Machine Learning Engineering Nanodegree

Capstone Project

Kimish Patel
Date

I. Definition

Project overview

Domain of computer vision aims, in part at least, at teaching machines the ability to receive images as inputs and derive some semantic understanding of what they represent. This can include learning to recognize faces in the image, reading any text present in the image, object recognition and classification, image captioning etc. They have very many applications such as aiding the disabled, machine translation, and product categorization in supermarket and so forth. What this project aims to do is to take on a task of recognizing the house number from images obtained via street view. Even though the project specifically aims at recognizing house number, the approach and the framework can be extend to object recognition and classification, image segmentation etc.

The dataset of street view images containing house numbers is open source and is available at [1]. The dataset has variable sized (variable dimension) images, details of which will be given in the sections to follow. This dataset is divided in training and test and where each image has corresponding label that contains the following set of information: length of the house number, each digit of the number and position (co-ordinates in the image) and dimensions (width and height) of the bounding box corresponding to those digits. Along with this also comes digits dataset. This dataset contains training and test images of size 32x32 (width x height) and labels that contain: single digit number in the center of the image.

Problem Statement

The aim of the project is to develop a framework that learns from the training data and achieves reasonable accuracy (defined by the metric explained later) in predicting house number on the test images. More precisely the framework shall accept an image of arbitrary size, that contains some number, and give as an output the number contained in the image.

The general solution framework proposed here utilizes deep neural network. In particular I propose to use convolution neural network (CNN). Using this framework I propose three alternatives for the aforementioned task of recognizing house number.

1. Train CNN to recognize individual digit in a patch of an image. Once such a network is trained sweep the input image, with some stride, so as to extract the patches of image, which are fed to CNN to extract the digit in that patch if any. Once the all patches of the image are processed, post-process the extracted digits to output the house number.
2. Instead of training CNN on image patches, train it on the entire image. In particular this approach does the following:
    a. Train a CNN on entire image to recognize the number of digits in the house number contained in the image. The output of this CNN will be some number upto N, where N is the maximum length of a house number.
    b. For N, train N different CNNs each specifically trained to recognize some dth digit (where d=1,2,…,N) of N digit house number.
    c. Each of the CNN above will emit a probability distribution over set of classes. The first one will be over N classes, each containing the probability of the house number length being n ($n \in \{1,2,..,N\}$). The other N will be over 10 classes (corresponding to digits 0 to 9). The house number then is obtained by equation 1 shown below.
3. Train a single CNN that predicts house number length and each digit in one go. That is you do not have a separate network for each of the subtasks of approach 2 but instead train only one network to predict everything that approach 2 predicts as separate steps via separate networks.

Both approaches (2nd and 3rd) rely on the following formula to predict house number in this.

$$\boldsymbol{argmax_{\{over\ n \in \{1,..,5\}\}} \left\{ P(length = n|data) \prod_{d_n\ n \in \{1,..,5\}} max_{\{over\ D\ \in \{0,...,9\}\}} P(d_n = D|data) \right\}} \ldots \text{Equation (1)}$$

Here I have set N=5. Note that this approach is similar to what was done at [3].

Pros and cons of the aforementioned approaches:

First approach: pros: It is generalizable, cons: inference time may take longer as the image patch at each location must be expanded or shrunk to look for a digit, or in other words we must sweep the image with patches of different sizes

Second approach: pros: inference time does not rely on sweeping the image with patches of different size, cons: less generalizable to arbitrary house number length.

Third approach: pros: the network may be able to learn more because, for example, if it predicts that the house number should be 2 digits long then that knowledge can help improve the accuracy of predicting absence of a digit for 3rd, 4th and 5th position. Cons: likely such a network would have to be wider and deeper and will require long time to train.

Given that most of the house numbers are not expected to be longer than 5 digits, the simplicity of second and third approach was inviting for this specific task and I decided to evaluate both of them. From here on I will refer to the second approach as Distributed and third approach as Unified.

The following steps were taken to implement these approaches.

1. Download train and test dataset from the SVHN database.
2. Analyze the data, remove outliers if necessary.
3. Standardize the data.
4. Train CNN.
5. Test it on test data and then on some other data.

Metric

We will evaluate the performance of the aforementioned framework by evaluating the accuracy of the prediction. In other words how often can the framework figure out the house number in the image. More precisely accuracy will be defined as.

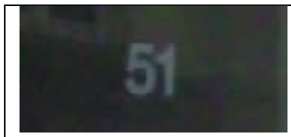
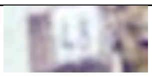
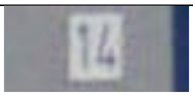
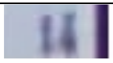
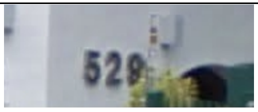
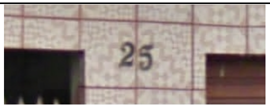
$$Accuracy\ (\%) = \frac{\text{\# of correctly classified images in test set} * 100}{\text{\# of images in the test set}}$$

Here correctly classifying an image would entail reading correctly the house number contained in the image. Furthermore we will also evaluate the accuracy of how often do we get just the house number length, first digit, second digit and so on correctly to understand how we do across individual category.

II. Analysis

Data Exploration and visualization

As mentioned earlier, dataset for street view house numbers (SVHN) is available at [2]. The dataset contains street view images of varied dimensions. Following table shows some of the training images with their respective dimensions.

