Capstone Project

Convolutional Neural Networks for Street View House Numbers

Hongwei Liu

1. Introduction

1.1 Project Overview

SVHN (Street View House Numbers) is a real-world image dataset with house number pictures cropped and collected from Google Street Views. It is usually used to test the effectiveness of new machine learning and object recognition algorithms. In general, it is often compared to MNIST (handwritten digits) in many aspects and SVHN tends to be even harder to deal with because those real world housing number pictures from Google Street Views contains much more unexpected noise than MNIST's.

Deep learning research is super hot these years with its powerful help in various areas like object classification in photographs, automatic handwriting generation, automatic machine translation and so on. It can significantly save people from boring and repeating works so that they can focus on creative work. For example, postal services has been using handwriting recognition technique to classify incoming letters which is not only faster but more accurate than human. Most recently, IBM has unveiled its research achievement in health care using deep learning to read medical images to help patient diagnosis, treatment and monitoring.

Many research papers have been working on different algorithms and models to improve the accuracy of image classification in SVHN and other datasets. Generally, human will have a 98% accuracy in SVHN test (Sermanet, 2012), in his research, he adapted the traditional ConvNet architecture by using multi stage features and Lp pooling to achieve 94.85% accuracy on the SVHN dataset. Later different researchers have managed to improve the accuracy to the latest 98.31% (Lee, 2016) by December, 2016.

1.2 Problem Statement

The goal of this project is going to build a model composed of several layers of neural networks. All the training is conducted using SVHN datasets. After the model has been well tuned by learning information from quantities of images and their corresponding numbers, the model could then be used to recognize the number in pictures not in the SVHN datasets which means it would be a generalized model for extracting numbers from picture used in many areas. In general, there are many types of number classification. For example, it can be multiple digit classification, single digit classification, mixed classification (having numbers and letters) and so on. In this project, the model is going to focus on single digit classification.

Inputs:

train_32x32.mat, test_32x32.mat and extra_32x32.mat could be downloaded from the Street View House Numbers Dataset website (http://ufldl.stanford.edu/housenumbers/).

Figure 1 Samples of Original SVHN Images

There are 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.
There are 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
The data file contains four dimensions. For example, training data's shape is (73257, 32, 32, 3) where 73257 means the number of images; 32 indicates the size of images; 3 refers to RGB pictures which has 3 color channels while grayscale picture has only 1 channel.

The model is supposed to use several convolutional layers, pooling layers, fully connected layers together with many statistical methods, linear algebra and other techniques to build a well behaved model.

1.3 Benchmark

As is mention above, until December, 2016, the highest accuracy rate 98.31% is achieved by Lee (2016). Among the papers trying to improve accuracy in classifying SVHN datasets, most researchers achieving high accuracy are stuck in the level of 98%. Considering my computer couldn't afford that many layers neural networks as people did in the research, I decided to set 93% accuracy as benchmark.

1.4 Metric

Since this model is focused on single digit classification, the metric to determine the quality of the model is to calculate the number of correctly predicted numbers dived by total number of predictions.
Also validation accuracy is paid special attention since overfitting might exist when the accuracy is high. Test accuracy only reflects the result of training processing so the difference between accuracy and validation accuracy is used to demonstrate if the trained model is a generalized one instead of a specific model for training data itself. The smaller the difference, the more general the model.

