Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

E4040.2017Fall.STFU.report

Arman Uygur au2205, Jonathan Galsurkar jfg2150, Nitesh Surtani ns3148
Columbia University

Abstract

This paper addresses the problem of recognizing house numbers, comprising of series of digits, from natural Street View images. The paper follows the unified approach [2] using CNNs. After multiple experiments, we achieve a performance accuracy of 95.59% in recognizing the correct sequence of house numbers from images.

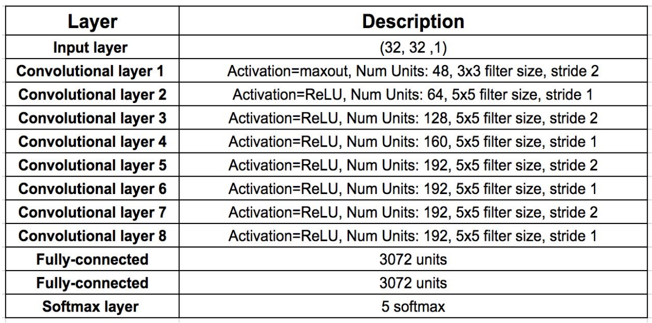
1. Introduction

Recognizing text of arbitrary length in natural photographs is a hard problem. The original paper addresses an equally hard sub-problem of recognizing house numbers, comprising of series of digits, from natural Street View images. These images are taken from the Street View House Numbers (SVHN) dataset, which contains real-world images of street numbers annotated with bounding boxes and labels. Some traditional approaches employ separating out the localization, segmentation, image feature extraction, and recognition steps using various ML models, but we follow the unified approach presented in [2,3,4,5] using Convolutional Neural Networks (CNNs). We initially implement the CNN architecture presented in the original paper, but quickly realize some shortcomings in terms of data and resources available. We address these issues through one slight parameter change in the model, which was the high dimension of fully connected layer. We further explored the possibility of using a smaller network to get comparable performance. Similarly, we explored the use of a larger network in an attempt to achieve better accuracy. Given the computational and architectural configurations of our experiments, we are unable to replicate the performance closer to the original paper.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The original paper employs a unified approach for digit recognition instead of separating out the localization, segmentation and recognition steps traditionally used. We attempt to replicate the Convolution Neural Network described on the SVHN dataset.



Layer	Description
Input layer	(32, 32 ,1)
Convolutional layer 1	Activation=maxout, Num Units: 48, 3x3 filter size, stride 2
Convolutional layer 2	Activation=ReLU, Num Units: 64, 5x5 filter size, stride 1
Convolutional layer 3	Activation=ReLU, Num Units: 128, 5x5 filter size, stride 2
Convolutional layer 4	Activation=ReLU, Num Units: 160, 5x5 filter size, stride 1
Convolutional layer 5	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 2
Convolutional layer 6	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 1
Convolutional layer 7	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 2
Convolutional layer 8	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 1
Fully-connected	3072 units
Fully-connected	3072 units
Softmax layer	5 softmax

Figure 1. Architecture of Original CNN model

2.2 Key Results of the Original Paper

The original paper achieves the best performance accuracy of 96.03% on multi-digit transcription task. They use confidence thresholding, by which they obtain 98% accuracy with 95.64% coverage. The paper also reports that increasing the number of layers in the network improved the performance of the model every time, achieving the best accuracy with 11 layers.

3. Methodology

In our initial attempt, we replicated the model presented in the paper, using the same suggested architecture. But the paper doesn't explicitly mention the dropout rate at each layer, so we experimented with various dropout rates, learning the importance of this parameter in terms of model convergence. We ended up achieving decent performance with just over 91% test accuracy.

We implemented many different CNN architectures and explored various parameters. This includes changing the number of layers, number of fully connected units, and other various parameters such as the learning or dropout rate.

3.1. Objectives and Technical Challenges

Our goal is to predict with high accuracy, the complete sequence of digits in Street View House Number images, performing as close as possible to the original paper.
The first set of technical challenges we encountered were the limited resources available to us:

1. The original paper is able to generate and use several times more data by randomly augmenting images. This is due to the high computational resources available to the original authors. We on the other hand, have limited RAM, not allowing us to load the data of comparable memory size[1].