 171 x 81	64 x 27	36 x 13	 54 x 25
169 x 84	38 x 14	66 x 32	 69 x 32
148 x 59	140 x 68	36 x 14	183 x 79
106 x 50	 42 x 20	 187 x 77	 311 x 119

As can be seen in the table sizes of these images vary widely. To get a better idea of what these sizes are, the following plot shows the histogram of heights and width of these images.

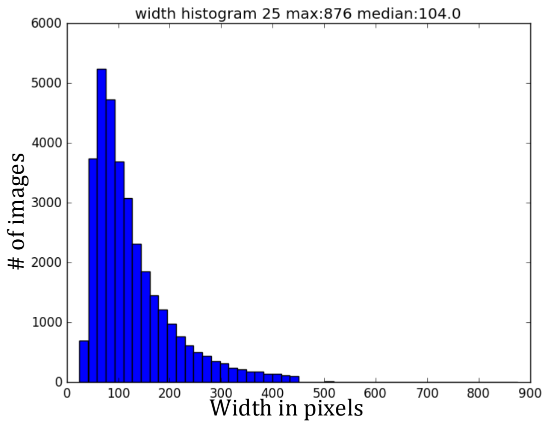


Figure 1 Image widths

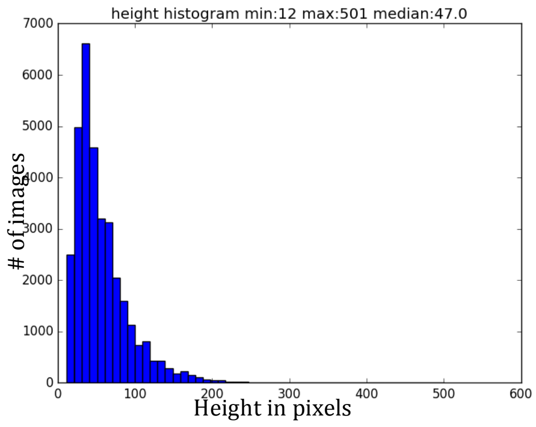


Figure 2 Image heights

Based on this, initially it was decided that the all the images would be scaled close to the median of their dimension. Thus the images were scaled to 120 (width) x 50 (height). However, later on I found that reducing this to 50x50 did not alter the accuracy significantly but it helped with computational complexity. Thus all the images were scaled to 50x50.

The subsequent step in understanding the data involved looking at the length of the house numbers and how they were distributed. During this step a sole outlier was found whose length was 6. This image simply contained the sequence '123456'. The outlier was removed. Remaining data analysis is without this outlier.

Following graph shows the distribution of house number length in both training and test data.

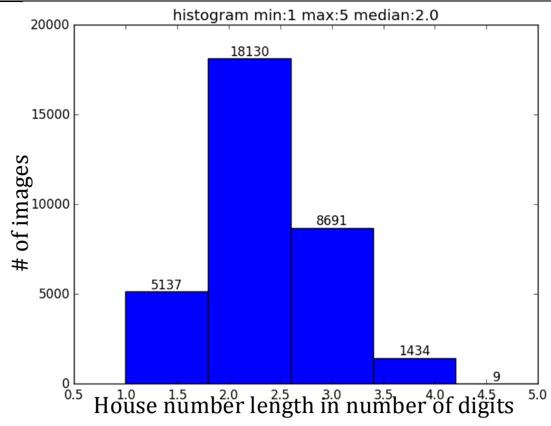


Figure 3 Training Data

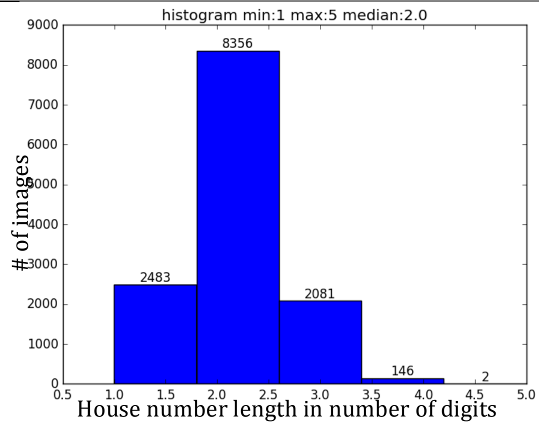


Figure 4 Test Data

As can be seen in Figure 3 for training data ~70% of the house numbers are of length 2 whereas for test data, Figure 4, ~83% of the house numbers are of length 2. On the other hand only 26% of the house numbers have third digit present in them in training data. This figure stands at 16% for test data. By the time we reach to the fourth digit this number drops to ~4% in training data and ~1% in test data. For the 5th digit we have a very few houses with house number that is 5 digit long.

Next I looked at the image data itself. Here I was interested in looking at distribution of pixel values. Note that the data below shows the grayscale values of pixel across entire dataset rather than R, G, B. Here I had reduced the number of channels from 3 to 1, as detailed in the data preprocessing section.