1.5 Label Data Visualization:

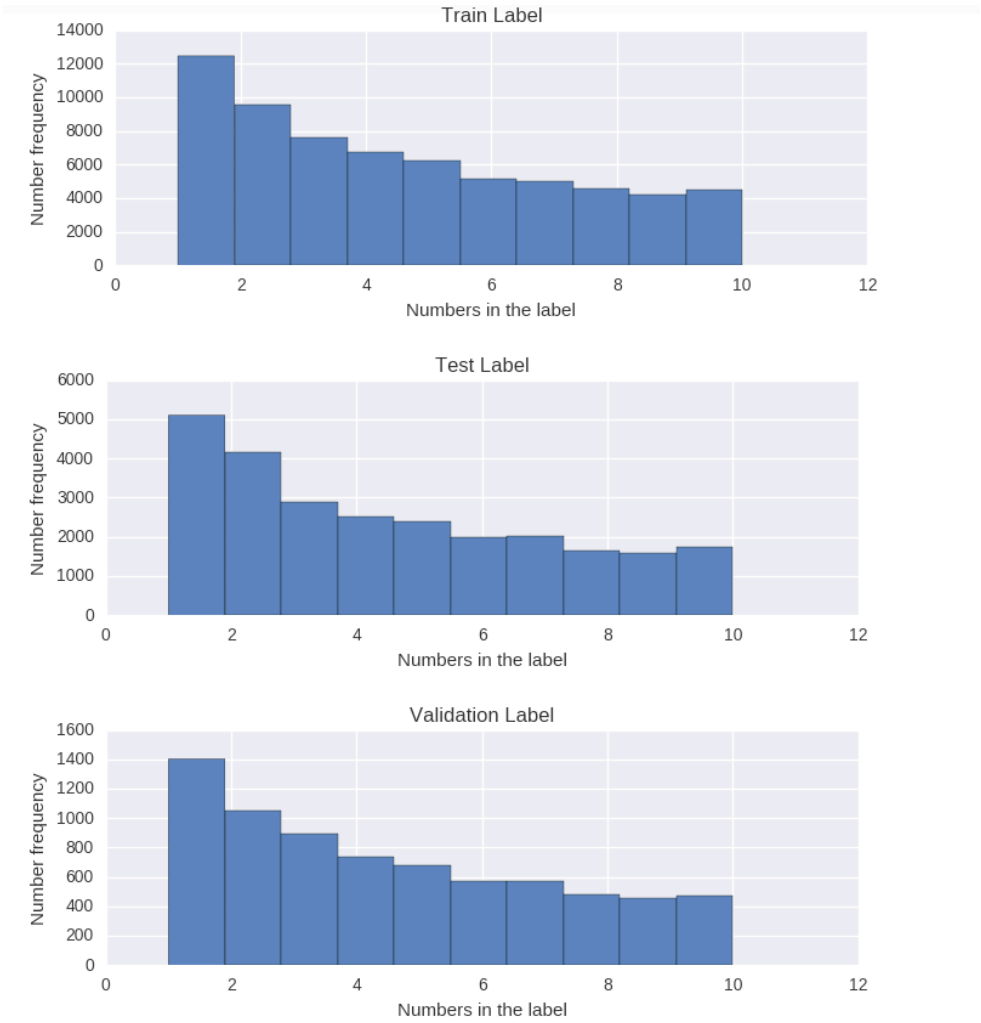

Figure 2 Histogram of Numbers in Labels in Train, Test and Validation Label Datasets Respectively

From the above graph, it is quite clear that train, test and validation datasets all have quite similar distributions in label numbers. It is quite important to neural networks because different number has different features. Some numbers like 8 has more features than 1 because 8 has two circles which is easier for neural network to recognize. The similar distribution could guarantee the neural networks are facing similar conditions in train, test and validation datasets.

2. Algorithms and Techniques

There are two major parts of processing in building the model: images preprocessing and convolutional neural networks building and tuning (learning algorithms).

In the image preprocessing part, generally we will first turn a colored picture (RGB) to to a grayscale picture to get rid of unnecessary information because colored pictures have 3 channels while grayscale picture (1 channel) could also convey sufficient information.

Pixels in the pictures are highly correlated with nearby pixels even though most of them are useless which could bring in huge redundant information for computation.

In order to reduce the affections of those irrelevant connections between different features in a picture (like features' position, direction and so on), normalization and whitening are often used to reduce unnecessary computations for computer.

2.1 Image Preprocessing

2.1.1 RGB to Gray

In SVHN datasets, color in the pictures doesn't make any sense since the only information we want to know is what the number is. Considering color pictures have 3 channels and grayscale 1 channels, the picture would be convert to grayscale to reduce unnecessary computation.