2. The next limiting resource is the computing power. The original paper uses DistBelief network to train their large Neural Networks, which enables the author to train the complex model on the larger dataset for many iterations (6 days of training). Due to the constraints on our GPU resources, the experiments required much higher training time, even on a smaller dataset and with limited GCP coupons, we would not be able to train for 6 days as done in the original paper.

3. The paper did not include all the model configuration details. For instance, the dropout rate was crucial in the model performance. The paper had no mention of the values of this parameter that they used. This made it difficult for us to replicate their model exactly and required significant trials to tune the model. Some other parameters, like the learning rate etc. were also not described in the paper.

3.2. Problem Formulation and Design

It is important to note that this problem is essentially a sequence recognition task. Each SVHN datapoint is an image with a sequence of numbers, the order of which is obviously important in this case. Our system should be able to identify those numbers in the correct order. To compute the accuracy, we compute the proportion in which the overall sequence matches the given image. Improper ordering or any character incorrectly matching in the sequence is considered incorrect. We implement the same approach taken in the original paper, which essentially trains a probabilistic model of sequences given the images.

The probabilistic model basically attempts to compute the posterior distribution of sequence, given the set of images.

$$P(\mathbf{S} = \mathbf{s}|X) = P(L = n \mid X)\Pi_{i=1}^{n} P(S_i = s_i \mid X)$$

where *S* represent the output sequence and *X* represent the input image. Our goal is then to learn a model of *P(S|X)*. Each of the variables in the sequence is discrete and independent and can take values only between 0-9. The length of the sequence also has small number of possible values, i.e. 0-5 (we remove sequence with length "more than 5")

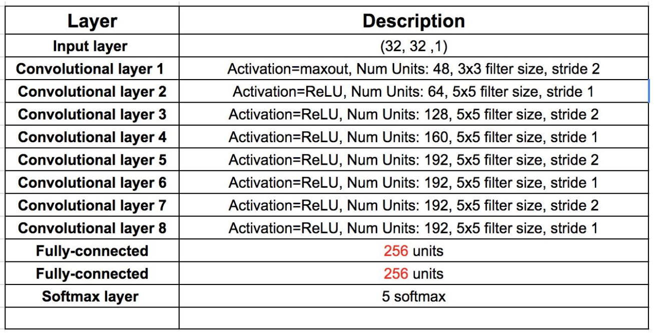
4. Implementation

We experimented with various network architectures and parameters both described in the paper and explored on our own. The following are architectural details.

4.1. Deep Learning Network

We implement the 10-layer model described in the paper, with the following architecture:

[1] We have used ~100 GB RAM, 16 CPU and 2 GPU configuration. In some simulations, we encountered "ResourceExhaustedError" which was caused due to insufficient space in the memory.



Layer	Description
Input layer	(32, 32 ,1)
Convolutional layer 1	Activation=maxout, Num Units: 48, 3x3 filter size, stride 2
Convolutional layer 2	Activation=ReLU, Num Units: 64, 5x5 filter size, stride 1
Convolutional layer 3	Activation=ReLU, Num Units: 128, 5x5 filter size, stride 2
Convolutional layer 4	Activation=ReLU, Num Units: 160, 5x5 filter size, stride 1
Convolutional layer 5	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 2
Convolutional layer 6	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 1
Convolutional layer 7	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 2
Convolutional layer 8	Activation=ReLU, Num Units: 192, 5x5 filter size, stride 1
Fully-connected	256 units
Fully-connected	256 units
Softmax layer	5 softmax

Figure 2. Architecture of our Best Performing CNN model

After having sub-par results (50% validation accuracy), the reasons for which were described earlier (fc unit size & dropout rate), we used the same architecture with different parameters, most importantly, changed the fully-connected hidden dimension size, which gave a significant boost in accuracy. We later also achieved similar accuracy with fine tuning of the dropout rate. We then pondered on the use of a smaller model, resulting in the following architecture.

For training, we used the adam optimizer, saving the model with the highest validation accuracy after every 100 iterations. We have experimented with different values of learning rate, with different models performing best at different learning rates.