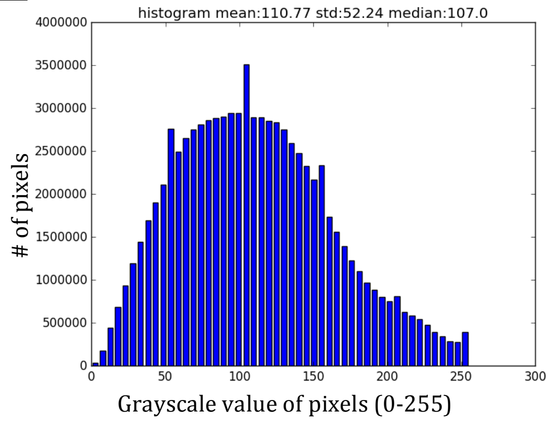


Figure 5 Training Data

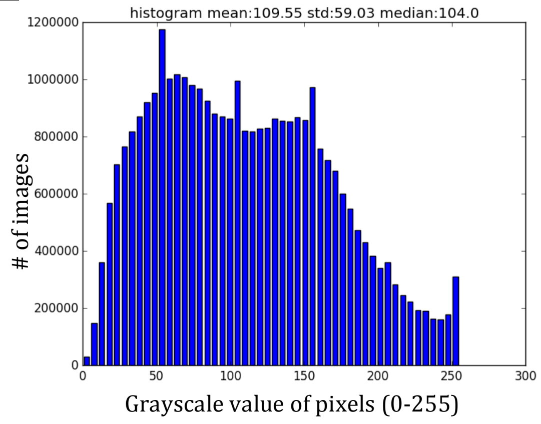
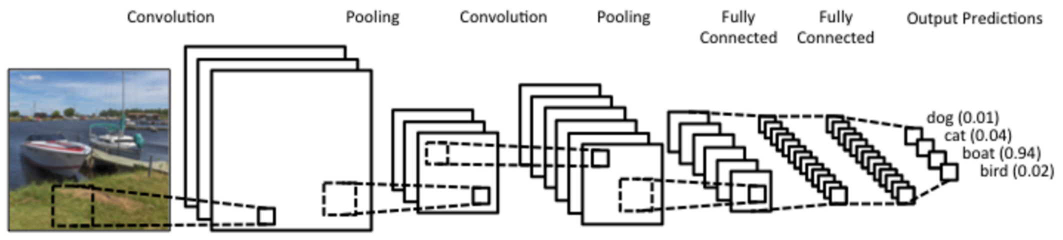


Figure 6 Test Data

What I observed from Figure 5 and Figure 6 is that the distribution of grayscale values of pixels across training dataset is very different from that of the test dataset. This can make learning very difficult because you show one characteristics of dataset during learning but get very a different dataset during test. To remedy this problem I, as will be shown in data preprocessing step, mixed training and test dataset, shuffled and divided it in the same proportion (so as to get the same number of training images and test images as before). To be sure, this was not just a theoretical assumption but also an observed behavior. During the first few iterations, I found that while training as well as validation accuracy improved, no improvement was found in test dataset. It was this observation that lead to suspicion that while we were learning features from training dataset, this features were not applicable to test dataset.

Algorithms and Techniques

In the domain of image processing convolutional neural networks (CNN) have been found to be very successful at extracting features from images and combining these extracted features, depending on the CNN architecture, in order to classify images or objects. So, what is convolutional neural network? The following diagram, borrowed from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/, shows example architecture of CNN based deep neural network.



As shown in the figure, convolution layer in CNN convolves over image or previous layer input using a 2D or 3D filter. For example in the first layer a 3D filter of size, say 5x5x3 (5 pixels by 5 pixels and depth of 3 channels), sweeps over image (across all three channels R, G, B of the image). This filter has 5x5x3=75 weights that are learned during training. The weights learned are supposed to extract certain features at different locations of the image. The features can be edges of different orientation or some primitive geometry etc. The output of one convolution layer can represent the presence or absence of certain feature at different locations of the image. During the latter layers the features extracted from earlier layers are convolved over using different set of filters. These layers combine features found in certain locations of the image, during convolution in previous layer, to provide some abstract representation of a new feature. At the end, these features, most often connected via fully connected layers, are used for classification. It is during training steps where the weights of filters and fully connected nodes are learned in a way that they represent some abstract features which when combined together aid in classification.

For its efficiency and prevalence I decided to use CNN based deep neural network framework to recognize the house number from street view images. This system differs from [1]. The generic approach is given below.

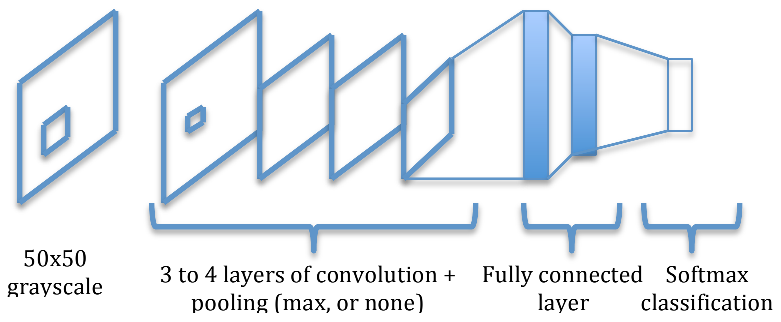


Figure 7 Convolutional Neural Network Architecture

The architecture depicted in Figure 7, forms the backbone of the framework of this project. Such a network is deployed for two tasks:

1. In Distributed approach it:
    a. Predicts the number of digits (from 1 to 5) in house number within the image and
    b. Predicts first, second, third and fourth digit (from 0 to 9) within the image. If any of the digit is not present, ideally, network should detect that.
2. And in Unified approach:
    a. One single network produces 6 sets of softmax classifiers in the very last later.
    b. The first one predicting the house number length
    c. And the rest predicting the digit at their respective location.

Each of the steps outputs softmax probabilities for each class. These probabilities are used as in equation 1 to predict the output with maximum probability.

Benchmarks

The known results available for this task are from SVHN paper at [1] and work done at google [3]. They achieve above 90% and above 95% accuracy on SVHN database. Note that in the case [1], the approach taken to the problem was different. Here authors chose to learn to recognize the digits individually first. Once this was done, their pipeline would first scan the image to find the rectangular box containing the digit and use their learnt model on the rectangle to predict what that digit is. On the other hand authors in [3] took the entire image instead of segmenting the image to predict a particular digit. The latter of the two approaches is closer to what I have done here, albeit without the vast resources and vast number of training images available at the disposal of Google.