2.1.2 Normalization

Normalization is to center data and make data distribute uniformly a certain area through subtract the mean of the matrix and divide by the standard deviation of the matrix.

Subtracting the mean could mitigate image's brightness variations and divided by the standard deviation could minimize variation in the spread of the matrix.

2.1.3 Whitening

Whitening is mainly used to get rid of useless high correlation between adjacent pixels by extracting eigenvalue and eigenvector from a picture and keep through principal components analysis (singular vector decomposition if image doesn't have same length and width, not a square matrix.).

Here is one whitening example published on Stanford's deep learning website:

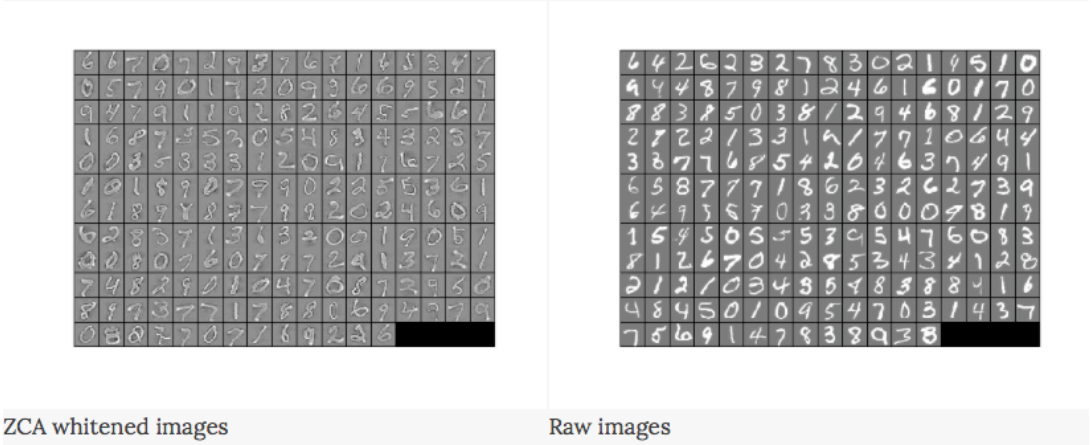

Figure 3 ZCA Whitening Images VS Raw Images without ZCA Whitening

From the raw images to ZCA whitened images, it is clearly that details of raw images are remove with only important, feature like information left.

2.1.4 Local Response Normalization

Local Response Normalization(LRN) type of layer is useful when using neurons with unbounded activations like ReLU (activation function). LRN will dump responses that are uniformly large in a local neighborhood while permitting the detection of high-frequency features with a big neuron response. It is a type of regularizer that encourages "competition" for big activities among nearby groups of neurons (Ioffe, 2015).

2.2 Learning Algorithms

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks are a type of Neural Networks especially designed for images recognition. Similar to regular neural nets, it is made up of neurons containing weights and biases which could be updated by later input data.

In general, the Convolutional Networks are mainly made up of three different layers including convolutional layers, pooling layers and fully connected layers.

A Typical CNN Architecture (Hu, 2015)

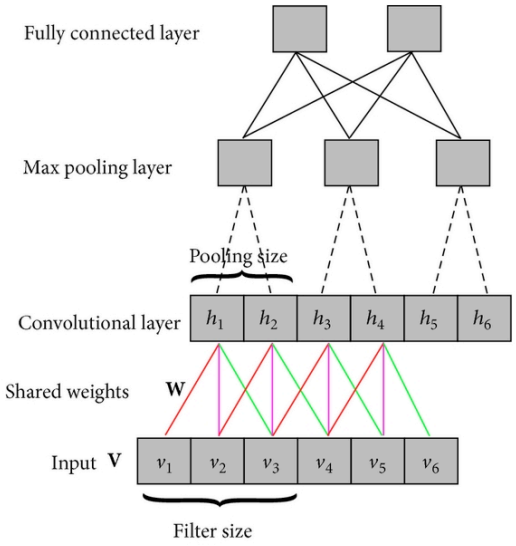

Figure 4 A Typical CNN Architecture With One Convolutional Layer, One Max Pooling Layer and One Fully Connected Layer

From the picture above, we can find that this covolutional neural network is using convolutional layer to screen and read in information, use one max pooling layer to drop some information to shrink data while keeping main features and finally using a fully connected layer to classify input pattern with high levels of features extracted by previous convolutional lay and max pooling layer.

