Image Classification and Unsupervised Image Object Removal in the Street View House Numbers Dataset

Group 7 Individual Project Report

Introduction

The aim of this project is to solidify the concepts of Deep Learning and diving into the practical implementation of a neural network architecture on real world problems. The problem of recognizing characters in the images have been studied extensively in the past few years. In this project, I intend to train a model that can decode a sequence of digits from natural images. The model will be trained on The Street View House Numbers (SVHN) Dataset, a public subset of street numbers captured by Google street view cars. My objective is to develop a sound theoretical and practical understanding of Convolutional Neural Networks, and to gain the ability to implement training at scale (deep model with a lot of data) with Caffe.

Data Description

Source: http://ufldl.stanford.edu/housenumbers/

SVHN is a real-world image dataset 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.

The overview of dataset is as follows:

- 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
- 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.

I have used format-2 of this data, the MNIST-like 32-by-32 images centered around a single character.

Why Caffe?

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. The reason for using caffe as a service platform is as follows:

- Expressive architecture encourages application and innovation. Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.

- Speed makes Caffe perfect for research experiments and industry deployment. Caffe can process over 60M images per day with a single NVIDIA K40 GPU*. That's 1 ms/image for inference and 4 ms/image for learning and more recent library versions and hardware are faster still. We believe that Caffe is among the fastest convnet implementations available.

Data Preprocessing

Pre-processing the 32-by-32 images from the SVHN dataset centered around a single digit. In this dataset all digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions.

The steps I had to take to get the data ready are described as follows:

1. Caffe takes a Lightning Memory-Mapped Database file as input for training the model. The initial step was to build a lmdb database and put the data into that database. The data was available in a matlab file as a 4D array. The shape of the data was to be changed in the following format:

   (Width, Height, Channel, Number of images) -> (Number of images, Channels, Width, Height)

2. The digit "0" in the dataset was labelled as "10", hence the labels were fixed before the data was inserted in the database.

3. Normalization: Once I got the data into the database, the next part was to normalize the images. I wrote a script to calculate the mean of the data in the training set. The script generates a binaryproto file that is feed to the model in the input layer i.e., the data layer.

Deep Learning Network and Training Algorithm

The Caffe network is designed using the following strategy: a convolution layer is feeding in to a Relu transfer function which is connected to a pooling layer (average). This setup is further connected to a batch normalization layer before the input goes into the second convolution layer.

The design of the final model was then decided by tweaking the network parameters. I tried various batch sizes, learning rates and optimizers to see the changes in the model accuracy.

Experimental Setup

The code required to run the caffe model is in /code/caffe folder in the repository. The folder contains the code for crating lmdb database, data preprocessing, and model training.

These are the following files required to build a model for prediction:

svhnDatareader.py: This file manages to read the data from matlab file and then create a lmdb database for training and testing in the same working directory. This file also handles the mislabeling of digit "0" before putting it in the database.

preprocessing.py: This file calculates the mean of the images in training dataset. It generates a file named “mean.binaryproto” which is further used in the model architecture for normalization.

svhnet_train_test.prototxt: This model architecture is defined in this file.

svhnet_solver.prototxt: This is a solver file. The model parameters such as batch size, learning rate, optimizer etc. are defined in this file.

train_svhnet.py: This is the training file. It takes in the solver and starts traing the model. The model is evaluated in this file.

Model Architecture:

The final model architecture is shown in the following figure:

