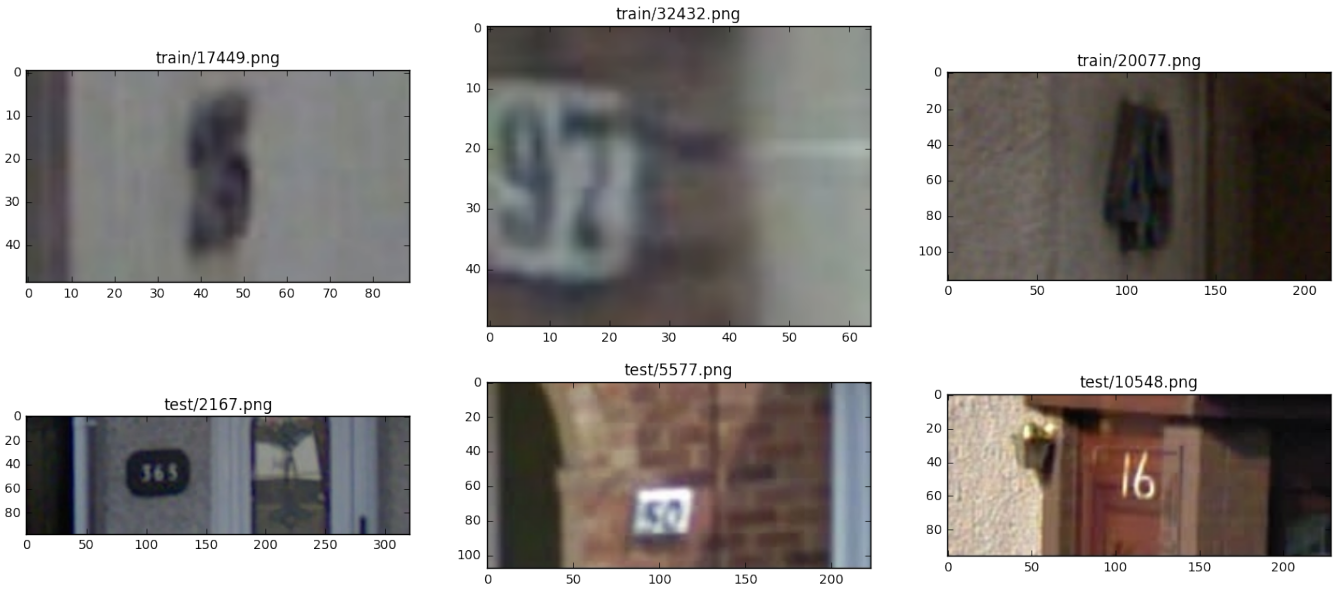

Figure 2. Examples of the images from train and test datasets.

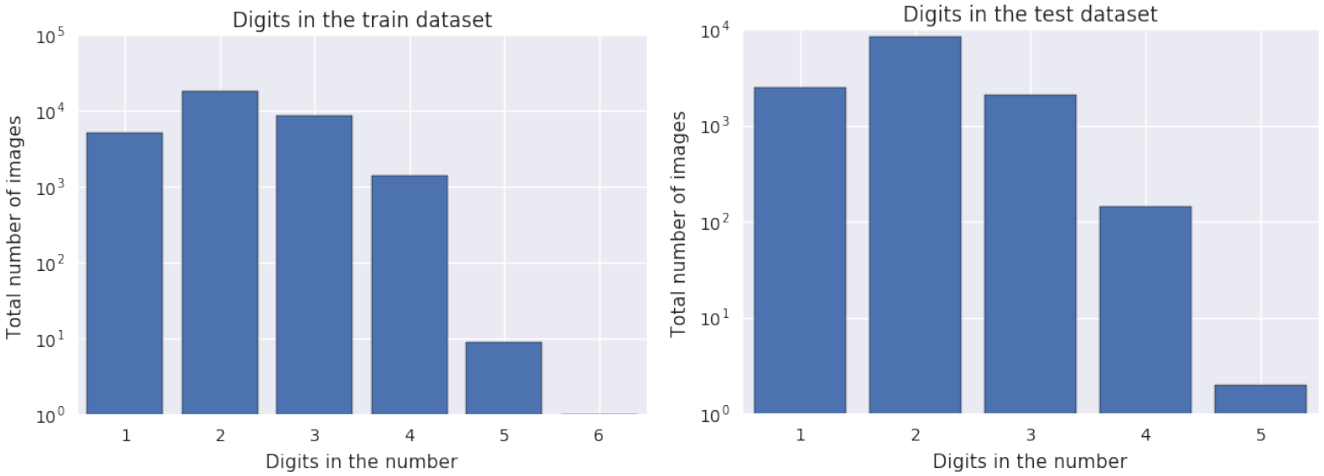

Figure 3. Train and test datasets digits distributions.

The train and test datasets images height and width statistics are summarized in the Table 1. In addition, the image size distribution is shown in Figure 4. The statistical measures listed in the table and shown in the Figure 4 suggest that the train and test datasets have pretty similar distributions of images height and width. The final datasets

will be constructed by combining those train and test datasets, randomized and spitted into training (70%), validation (15%) and test (15%) datasets.