2.2.2 Pooling Layer

Generally, pooling layer is used to progressively reduce the size of the input parameters to reduce amount of parameters and computation by dropping some information. Also, it helps in avoid over fitting. There are different types of pooling including Max, L2-Norm, Min and Average pooling and Max is the mostly used one. Here is one demonstration how max pooling drop information while keeping main features of each parts. In the picture we could find that in each square, max pool get the biggest number, keeping significant values, while dumping other information to save space and reduce the amount of computation.

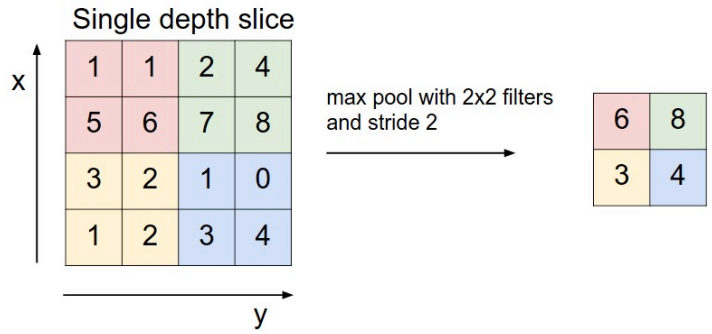

Figure 5 Max Pooling Demonstration

2.2.3 Fully connected layer and Convolutional Layer

Fully connected layer is 1-dimension structure with each node being fully connected to all input nodes to compute weighted sum of all inputs while neurons in convolutional layer only connected to a certain area (defined by filter size and stride) in the input and many neurons share same parameters. However, there are same dot products of weight and picture data happening in neurons which means convolutional layer and fully connected layer are quite similar in general even though they are used in different ways.

2.2.4 Activation Function

An activation function serves as a threshold (classification) to generate reactions when receive signals (data). Softmax and ReLU (Rectified Linear Units) are two popular activation functions for neurons with softmax often used in classification layer and ReLU used hidden layer.

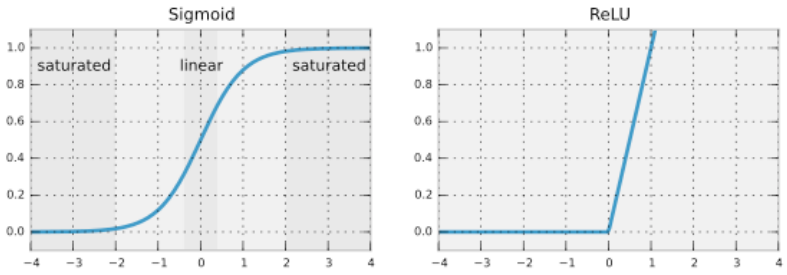

Figure 6 Visualization of Sigmoid and ReLU Function

2.2.4.1 Advantage and Disadvantage of Sigmoid

1. Sigmoid is quite sensitive to small value inputs.
2. When the initial input data is extremely large or extremely small, values between each neuron tend to be same (saturation) which leads to no gradients making it hard for neural networks to learn from following data.

2.2.4.2 Advantage and Disadvantage of ReLU

1. ReLU is easy and faster to calculate and there is no saturating.
2. ReLU is easy to die when there is a large gradient flowing through it and the neuron will never activate on any later input data. This could be caused by a high learning rate.

2.2.5 Optimizer

Optimizer plays a role of optimizing neural networks. There are several optimizers in TensorFlow. Two optimizers used in the MNIST tutorial are Gradient Descent and Adam.
Gradient descent is one of the most popular algorithms to minimize the cost function. It always goes the opposite direction of the gradient to achieve fastest descent.
Adam use adaptive learning-rate and moving averages of parameters during optimization which enable it to have a larger effective step size than gradient descent.