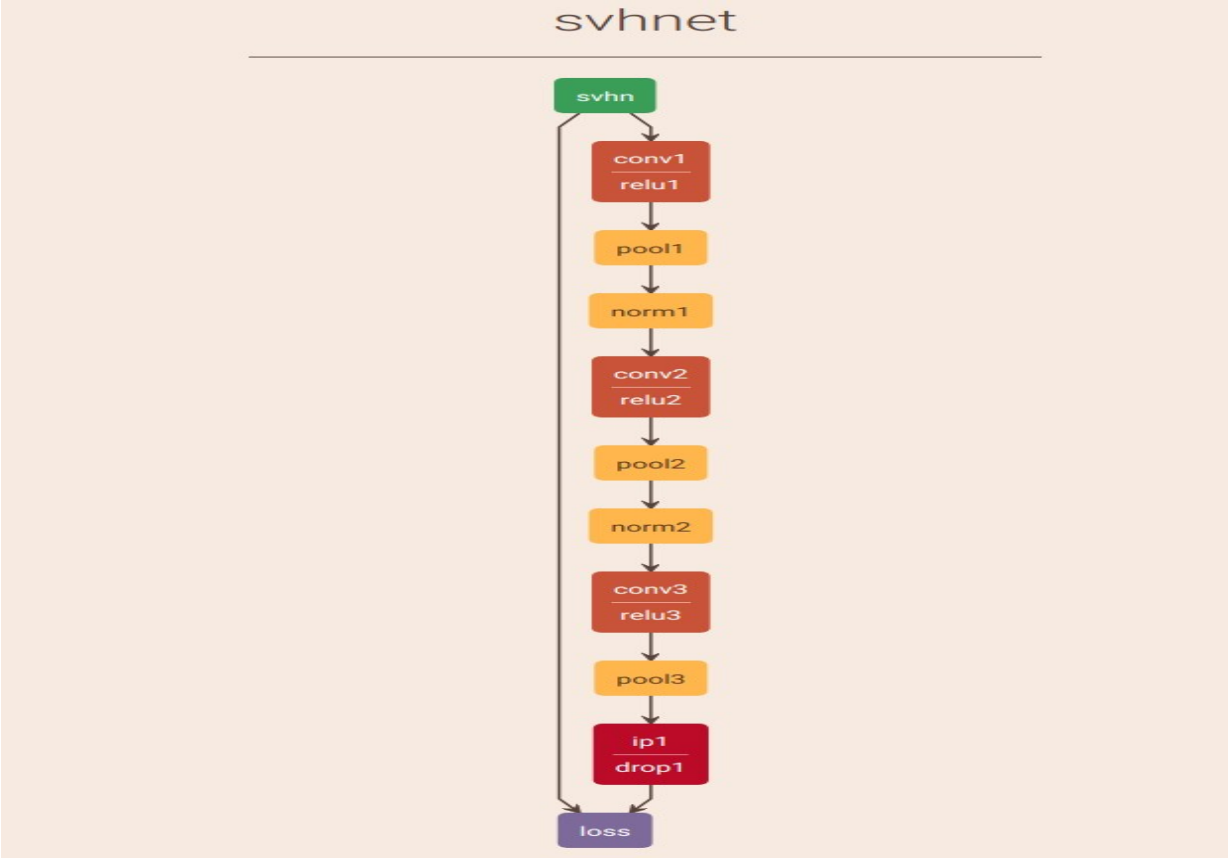
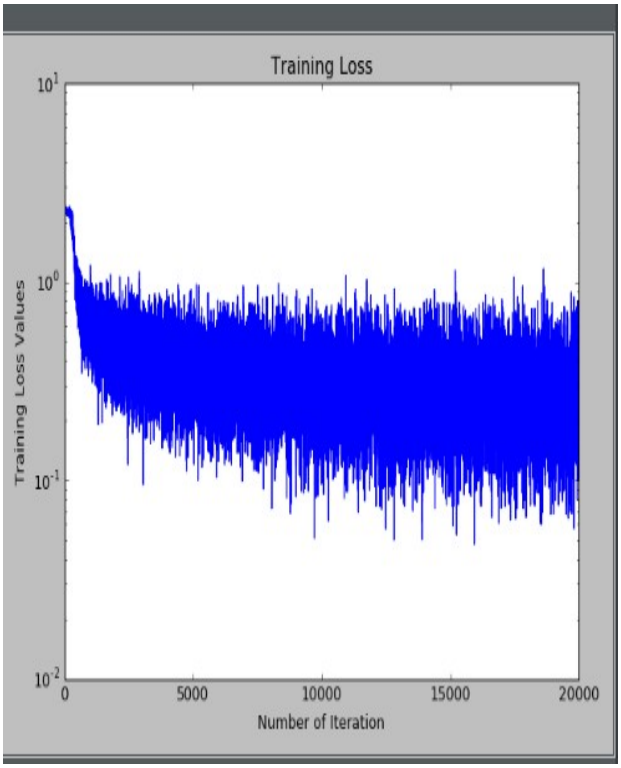


FIGURE1 BEST ARCHITECTURE CAFFE

This network architecture comprises of three convolution-relu-pooling layers. The batch normalization is applied after layer 1 and layer 2. A dropout layer is added to the fully connected layer with the dropout value of 0.5. This model uses Adam Optimizer with the base learning rate of 0.001.

The training loss and the model accuracy graphs are shown below:



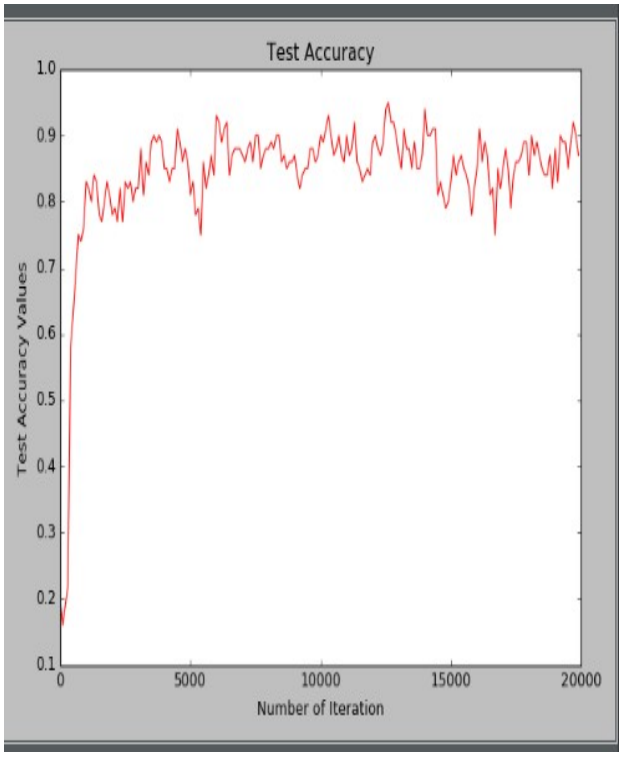


FIGURE 2 Training loss and Test accuracy

The model seems to be performing well. The model has achieved the accuracy of 91%.

Internet Code

I used some of the code for creating the lmdb database from internet and the preprocessing code. That code came from a blog mentioned in the references. The ratio is:

64/488 = 13.11%

References

[1] Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks (https://arxiv.org/abs/1312.6082)

[2] Blog - https://blog.csdn.net/yj3254/article/details/52370767