Table 1. Train and test datasets images height and width statistics in pixels.

	max	min	median	mean	std
Train dataset height	501	12	47	57	36
Train dataset width	876	25	104	128	80
Test dataset height	516	13	53	72	52
Test dataset width	516	13	173	132	123

Figure 4. The distribution of relationships between image height and width in train and test datasets.

2.2. Algorithms and Techniques

Deep ConvNets have been at the heart of spectacular advances in deep learning and become a very useful tool for machine learning practitioners [9]. The models built and validated in this project are all ConvNet models of different complexities and configuration. ConvNets are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity [10]. The whole network expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other and a loss function (in my case a Softmax) on the last (fully-connected and locally-connected) layers. ConvNet architectures make the explicit assumption that the inputs are images, which allows encoding of certain properties into the architecture. These then make the

forward function more efficient to implement and vastly reduce the amount of parameters in the network.

ConvNets take advantage of the fact that the input consists of images and, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, and depth. The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output for my ConvNet would have 4 locally connected layers with dimensions 1x1x11, thus the ConvNet architecture will reduce the full image into a single vector of class scores, arranged along the depth dimension (Figure 5) [10].

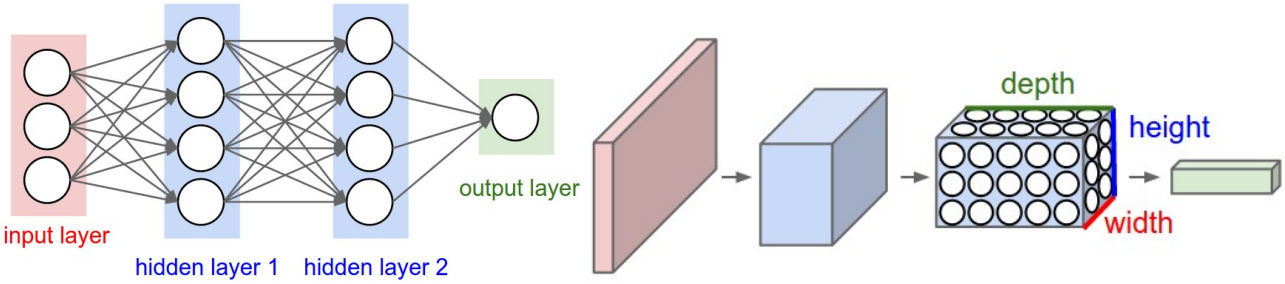

Figure 5. Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels) [10].

ConvNets architecture explored in the project has the following components: Convolutional Layers (3 and 4, each convolutional layer will be followed by a rectified linear unit layer and a max pooling layer), Fully-Connected Layers (1 and 2, rectified linear unit followed by a 75% dropout), and 4 Locally-Connected output Layers. The training data will be spitted into mini-batches and the model will be trained for a number of epochs.

Two optimizers will be used in my models: Gradient Descent Optimizer (mini-batch/ stochastic) [11] and Adam Optimizer [12].

Gradient Descent method is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks [11]. The *Stochastic Gradient Descent* (SGD) performs a parameter update for each mini-batch of training examples and labels, thus making a trade-off between the accuracy of the parameter update in each epoch and the time it takes to perform an update. It has been shown that when we slowly decrease the learning rate, SGD shows very good convergence behavior, and almost certainly converge to a local or the global minimum for non-convex and convex optimizations.

Adam method is a simple and computationally efficient algorithm for gradient-based optimization of stochastic objective functions. This method is aimed towards machine learning problems with large datasets and/or high-dimensional parameter spaces. The method combines the ability to deal with sparse gradients, and the ability to deal with non-stationary objectives. Adam method requires little memory and was found to be robust and well-suited to a wide range of non-convex optimization problems in the field of machine learning [12].

2.3. Benchmark

Ideally we would like to have a model perform similarly to humans. Causes of human mistakes in digit recognition tasks are foreground/background confusion and wrong classifications due to very low resolution/blurred images. The estimated human performance on the SVHN test dataset is 98% [7]. In the literature, the best performance of neural networks with the SVHN dataset (over 600,000 training digit images, training dataset used in this project has only 32256 images) was achieved using a ConvNet with the deepest architecture of eleven hidden layers [13]. The state-of-the-art accuracy of a per-digit recognition task was 97.84% and 96% accuracy in recognizing of a complete street number (10 days of training).

In this project I plan to use a 4-5 hidden layers ConvNet to build models for recognizing digits and numbers in natural scene images from the SVHN dataset. All data pre-processing was done on a Docker machine running on Windows 10 laptop. 10Gb random accessed memory and 4 CPU cores were dedicated to Docker. A Docker image from "Deep Learning" course at Udacity [14] was used to pre-process datasets and start the initial ConvNet tuning. Eventually, I received access to a remote server with Nvidia Tesla K20 GPU, thus I was able to run more intensive model tuning and run some cross-validations. Taking into account the time, computer power, and ConvNet architecture of maximum 4 convolutional layers, I have set up the following benchmarks for measuring performance of my best model: single digit accuracy recognition to 90% and the whole number accuracy recognition to 75% (maximum training time for the final model set to 60 minutes).