III. Methodology

Data Preprocessing

As mentioned in the analysis section, I identified and removed one sample image that contained 6 digit house number '123456'.

Besides this, the preprocessing done to address two of the three issues mentioned in analysis section are as follows.

- Converting image to grayscale
- Image size standardization
- Mixing training and test dataset in order to remove differences between training and test data distribution
- Shuffle the set and divide it into training and test without changing the size of original training and test dataset.

Furthermore before the input was fed to CNN each pixel value was normalized between -0.5 and 0.5 via following formula.

$$newpixel_{value} = \frac{pixel_{value} - 127.5}{255}$$

The following pipeline shows the consolidated picture of the preprocessing:

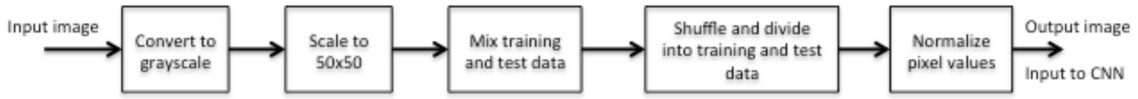


Figure 8 Data Preprocessing Pipeline

Implementation

The implementation of the proposed framework required development of three unit frameworks.

1) Preprocessing training and test data to generate a training/test combined dataset.
2) Deep learning module, which I called LayerCake, that allowed quick experimentation with various CNN architecture without modifying the code.
3) A wrapper framework that preprocesses data, feeds the deep learning module according to a config file and trains it.

Each of these three steps is detailed below.

Generating training and test dataset

Following diagram shows the modules that implemented the first two stages of Figure 8.

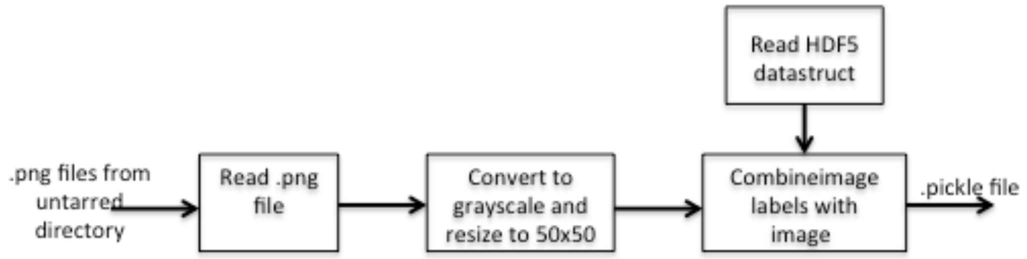

.png files from untarred directory
Read .png file
Convert to grayscale and resize to 50x50
Read HDF5 datastruct
Combineimage labels with image
.pickle file

Figure 9 Data Preprocessing Utilities

Deep Learning Module

The aim of this part was to develop a framework that allowed one to be able to read a config file describing a network and generate a tensorflow graph. One can stack up convolution layers followed by fully connected layers without having to change the code. Config file also allows you to play with hyper-parameters such as learning rate, regularization, choice of optimizer such as SGD, Momentum, or adagrad etc.

As shown in Figure 10, a network described in .json format is read into Network module. The Network module then feeds into LayerCake module via a wrapper. LayerCake module essentially reads network architecture slice by slice. Each slice can contain multiple layers. If a slice is of convolution type then ConvLayer module is invoked to construct convolution layers from network definition and return the output. If a slice is of fully connected type then DeepLayer module is invoked to construct fully connected layers. LayerCake module then connects all layers together and feeds input to the first layer and reads output off of the last layer.

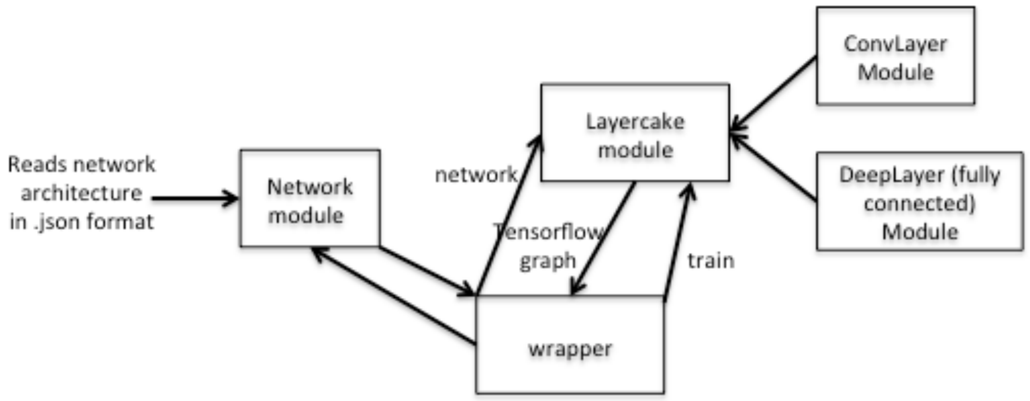


Figure 10 Deep Learning Framework

Figure 11 shows an example topology. As shown, it contains two slices. One of type convolution and the other of type fully connected. Each slice has multiple layers. Convolution slice layers are configured with different pooling and activation types. Fully connected layers are also configured with different activation types.

```
{
  "input_size": 0,
  "output_size": 0,
  "slice0": {
    "type" : "conv",
    "layers" : [[3, 3, 1, 64], [3, 3, 64, 96], [5, 5, 96, 128], [5, 5, 128, 256]],
    "activations" : ["relu", "relu", "relu", "relu"],
    "pooling" : ["none", "max", "none", "max"]
  },
  "slice1": {
    "type" : "fc",
    "layers" : [128, 64],
    "activations" : ["relu", "relu"]
  },
  "output": {
    "final" : "softmax",
    "loss" : "cross_entropy"
  },
  "learning_rate" : 0.005,
  "regularization" : 0.02,
  "droput": 0.90,
  "optimizer" : "Momentum",
  "num_iterations": 20001,
  "model_name" : "digit_1"
}
```