Data Used: We have performed experiments on the SVHN multi-digit dataset. The datasets consists of a total of 248,823 images, which is organized as 33,404 training, 13,404 testing and 202,355 extra images, which are relatively easier to recognize. We create two datasets for our experiments using this data:

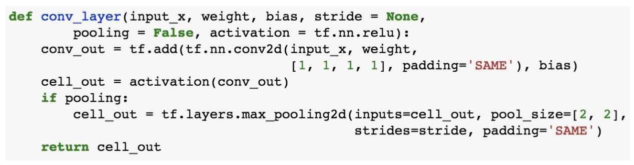
This data contains all the images from all the 3 sets. We combine the training and extra images (*C)* to create the training (*T*) and validation set (*V)*, by taking 3,000 images randomly from training and 2,000 images randomly from the extra set to form the validation set (*V)*, while all others are used for training *(T).*

We do not have access to the other two data sources form the original paper, the internal Google street view data and ReCaptcha data. Therefore, we will present our findings only on the SVHN dataset.

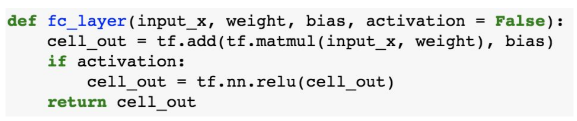
Data Preprocessing: The processing step loads the bounding boxes for each train, test and extra set, and marks the bounding box for each digit. Then, we crop the image around the bounding box that contains all of the digits. We perform this operation by selecting the box with the smallest area for each digit in X-Y coordinate-space. We excluded images with more than 5 digits, which is only 1 data point for this dataset. We then subtract the mean from every image and store the images in a HDF5 file.

4.2. Software Design

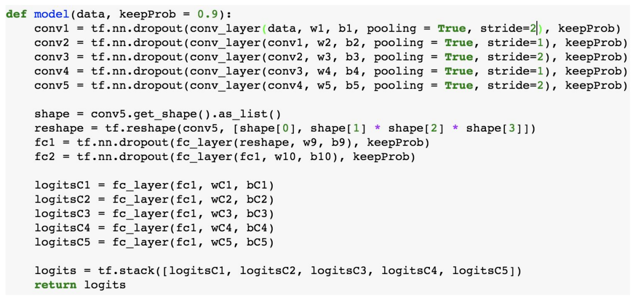
We have tried to keep our code modularized and highly reusable. The convolution layer function returns a single convolution cell, which forms the building block for the complex model. All the common operations such as Max Pooling and various parameters like activations can easily be passed as function parameter.



```
def conv_layer(input_x, weight, bias, stride = None,
        pooling = False, activation = tf.nn.relu):
    conv_out = tf.add(tf.nn.conv2d(input_x, weight,
                                   [1, 1, 1, 1], padding='SAME'), bias)
    cell_out = activation(conv_out)
    if pooling:
        cell_out = tf.layers.max_pooling2d(inputs=cell_out, pool_size=[2, 2],
                                           strides=stride, padding='SAME')
    return cell_out
```



```
def fc_layer(input_x, weight, bias, activation = False):
    cell_out = tf.add(tf.matmul(input_x, weight), bias)
    if activation:
        cell_out = tf.nn.relu(cell_out)
    return cell_out
```



```
def model(data, keepProb = 0.9):
    conv1 = tf.nn.dropout(conv_layer(data, w1, b1, pooling = True, stride=2), keepProb)
    conv2 = tf.nn.dropout(conv_layer(conv1, w2, b2, pooling = True, stride=1), keepProb)
    conv3 = tf.nn.dropout(conv_layer(conv2, w3, b3, pooling = True, stride=2), keepProb)
    conv4 = tf.nn.dropout(conv_layer(conv3, w4, b4, pooling = True, stride=1), keepProb)
    conv5 = tf.nn.dropout(conv_layer(conv4, w5, b5, pooling = True, stride=2), keepProb)

    shape = conv5.get_shape().as_list()
    reshape = tf.reshape(conv5, [shape[0], shape[1] * shape[2] * shape[3]])
    fc1 = tf.nn.dropout(fc_layer(reshape, w9, b9), keepProb)
    fc2 = tf.nn.dropout(fc_layer(fc1, w10, b10), keepProb)

    logitsC1 = fc_layer(fc1, wC1, bC1)
    logitsC2 = fc_layer(fc1, wC2, bC2)
    logitsC3 = fc_layer(fc1, wC3, bC3)
    logitsC4 = fc_layer(fc1, wC4, bC4)
    logitsC5 = fc_layer(fc1, wC5, bC5)

    logits = tf.stack([logitsC1, logitsC2, logitsC3, logitsC4, logitsC5])
    return logits
```

Figure 3. Snippets to demonstrate modularization of code

The above snippet of code presents a small part of our CNN model. The above model comprises of 5 Convolution layers and 2 Fully Connected layers.