3. Methodology

3.1. Data Preprocessing

The following data preprocessing steps have been performed prior to ConvNets model building and training:

- the images were cropped base on bounding box sizes, and resized to 32x32
- the images with more than 4 digits were excluded from datasets due to the reason explained in section *2.1*

- all images were converted to grayscale using a colorimetric (luminance-preserving) conversion to grayscale: R 299/1000 + G 587/1000 + B * 114/1000 [15], and normalized using mean of each image and image intensities standard deviation.
- the image labels were created using the following schema: [*N, D1, D2, D3, D4*], where *N* is a number of digits in the image, D_n is a number from 0 to 10, 0 means that there is no digit at this position, and 10 means there is a 0 in this position. For example, 201 was labeled as [3, 2, 10, 1, 0] (some examples of the images are shown in Figure 6), the model were trained using only [*D1, D2, D3, D4*] labels.
- the original training and test datasets were merged, randomized, and split into 70% training, 15% validation, and 15% test datasets.
- all datasets and their labels were pickled for future reuse.

Figure 6. Examples of randomly picked up examples for train, validation, and test dataset.

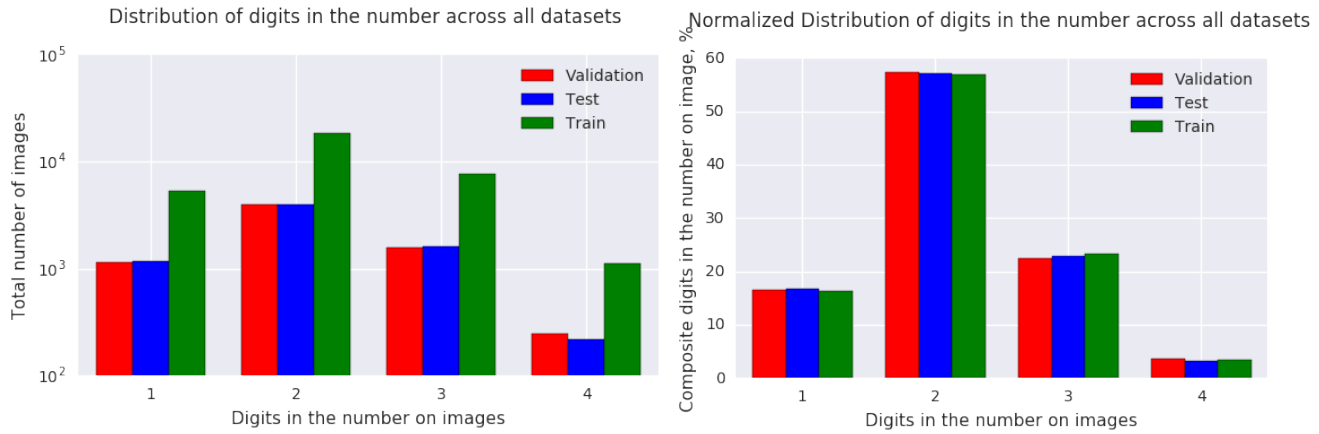

Figure 7. Validation, test, and training dataset number of digits in the whole number distributions.

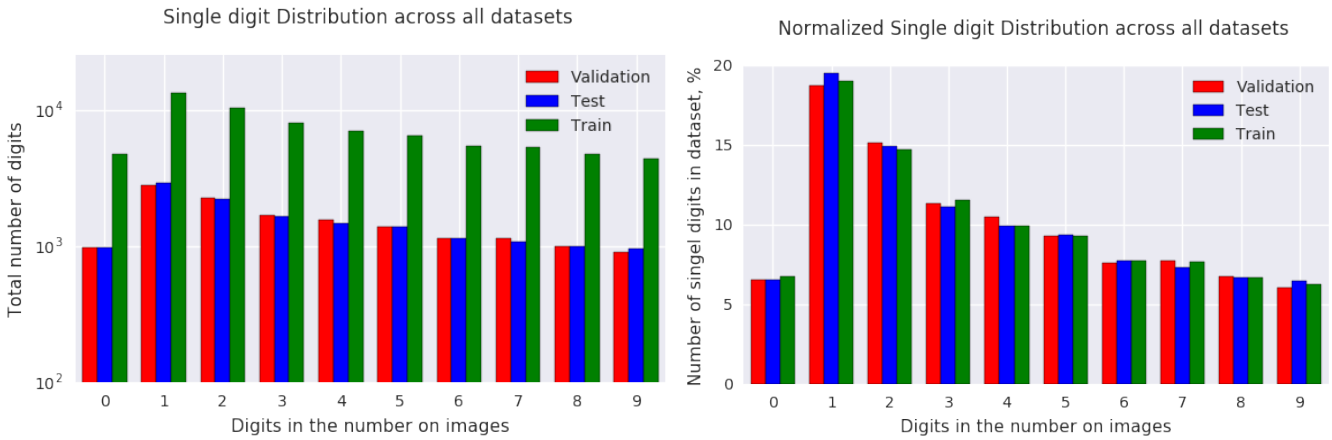

Figure 8. Validation, test, and training dataset of single digits distributions.