Figure 11 Example network topology in json format

Wrapper Module

This module wrapped around the previous LayerCake and Network modules. Furthermore it also implemented few utility functions responsible for the last stage of data preprocessing, shown in Figure 8.

Last, once all the models are trained, test version of this module generates softmax probabilities for each digit and number of digits for the training dataset. These probabilities are then combined as in equation 1 to determine length and digits of house number. The same framework is used for both Unified and Distributed approaches. The difference between them being the wrapper module and how many networks the wrapper module has to train. Five in the case of Distributed approach and one in the case of Unified approach..

A note on the computational complexity. Given the size of graphs generated by deep CNN it was impossible to use CPU based personal laptop with limited memory. This resulted in exploring Amazon's AWS framework and use of GPUs available on them. There was considerable amount of learning involved about AWS along with significant financial investment. However due to available GPUs the size and complexity of the problem became tractable. Another issue encountered during implementation was that of memory size. Even the P2 instances available on GPU came with only 12GB of memory. When a shallow network was implemented it was rather fine, but with deeper CNN the validation and test data could not fit in GPU memory. Tensorflow does not yet have good solutions to memory issues as was found via quick search. Thus batch processing of validation and test accuracy routines had to be implemented such that not entire validation set or test set would be fed to the trained network.

Refinement

Given how hard neural networks are to train, finding good hyper-parameters was crucial. I experimented with a few different architectures, simplest of which contained only fully connected layer. However, without convolution layer it was hard to get accuracy for determining even the number of digits in the house number. Generally the accuracy remained below 50%. With convolution layer it was possible to get accuracy close to 77% for determination of number of digits in the house number.

Once it was determined that convolution layers will be required, experimentation revolved around finding good network architecture and hyper-parameters mainly learning rate, regularization and dropout. These did help especially as network got deeper. It was soon evident that deeper network required slower learning rate otherwise either we will get stuck in local minima too soon or would start getting 'nan' results from optimizer. But even after the knobs adjustments, one issue persisted and that was of training and validation loss/error improving but not of test, as described in analysis section. When I finally looked at the data of Figure 5 and Figure 6, it was clear that little learning was going to happen, as the test and train data distribution was very different. Once this was fixed in the data preprocessing step test errors improved and matched that of validation.

After the aforementioned fixes when I trained the network, for Distributed approach, to predict the number of digits or to predict 1st or 2nd digits I saw continuous improvement. However when it came 3rd or 4th digits, as I was training network on each digit separately, I saw high accuracy right away. Especially with the 4th digit I would be getting close to 90% accuracy right away and final accuracy of 98%. This was rather strange as it was thought that the latter digits would be harder to predict. However when I went back to look at the graphs of Figure 3 and Figure 4, the reason became clear. Because I had such a large number of training examples where 3rd (74%) and 4th (96%) digits were not present, it was easy to learn to predict absence of a digit. However, the task of "really" learning to predict third and fourth digit will be considerably tougher because of the very same reason: too few positive example and too many negative ones. From these observations I took the following two steps to remedy the situation for Distributed approach:

1) For 3rd and 4th digit I did not use the entire training data. Instead I composed new training data from existing training data such that 50% of the new

training data has positive example, where there is a 3rd or 4th digit present, while the other 50% has negative example where there is no 3rd or 4th digit.

2) I entirely ignored 5th digit.

I did not have to do this preprocessing steps for Unified approach as I was training everything under one network.

IV. Results

Model Evaluation and Validation

Deriving the right kind of CNN topology was challenging. Factors beyond topology, such as learning rate, regularization rate, dropout based regularization, kind of optimizer used (SGC vs Momentum vs Adagrad) etc. had varying amount of impact on the results. Therefore for each task specified in section Algorithms and Techniques I tried few candidate network topologies with varying hyper parameters. Once I found a good network topology for each of those steps (predicting number of digits and predicting each digit in Distributed appraoch), I then varied hyperparameter space to fine-tune some results. Final network topologies are present in the repo under num_digit_networks, digit_1_networks etc. folders for Distributed approach, while svhn_network_entire_number.json corresponds to the network used for the Unified approach.

One of the things I learnt while training networks was that as training proceeded, beyond a certain point it was hard for the network to learn generalized feature. The validation accuracy was not getting worse however training kept improving and the gap kept widening. I tried simple as well as complex networks that my available computing power allowed and it did not alter the results much. This leads me to believe that more data and computing power to process more data could have yield better results.

Now let’s look at the accuracy in final predictions, that is how often can we predict the house number correctly. Here I have evaluated accuracy 1) on the test data available at SVHN website and 2) on the extra data available in the same place.