2.2.6 Drop out

Drop out is a regularization technique widely used to reduce overfitting in neural networks. The idea of drop out is to randomly drop units along with their connections from the neural networks during training to prevent units from co-adapting too much (Srivastava, 2014).

2.2.7 L2 Regularization

L2 regularization is used to deal with overfitting by penalizing complexity (higher weights of the model). In general, L2 Regularization shrinks all the weights by the same proportions.
In TensorFlow, there is such a function: tf.nn.l2_loss so L2 Regularization of all the weights are added into loss function then minimize the loss function.

3. Methodology

3.1 Preprocessing

10% of training data has been spitted to be used as validation data so the ratio of final train, test, validation data is 65931 : 26032 : 7326.

Considering the computing ability of the computer I am using and the fact that PCA/ZCA whitening is not often done in practice in large datasets like MNIST, rgb to gray, normalization and local response normalization are applied to SVHN image data.

In addition, since the convolutional neural networks is going to be built under TensorFlow, label data need to be shaped to one-hot encoding like [0 0 0 1 0 0 0 0 0 0] (It is a way to represent numbers in 0 ~ 9).

Sample SVHN images after preprocessing:

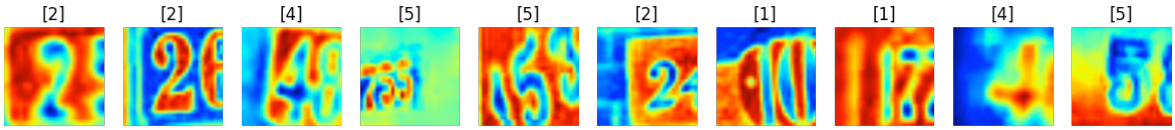

Figure 7 Sample of SVHN after Normalization

3.2 Neural Networks Building and Tuning

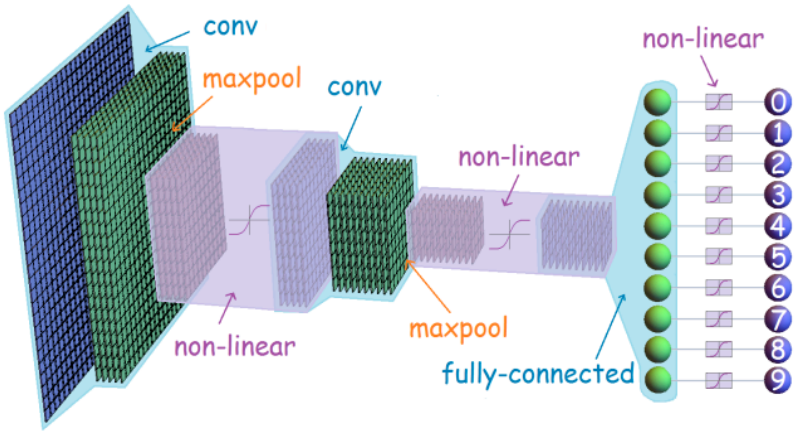

Figure 8 Example of The Procedure of Number Discovery

3.2.1 Building

SVHN datasets' image size is 32 x 32

Parameters for different layers:

patch size	5
tf.nn.conv2d	Strides = [1, 1, 1, 1]
tf.nn.max_pool	ksize=[1,2,2,1], strides=[1,2,2,1]

The size of a feature map (when convolutional layer padding is 'VALID') is calculated by:

(image's height – patch size + stride)^2

Possible Solution 1

1st Convolutional Layer (padding = 'SAME'), Weight = [5, 5, 1, 16]
Before: 32 * 32 * 1 After: 32 * 32 * 16
1st Max Pooling:
Before: 32 * 32 * 16 After: 16 * 16 * 16
2nd Convolutional Layer (padding = 'SAME'), Weight = [5, 5, 16, 32]
Drop out
Before: 16 * 16 * 16 After: 16 * 16 * 32
2nd Max Pooling:
Before: 16 * 16 * 32 After: 8 * 8 * 32
1st Fully Connected Layer:
Before 8 * 8 * 32 After: 1024
2nd Fully Connected Layer:
Before 1024 After: 11