The absolute and normalized number of digits in the whole number distributions in train, validation and test datasets are shown in the Figure 7. Most of the images are sequences of 2 digits, therefore, the images of different complexity numbers were proportionally distributed across the datasets and the split could be considered as a good preprocessed datasets. Figure 8 shows single digit distribution in train, validation, and test datasets. The most frequent digits are 1 and 2. All the digits are distributed proportionally across the datasets. The final size of the test dataset was set to 32256 images (to properly match mini-batch sizes), while validation and training datasets each contained 6968 images.

3.2. Implementation

A generalized schema of my convolutional neural network architectures is illustrated in Figure 9. In short, the ConvNet models would look like:

INPUT -> [CONV - RELU - POOL] x 3 (and 4) -> [FC - DOUT] x 1 (and 2) -> LC x 4.

INPUT [32x32x1] will hold the raw pixel values of the image of width 32, height 32, and depth 1 (grayscale).
CONV layers will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Two models will be explored with different filters: [16, 32, 64] in 3 CONV layers architecture, and [8, 16, 32, 64] in 4 CONV layers architecture configuration.
RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged (for example, for the filter 8 it will be [32x32x8]).
POOL layer (max pool) with kernel size [1, 2, 2, 1] will perform a downsampling operation along the spatial dimensions (width, height) (for example, [32x32x8] pooling will result in volume such as [16x16x8].
FC - DOUT (i.e. fully-connected) each neuron in this layer ([1x1024] in single layer and [1x1024] – [1x256] in double layer model) will be connected to all the numbers in the previous volume. 75% dropout followed FC is for reducing overfitting in neural networks, which could prevent complex co-adaptations on training data.
LC (4 locally-connected layers, each one per digit) layer will compute the class scores, resulting in 4 volumes of size [1x1x11], where each of the 11 numbers correspond to a digit score for a particular digit in the sequence.

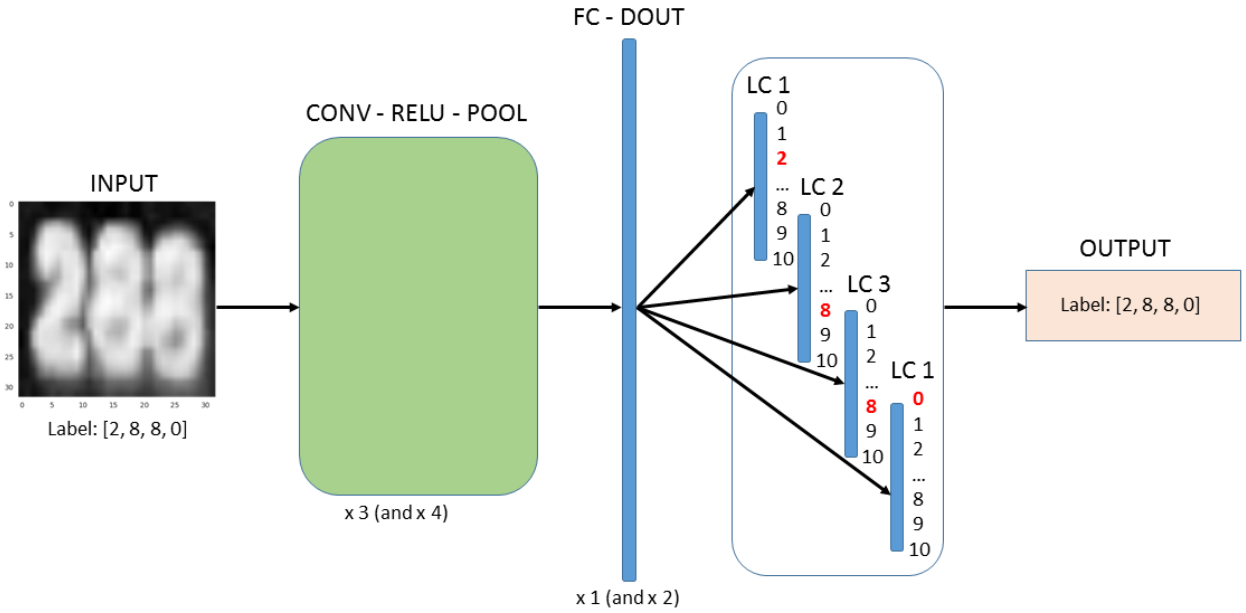

Figure 9. Schematic diagram of the convolutional neural network used in the project with an example of an image of a number 288.

In the way described above, ConvNets will transform the original image from the original pixel values to the final digit scores and combine them into a sequence of digit predictions for the whole number captured on the input image.

3.3. Refinement

In this project I have decided to start with a 3 x [CONV - RELU - POOL] and one [FC-DOUT] layers model. I have introduced a *beta* L2 regularization parameter into each layer of this Model 1. The first step was to tune the L2 regularization parameter by a quick training of Model 1 (10000 epochs) using the following set of *betas*: [0.05, 0.01, 0.0075, 0.005, 0.0025, 0.001, 0.0005, 0.0001]. The results of this cross validation are depicted in Figure 10. The best validation single digit accuracy of 87.5% and the whole number accuracy of 68.2% were achieved using beta 0.0075, thus the *beta* = 0.0075 will be used in the final Model 1 (50000 training epochs) validation.