Table 1 Accuracy in predicting house number

	Test data		Extra Data		Random
Accuracy in %	Distributed	Unified	Distributed	Unified	Extra data
Number of digits	51.86	77.23	33.8	47.72	19.47
1st digit	58.76	58.84	34	33.51	10.05
2nd digit	40.42	49.96	18.82	26.38	9.45
3rd digit	73.98	76.27	42.09	43.31	30.72
4th digit	94.02	96.18	77	88.68	57.53
5th digit	0	-			
Entire number	23.4	33.2	6.4	9.6	0.5

First, performance is much better across individual category then for the entire house number. This is expected. Second, there is a big difference between results from Unified approach and Distributed approach. It seems that training everything together does give us an edge, as suspected earlier, where knowledge and mistakes from one part can be transferred to another to improve learning. Third, while Unified approach is better it does not perform as well as expected. However the test results do line up with what I saw for validation accuracy during training. This once again confirms my suspicion that while training error got quite a lot better validation/test did not and we did not have enough data to learn from (In fact limited by both data and computing power). Now when we go to see how well this generalizes by looking at extra data (20000 random images extracted from extra dataset available on SVHN website (extra.tar.gz)) column we see that it does not do that well. In fact accuracy across individual category as well as on entire number drops drastically. So what happened? We will come back to this little later in this section, but have we really learnt anything at all? How will we quantify this? For this I compared results against an approach that just randomly predicts the house number. It does that by first predicting the number of digits in the house number from 1 to 5 and after that for each of those digits it randomly predicts a number from 0 to 9. Last column in the table summarizes that result for the extra dataset. As we can see random approach does not really do well at all as it is hardly ever able to predict the house number correctly.

Furthermore looking at the actual outcome throws some interesting light. Following figure shows the misprediction on test data by both Distributed and Unified approach (First number in the label is the number of digits in the house number and subsequent number correspond to the particular digit of the number with -1 representing absent of a digit in that location).

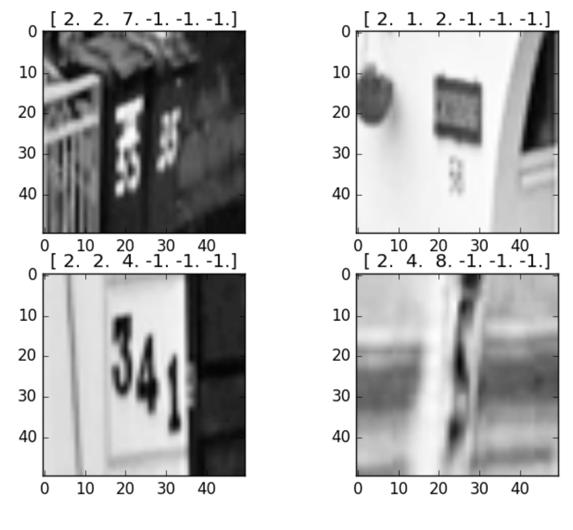


Figure 12 Midpredictions on Distributed Approach

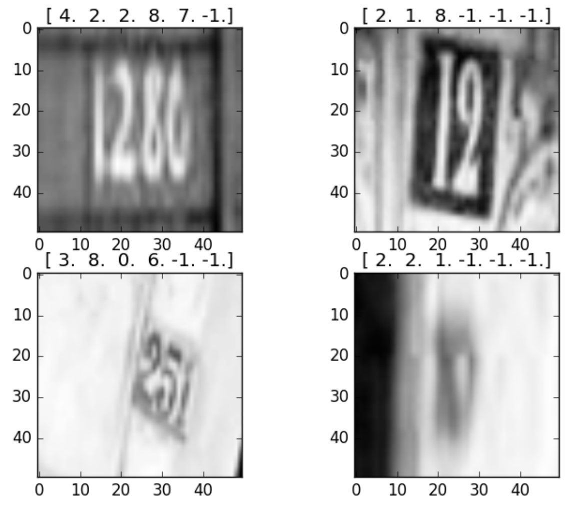


Figure 13 Mispredicts on Unified Approach

As we can Unified approach is usually able to get the number of digits right. Distributed on the other hand failed in 2 out of 4 images. Both usually get one of the digits right however overall it seems that Unified is better able to get more digits right even if it fails to get all of them right.

Following two figures show correct predictions on test data. As we can see Unified approach is able to get quite difficult predictions right as well compared to Distributed approach. But from some mispredicts example above we can see that the simple ones it is not able predict well as we mentioned earlier. We further examine this on the extra dataset.

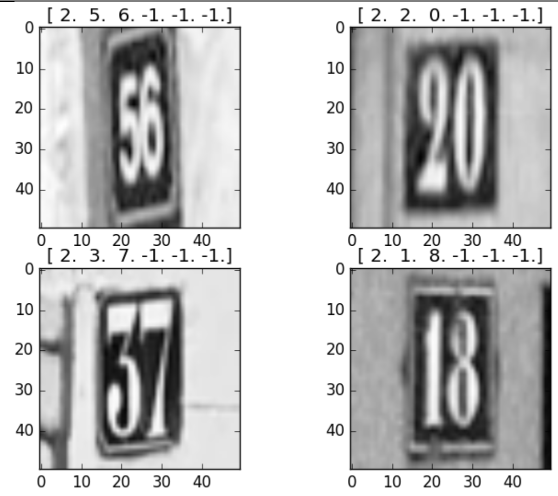


Figure 14 Correct Predictions Distributed Approach

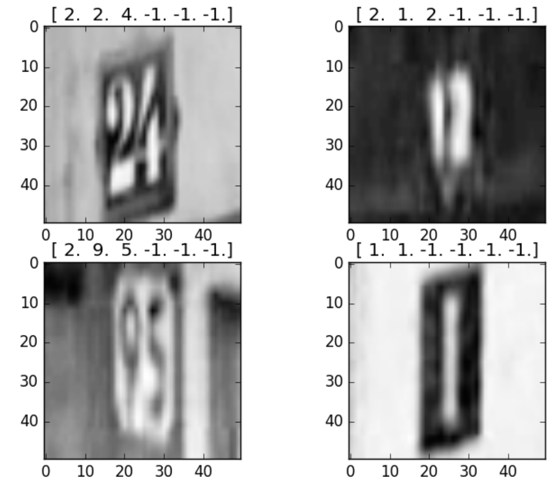


Figure 15 Correct Predictions Unified Approach

Following figure shows mispredicts on extra dataset.