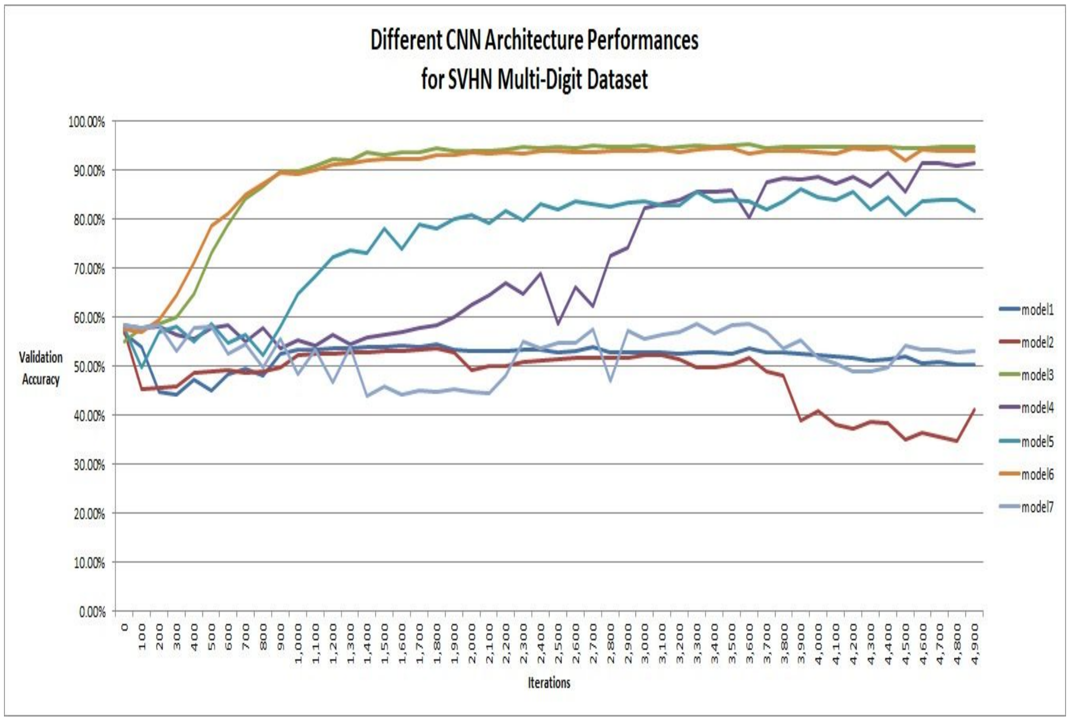
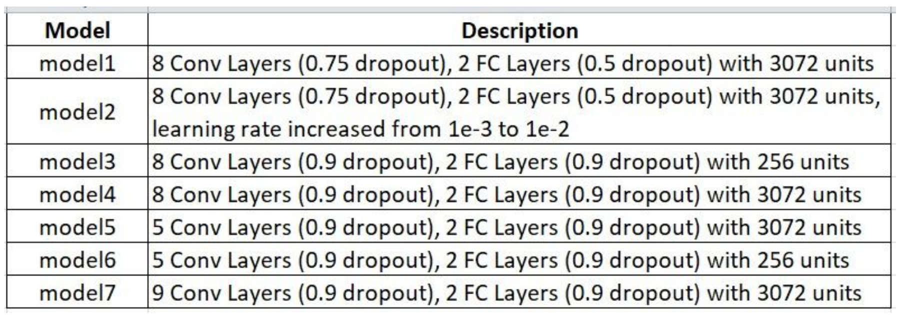