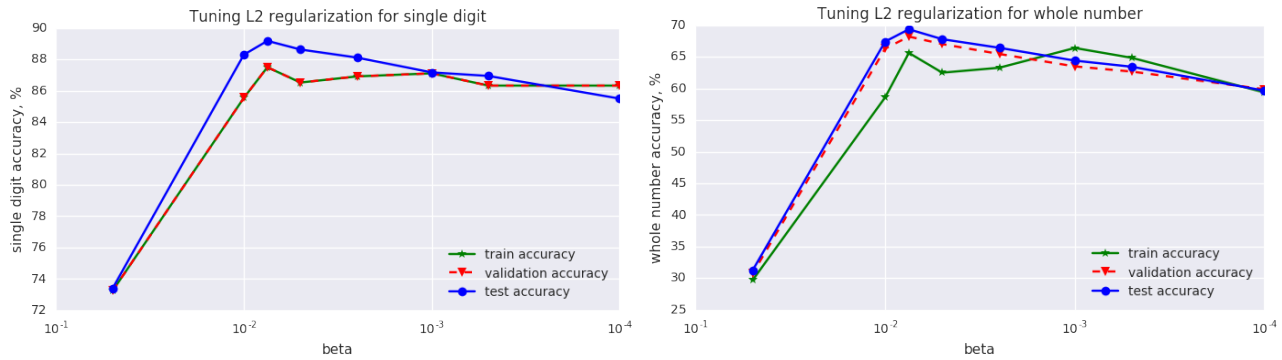

Figure 10. Tuning *beta* parameter of L2 regularization for Model 1.

The Model 2 was an extension of Model 1 by adding one more FC-DOUT layer [1x256]. The same *beta* and learning rate, and learning rate decay rate was used to validate Model 2 (50000 training epochs). The performance of the Model 2 was slightly worse compared to Model 1, and it may be a consequence of not properly tuning some of the parameters used in this model; for instance, beta and learning rates need to be tuned in order to get better performance. Because of the results achieved by Model 2, I have decided to keep only one FC-DOUT layer in all further models, thus reducing the complexity of parameters tuning.

The Model 3 is a model, with maximum complexity I was planning to train and validate. Model 3 is also an extension of Model 1, but the complexity was increased by adding one more convolutional layer to Model 1. Because the Model 3 has underwent very significant modification, and taking into account the results of Model 2 validation, I have decided to perform *beta* (*beta* = [0.01, 0.0075, 0.005, 0.0025]) and learning rate decay rate (starting learning rate will be the same = 0.01, decay rates = [0.875, 0.9, 0.925, 0.95]) tuning of Model 3 for better performance (10000 epochs per condition).

The results of this validation training for single digit and sequence of digits are shown in Figure 11.

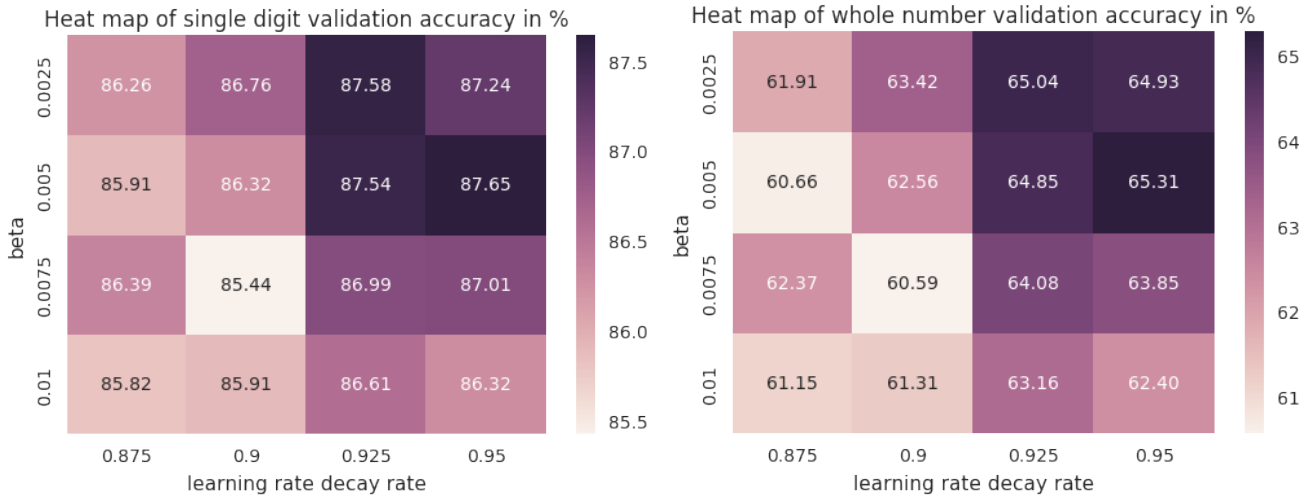

Figure 11. Heat maps showing performance of Model 3 for different *beta* L2 regularization and learning rate decay rate tuning parameters setup.

The ultimate purpose of this project is the accurate recognition of the whole number, thus I chose *beta* = 0.005 and learning rate decay rate of 0.95 with decay step of 1000 for the future models training. In addition to beta and learning rate decay rate I have decided to check the performance of Model 3 with different patch sizes = [3, 5, 8] and 10000 epochs (beta = 0.005, learning rate decay rate was 0.95). The heat maps of the validation results are illustrated in Figure 12. As we can see, the Model 3 performed better with path size = 5.