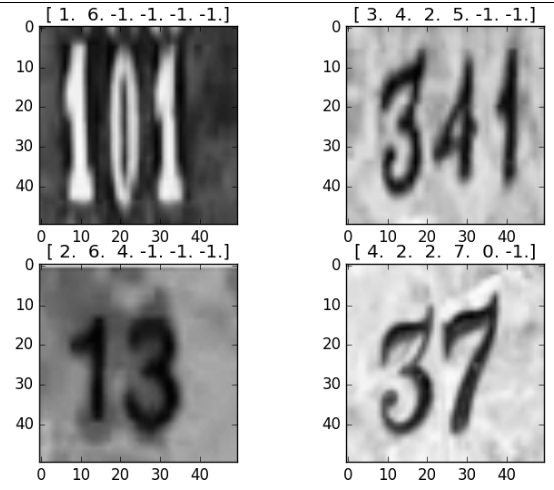


Figure 16 Mispredicts on extra dataset (Distributed)

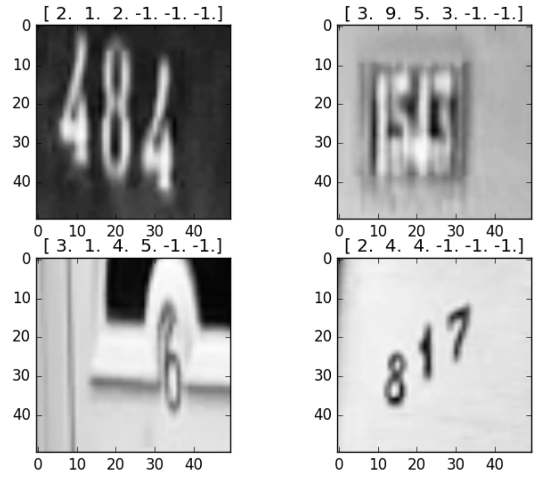


Figure 17 Mispredicts on extra dataset (Unified)

As we can see we are often getting the easy ones wrong, and in this case wrong by a large margin in most cases (sometimes we can figure out length sometimes we cannot). While the ones it is getting right are rather tough ones and are closer to the ones we have seen in our training set (Figure 18 and Figure 19).

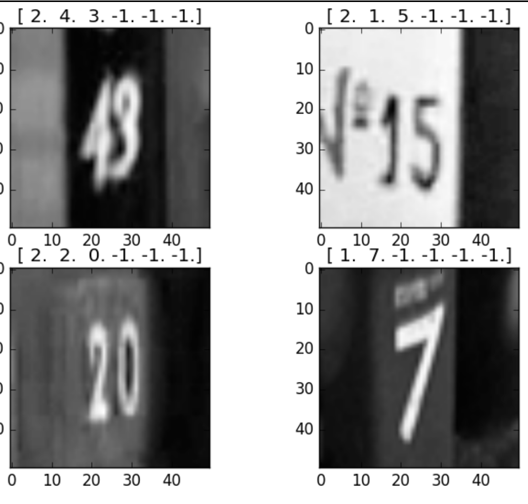


Figure 18 Correct predicts on extra dataset (Distributed)

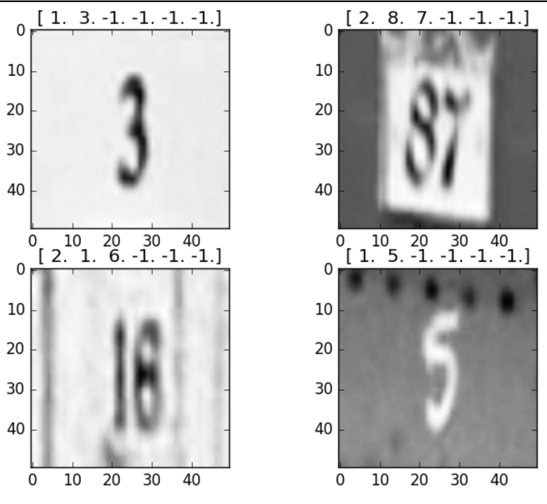


Figure 19 Correct predicts on extra dataset (Unified)

To make sure we were not overfitting I did run training with fewer iteration and stopped it early, but this did not alter results much. When I further dug into this to look at the distribution of the data I found once again that the distribution is different from the distribution we have seen in our training data. Following figures show that.