Possible Solution 2

1st Convolutional Layer (padding = 'SAME'), Weight = [5, 5, 1, 16]
Before: 32 * 32 * 1 After: 32 * 32 * 16
1st Max Pooling:
Before: 32 * 32 * 16 After: 16 * 16 * 16
2nd Convolutional Layer (padding = 'SAME'), Weight = [5, 5, 16, 32]
Before: 16 * 16 * 16 After: 16 * 16 * 32
2nd Max Pooling:
Before: 16 * 16 * 32 After: 8 * 8 * 32
3rd Convolutional Layer (padding = 'VALID'), Weight = [5, 5, 32, 64]
Drop out
Before: 8 * 8 * 32 After: 4 * 4 * 64
3rd Max Pooling:
Before: 4 * 4 * 64 After: 2 * 2 * 64
1st Fully Connected Layer:
Before 2 * 2 * 64 After: 128
2nd Fully Connected Layer:
Before 128 After: 11

Possible Solution 3

1st Convolutional Layer (padding = 'VALID'), Weight = [5, 5, 1, 16]
Before: 32 * 32 * 1 After: 28 * 28 * 16
1st Max Pooling:
Before: 28 * 28 * 16 After: 14 * 14 * 16
2nd Convolutional Layer (padding = 'VALID'), Weight = [5, 5, 16, 32]
Before: 14 * 14 * 16 After: 10 * 10 * 32
2nd Max Pooling:
Before: 10 * 10 * 32 After: 5 * 5 * 32
3rd Convolutional Layer (padding = 'VALID'), Weight = [5, 5, 32, 64]
Drop out
Before: 5 * 5 * 32 After: 1 * 1 * 64
1st Fully Connected Layer:
Before: 1 * 1 * 64 After: 32
2nd Fully Connected Layer:
Before: 32 After: 11

Later it turned out that only possible solution 3 could operate without kernel shutting down. It might due to the reason that too many neurons are set in hidden layers cause the kernel to die.

3.2.2 Tuning

To improve the solution 3 to increase accuracy and prevent overfitting:

1. The activation function ReLU is quite fragile and easy to die when the learning rate is very high. However, low learning rate would affect model's learning ability which cost longer time to training. In order to make sure ReLU could work normally which increase the speed of learning, adaptive learning rate (tf.train_exponential_decay) is used.
2. The gradient descent optimizer uses constant learning rate while Adam optimizer is designed to using adaptive learning rate. In addition, Adam optimizer uses momentum (moving averages of parameters) to gain a larger effective step size which makes it faster than gradient descent even though Adam needs more computation. So Adam is the best choice here.
3. Through multiple trials of keep_prob, I find that to my model, keep_prob = 0.7 has the best performance.