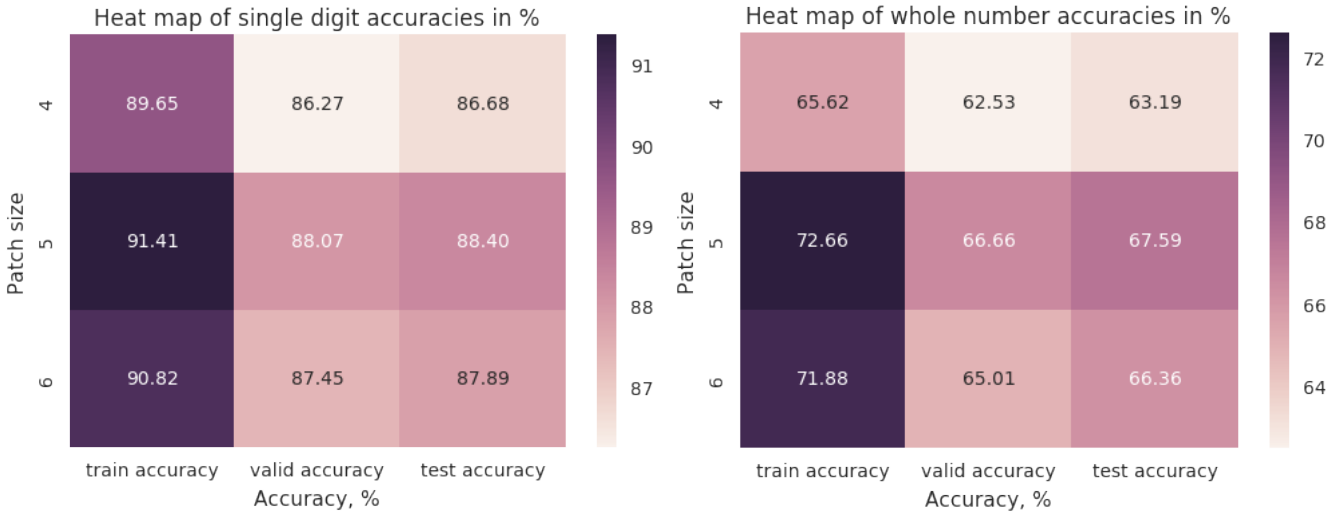

Figure 12. Heat maps showing performance of Model 3 for different patch sizes.

Finally, Model 3 was trained and validated using following parameters: batch size = 128, [CONV - RELU - POOL] x 4 layers, [FC - DOUT 75%] fully connected layer, patch

size = 5, beta (all layers) = 0.005, starting learning rate = 0.01, learning rate decay rate = 0.95 (every 1000 steps), number of training epochs=50000. The performance of Model 1, 2, and 3 looked very similar.

Further improvement of the Model 3 was done by changing the optimizer to the Adam Optimizer (the rest of parameters were the same). Model 4 exhibits slightly better performance than the previously trained models. Some more refinements and validation was done by changing batch size to 256 for Model 1, 3, and 4. Base on the validation accuracies for Model 5, 6, and 7 (see Table 2) the model with Adam Optimizer and batch size 512 (computer memory allowed me to run this training) were chosen as a *final Model*. The number of training epochs for the *final Model* were set to 100000 (final training time 77 min).

4. Results

4.1. Model Evaluation and Validation

In total, 8 models have been trained in this project and each higher-level model was a logical extension of the previous models and/or some parameters changes for better performance. *Beta* L2 regularization, learning rate decay rate, patch size, batch size, and optimizers were tuned to construct the *final Model*. The maximum possible batch size (base of the computer memory) and number of training epochs (base on training time) were used in the *final Model*. The summary of validation and test accuracies for all models is consolidated in Table 2.

Table 2. Basic parameters and best performance accuracies for all trained models.

Model	Batch size	Epoch	Opti-mazer	Single digit valid accuracy %	Whole number valid accuracy %	Single digit train accuracy %	Whole number train accuracy %	Train time min
1	128	50K	SGD	90.0	71.4	90.4	72.7	19
2	128	50K	SGD	88.0	66.9	88.4	68.3	22
3	128	50K	SGD	89.7	70.2	89.9	71.7	17
4	128	50K	Adam	90.0	71.4	90.0	71.5	17
5 as 3	256	50K	SGD	90.1	71.8	90.8	73.5	24
6 as 4	256	50K	Adam	90.3	72.2	90.7	74.1	24
7 as 1	256	50K	SGD	89.9	71.0	90.4	72.6	30
Final	512	100K	Adam	91.0	73.8	91.2	74.8	77

Here is the *final Model* architecture:

INPUT -> [CONV - RELU - POOL] x 4 -> [FC - DOUT] x 1 -> LC x 4.