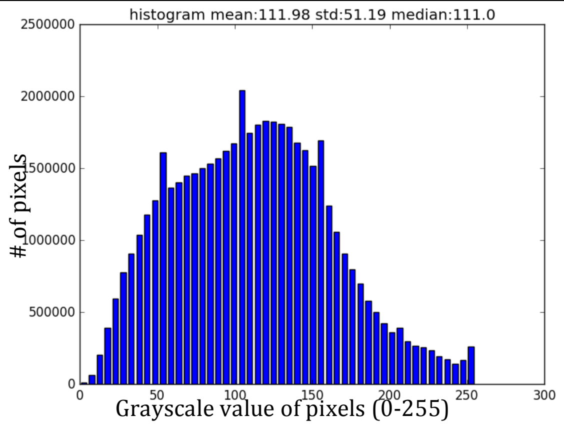


Figure 20 Extra data pixel value distribution

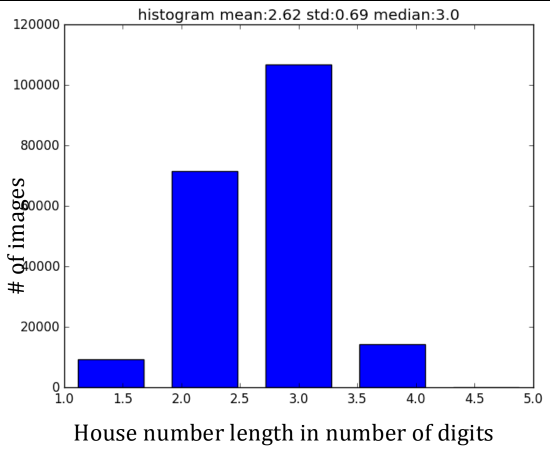


Figure 21 Extra data house number length distribution

As we can see images in the extra dataset that we used to quantify the generalization of our results show very different distribution, for pixel values as well house number length. It is important that our training sees variation of data that we expect to show it after training; however the data we trained our framework on saw very different distribution. I will reflect on this later and see how can this be addressed in future work that is more generalizable.

Justification

Results obtained through the framework of this project certainly do not match up with the benchmark of google that has obtained > 95% accuracy. But to ensure that the network has learned something I compared it against an approach that just randomly predicts the house number. Last column in Table 1 summarized the result. As we saw, random approach is barely able to get anything right at all. Through this experiment I just wanted to show that the framework has learnt something, even though on unseen data it has very low accuracy.

V. Results

Reflection

The project I undertook here required to solve one problem. Given a street view image containing house number, can you implement and train a machine-learning algorithm that is able to read the house number accurately. I approached this problem by developing convolutional neural network based framework. Given the problem, which is of computer vision domain, and CNN's effectiveness at handling computer vision problems I thought it was appropriate to employ CNN. In the process I first developed a baseline framework, called LayerCake, that can parse a network topology consisting of 0 or more layers of convolutions followed by 1 or more layers of fully connected network and return a tensorflow graph. The framework then can run training on such a graph given the input data and labels. I first employed this on SVHN digit database that contains cropped house number digits. On this, without significant tuning, I was able to achieve about 82% test accuracy. My initial plan was to use the network trained on the digit database to recognize house number digits in the images by searching the location of the digits and then running the trained network on the segmented image section containing the digit. This approach required predicting bounding box around each digit. My initial instinct was that this would be a tougher task. This led me to look for other approaches employed for the same problem. In the process I came across google paper [3] that employed single neural network to predict digits instead of segmenting the image. In particular their approach of predicting length of the number and digits separately and combining them to give final prediction, as shown in equation 1, seemed particularly appealing. This is the approach I decided to follow.

Subsequently I obtained the house images and labels from SVHN website [2] and resized them to 50x50 and converted them to grayscale. Following this, using the LayerCake framework, I experimented with various network topologies, but every time training and validation accuracy would improve but test accuracy did not change. This threw in the first challenge. Following this I looked at pixel value distribution presented earlier. This indicated that the distribution training was working with was different than test and this would make it hard to learn. When I mixed, shuffle and split training and test data, the test accuracy was more in line with what I saw in validation.

The second issue I encountered was in relation with 3rd and 4th digit. Here training immediately gave better result and when I looked at the distribution of house number length, it became apparent why. Because there was only about 25% samples where 3rd digit was present network was able to achieve 75% accuracy right away, as it was easy to predict absence of a digit. I fixed this by recombining data in a way that 50% of the sample contained a digit and 50% did not. This helped train the network better. Note this particular preprocessing was done only for the Distributed approach.

Besides this, the most difficult aspect of the project was trying to improve validation accuracy. Validation accuracy seemed to hit ceiling around 77%, for house number length, even after trying various networks including shallower to deeper. I also tried different batch sizes, learning rates, regularization and optimizers. After a while training will reach 100% accuracy while validation will be stuck around 77%. My conclusion of this was that the amount of training data available is not large enough to extract new features that would generalize more easily. Due to lack of time I was not able to invest time in extracting more data to have it generalize better.

Another challenge that I found was employing AWS. The compute resources I had available at my disposal were not enough to carry out complex network training. This led me to explore the world of AWS that turned out to be a great resource. I was surprised at how easy AWS was for acquiring resources. However there was a significant amount financial investment in this due to expensive GPU resources.

In final conclusion, I would think that this model, at least Unified one, had met some expectation but accuracy was far below what I was hoping for. I was at least able to narrow down the reasons behind it which were sources of great learning.

Improvements

There are couple of improvements I can see making for this framework.

One is that of gathering more data and confirming my suspicion that it was because of the limited data that the network was not able to extract interesting features that generalized and improved validation accuracy.

Another is that of employing a more generalized approach. I realized that a more general approach to the problem would be to train network to recognize individual digits. This would have to be followed by image segmentation learning that can generate bounding box around digits in the image and the learnt network can predict digits contained in the bounding box. Such an approach would be able to generalize in not only recognizing house number of arbitrary length but also across any form of digit recognition. This I found was the biggest shortcoming of my framework and in future, time permitting, I would like to address this.

VI. References

1. 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*
2. *http://ufldl.stanford.edu/housenumbers/*
3. Goodfellow, Ian J., et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." arXiv preprint arXiv:1312.6082 (2013).