Figure 4. Performance of different models on SVHN dataset



Model	Description
model1	8 Conv Layers (0.75 dropout), 2 FC Layers (0.5 dropout) with 3072 units
model2	8 Conv Layers (0.75 dropout), 2 FC Layers (0.5 dropout) with 3072 units, learning rate increased from 1e-3 to 1e-2
model3	8 Conv Layers (0.9 dropout), 2 FC Layers (0.9 dropout) with 256 units
model4	8 Conv Layers (0.9 dropout), 2 FC Layers (0.9 dropout) with 3072 units
model5	5 Conv Layers (0.9 dropout), 2 FC Layers (0.9 dropout) with 3072 units
model6	5 Conv Layers (0.9 dropout), 2 FC Layers (0.9 dropout) with 256 units
model7	9 Conv Layers (0.9 dropout), 2 FC Layers (0.9 dropout) with 3072 units

Table 1. Description of models presented in Figure 4

5. Results

5.1. Project Results

Using the architecture and model parameters from the original paper, we achieve up to 91.8% test accuracy. Based on trends in the reduction in loss and increase in validation accuracy, we could say that the model could perform much better if trained longer.

As mentioned previously, certain parameter values such as dropout's keep probability were not mentioned in the original paper. We experimented with a few values. With a dropout of 0.75 on convolutional layers and 0.5 on FC layers, we did not see an increase in validation accuracy. Of course, with 3072 hidden units in the fully connected layers, a lot more training time would be necessary. If the keep probability was increased to 0.9 as in model 4, the model improved significantly. We posit that convergence was not occuring in the first 5000 iterations due to small learning rate. We used 10x the learning rate in model 2, however did not see any improvement over model 1.

In model4, we decided to change the dropout rate to 0.9 while keep the same architecture as previous models (1 and 2). We started to see some improvement in the overall validation accuracy. Knowing that the original paper's model took 6 days to converge, we decided to simplify the model by reducing number of units in FC layers to 256 - model 3. This performed the best among 7 models that we are comparing, with achieving an 95.59% of test accuracy.

We then tried larger and smaller models to see how the test accuracy is affected by this change. The larger model (9 Conv Layer, model 7) did not seem to converge, which probably means either more time to train (most likely) or needs higher learning rate is necessary. In one of the small models such as model 6, we found that we could achieve similar results (e.g. 94% of test accuracy). This significantly boosted the efficiency in training time, the entire training taking only 21.6 minutes[2], while bigger models were taking more than 12 hours.

5.2. Comparison of Results

The original paper achieved 96% accuracy after a 6-day training on the model. With less computational resources and for efficiency, we decreased the fully connected layer number of units to 256. As can be seen from figure 4, Model 3 was the best achieving model among those we tested, and it yielded a test accuracy close to the one mentioned in the original paper. We believe that, with enough time and resources, our implementation of the original model (with FC units = 3,072) would converge to an accuracy around 96%.

5.3. Discussion of Insights Gained

As has proven to be the same countless times, more data and more training in general leads to better results. With less computational resources, we achieved around 1% less accuracy than the original paper model, as expected due to the less training time, data, and units in the fully connected layers. On the other hand, we were able to achieve comparable accuracy with a smaller network, which implies less training time and a smaller dataset. The original paper also mentioned that the data augmentation only provides an increase of .5%, showing that we receive close to optimal performance given that this augmentation is not performed. This shows the power of proper parameter selection and tuning.

[2] For 2 GPU, 16 CPU and ~100 GB RAM configuration

6. Conclusion

In this paper, we have attempted to solve the problem of automatically recognizing the sequence of digits of house numbers, using the Street View House Number (SVHN) dataset. We replicate the model presented in the original paper, and address various bottlenecks in handling the high complexity of the model. We achieve the best performance with less dense FC network, attaining the performance accuracy of 95.59%.

6. Acknowledgement

We are thankful to the instructor and the TAs to devote their valuable time and answer our useful questions on Piazza.

7. References

[1]https://bitbucket.org/ecbm4040/2017_assignment2_ns3148
https://bitbucket.org/ecbm4040/2017_assignment2_jfg2150
https://bitbucket.org/ecbm4040/2017_assignment2_au2205
[2] Goodfellow, Ian J., et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." *arXiv preprint arXiv:1312.6082* (2013).
[3] Matan, Ofer, et al. "Multi-digit recognition using a space displacement neural network." *Advances in neural information processing systems*. 1992.
[4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
[5] Ba, Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple object recognition with visual attention." *arXiv preprint arXiv:1412.7755* (2014).