Details of ConvNet shape architecture of the *final Model* is listed below:
INPUT [32, 32, 1]
CONV - RELU 1 [32, 32, 8]
POOL 1 [16, 16, 8]
CONV - RELU 2 [16, 16, 16]
POOL 2 [8, 8, 16]
CONV - RELU 3 [8, 8, 32]
POOL 3 [4, 4, 32]
CONV - RELU 4 [4, 4, 64]
POOL 4 [2, 2, 64]
FC-DOUT 1 [1, 1024]
LC digit 1 [1, 11]; LC digit 2 [1, 11]; LC digit 3 [1, 11]; LC digit 4 [1, 11]
Predictions shape [4, 1, 11]

The training and validation accuracies evolution during the training process of the *final Model* are shown in Figure 13. The model trains pretty fast in the first 5000 epochs (~ 65% validation accuracy for the whole number), and then exhibit a slow training behavior, but with a constant increase of the validation accuracy (went up to 73.8% at 100000 epochs).

Figure 13. Training and validation accuracies evolution during the training process of *final Model*.

4.2. Justification

The *final Model* (4 convolutional layers and one fully connected layer, training time 77 min) exhibit pretty impressive best accuracies for validation (91.0% - single digit, and 73.8% - whole number) and for test dataset (91.2% - single digit, and 74.8% - whole number). The achieved results are pretty close to the benchmarks I set for this project:

single digit accuracy recognition to 90% and the whole number accuracy recognition to 75% (maximum training time for the final model set to 60 minutes).

Unfortunately, the performance of my *final Model* is not perfect, especially comparing it to performance of the state-of-the-art model from the literature (96% accuracy in recognizing the whole number, 11 hidden layers, and six day training [13]) or human performance on the same SVHN dataset (98% [7]). Nevertheless, my *final Model* definitely could be used in single digit recognition and serve as a base model for recognizing digits and multi-digits numbers in natural scene images and further ConvNet models development.

5. Conclusion

5.1. Visualization and Reflection

In this project I have explored different architectures of the convolutional neural networks to build and train a reliable model of recognizing digits and numbers in natural scene images. The Street View House Number dataset from Google Street images was used for this purpose. The main approach of building a final model was to start with a simple model constructed from three convolutional layers (CONV-RELU-POOL), one fully-connected layer (FC-DOUT) and four locally connected layers (each layer per digit in sequence of digits in a whole number). All the models were tuned and cross validated to get the best beta L2 regularization, learning rate, and optimizer. The performance of the *final Model* was pretty good. The best single digit accuracy for test dataset was 91.2%, and the whole number accuracy was 74.8%.

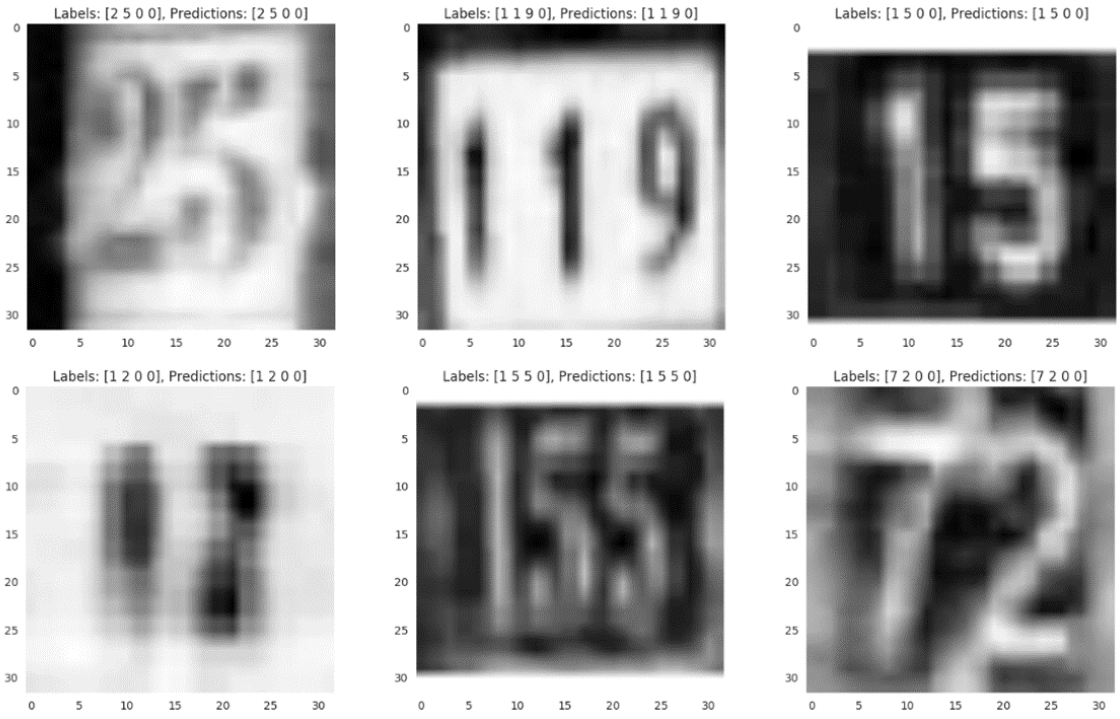

Figure 14. Examples of accurately predicted images from SVHN dataset using *final Model*.