4. Result and Model Evaluation

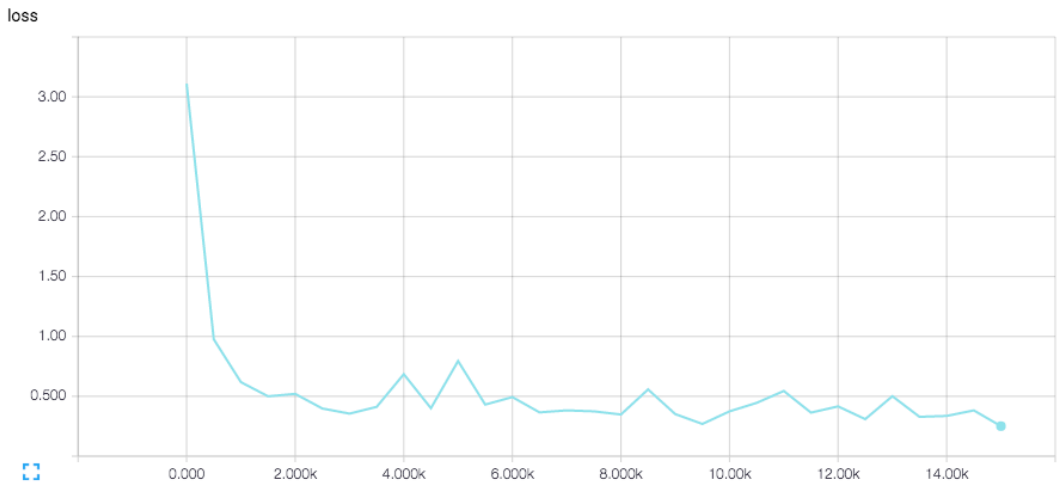

Figure 9 Loss Value During Training

Procedure	Loss	Test Accuracy	Validation Accuracy
Final	0.25	89.9%	90.4%

1. The loss value remains a small value – 0.25.

2. The final test accuracy is 89.9% which is lower than benchmark – 93%.
3. The difference between test accuracy and validation accuracy is 0.5% indicating there is no overfitting.

The model failed to achieve the benchmark. There could be many reasons including small number of neurons and hidden layers, unsufficient normalization to get rid to noise and so on.

5. Prediction Test

Prediction pictures are cropped from by myself in Google Street View in Manhattan, New York. These number pictures are from various places like car, bus, advertisement, house number, street slogan and so on. All 17 pictures are already resized to 32 x 32 pixels.

Figure 10 Samples of Prediction Pictures

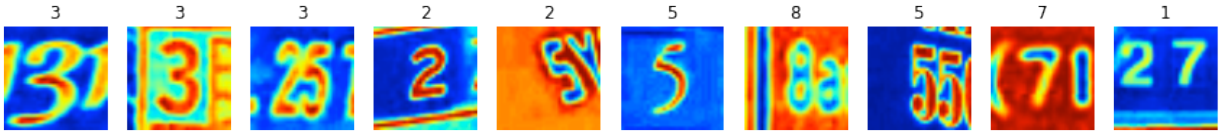

Figure 11 Prediction Result

From the above result, we find that 7 predictions are right and 3 are wrong. Considering the 3 wrong predictions are all multiple digit, the model still has good performance at classifying single digit.

6. Conclusion

The trained model has used many algorithms, techniques and methods, trying to achieve the goal of project. Unfortunately, it failed to achieve 93% accuracy in classifying images in the SVHN datasets. However, 90% accuracy rate is still good as we see in the result of prediction.

More improvement could be done to increase the test and validation accuracy. For example, in the data preprocessing stage, we could apply whitening to pictures to get rid of irrelevant connections in an image; in the data analysis stage, we could set more neurons and use more filters to generate more feature maps. Also, computers with strong computing ability is preferred to conducted more complex treatment with images. In addition, using GPU in deep learning is becoming more and more popular recently so it would be much better to have a powerful GPU to do deep learning investigation.

7. Reference

Hu, W., Huang, Y., Wei, L., Zhang, F., & Li, H. (2015). Deep convolutional neural networks for hyperspectral image classification. *Journal of Sensors*, *2015*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Lee, C. Y., Gallagher, P. W., & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *International Conference on Artificial Intelligence and Statistics*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929-1958.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sermanet, P., Chintala, S., & LeCun, Y. (2012, November). Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on* (pp. 3288-3291). IEEE.

Udacity Deep Learning

Coursera Machine Learning Class by Andrew Ng

https://www.tensorflow.org/

Neural Networks and Deep Learning, Michael Nielsen / Jan 2016

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

CS231n: Convolutional Neural Networks for Visual Recognition

http://ufldl.stanford.edu/tutorial/unsupervised/PCAWhitening/

http://deepdish.io/

http://sebastianruder.com/optimizing-gradient-descent/

https://www.researchgate.net/figure/282478197_fig8_A-typical-CNN-architecture-consisting-of-a-convolutional-layer-a-max-pooling-layer-and

https://www.quora.com/How-is-a-convolutional-neural-network-able-to-learn-invariant-features

https://cp4space.wordpress.com/2016/02/06/deep-learning-with-the-analytical-engine/