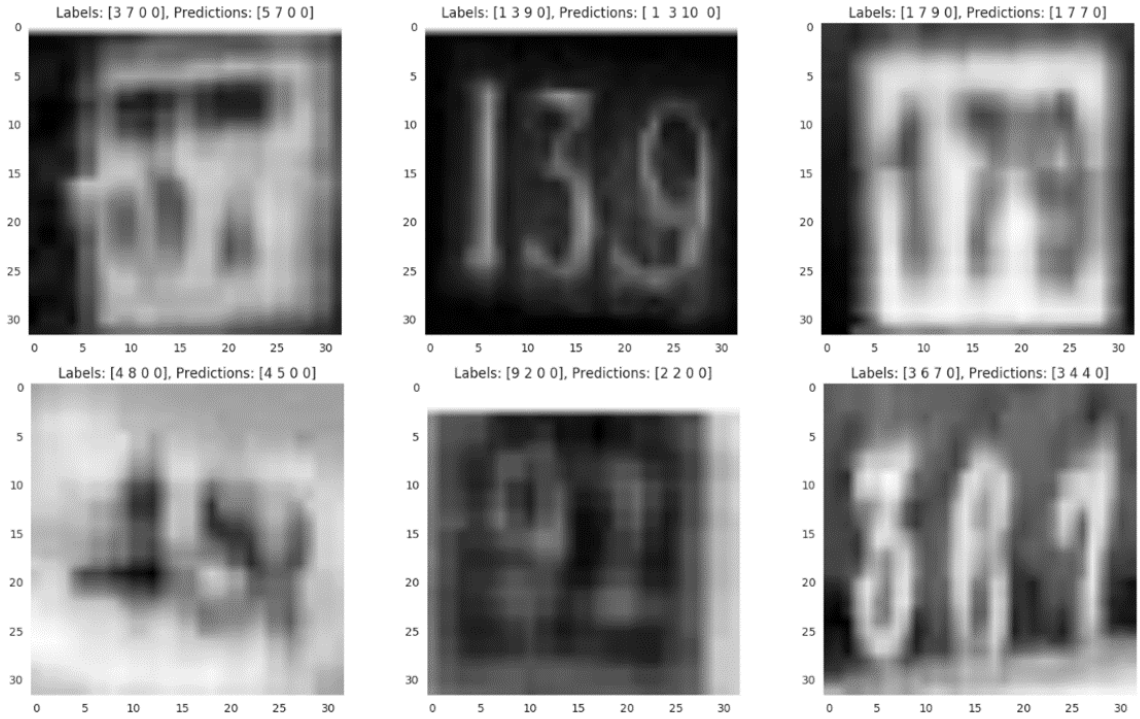

Figure 15. Examples of falsely predicted images from SVHN dataset using *final Model*.

The examples of accurately and falsely predicted numbers with their labels and predicted labels are shown in the Figure 14 and Figure 15, respectively. It looks like those with most of the falsely identified numbers are quite hard numbers for recognition even for human, especially in the grayscale. Nevertheless, despite some very blurred, shadowed, or shined whole numbers that were difficult to predict, most of the single digits in those hard-to-recognize numbers were processed very well by the *final Model*. Finally, we may conclude that the *final Model* is good for the task of single digit recognition and could be used for further ConvNets model development for sequence of digits recognition in natural scene images.

5.2. Improvement

The possible and probably the most effective first step in the improvement of *final Model* performance is to use color (RGB), rather than gray scale converted images to train the model. I believe that by using color images more features could be picked up by the model during the training procedures. The second, and probably the most important step in model accuracies improvement, would be increasing the training dataset size. The *final Model* in this project was trained only on 32256 images, therefore more than 600,000 training digit images remain available in the SVHN dataset [8]. Lastly, but not the least possible step to improve the digits recognition task performance, is to

use a slightly different architecture of neural nets, such as an *Inception* architecture, which is currently becoming very popular [6] and it is widely used by Google. The basic idea of this architecture is rather than going deeper with multiple convolutional layers, to stack different optimal convolutional layers on top of each other, thus decreasing computational cost very efficiently and at the same time modelling a very high performance network.

6. References

1. Sam Ashken. What is Computer Vision? – Part 4: Human Vision V.S. Computer Vision. https://blippar.com/en/resources/blog/2016/08/15/what-computer-vision-part-4-human-vision-vs-computer-vision/
2. T. E. de Campos, B. R. Babu and M. Varma. Character recognition in natural images. In Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal, February 2009. http://personal.ee.surrey.ac.uk/Personal/T.Decampos/papers/decampos_etal_visapp2009.pdf
3. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012 http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf
4. Image Net, http://image-net.org/
5. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge 2014. http://hci.stanford.edu/publications/2015/scenegraphs/imagenet-challenge.pdf
6. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567 2015. https://arxiv.org/pdf/1512.00567v3.pdf
7. Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning, *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
8. The Street View House Numbers (SVHN) Dataset, http://ufldl.stanford.edu/housenumbers/
9. Vincent Dumoulin, Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv:1603.07285v1 [stat.ML], https://arxiv.org/abs/1603.07285

10. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/
11. http://sebastianruder.com/optimizing-gradient-descent/
12. Diederik Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980v8 [cs.LG], https://arxiv.org/abs/1412.6980
13. Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. arXiv:1312.6082v4 [cs.CV], https://arxiv.org/abs/1312.6082
14. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/udacity
15. http://pillow.readthedocs.io/en/3.4.x/reference/Image.html