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

E4040.2019Fall.ZXXZ.report
rx2166.wz2466.yz3075
Runkun Xie rx2166, Wei Zhuo wz2466, Yifei Zhao yz3075
Columbia University

Abstract

Recognizing multi-digit numbers with varying length from Street View images is a difficult problem. In this paper, we apply deep convolution neural network (Ian et al., 2013) to raw street view images directly. Due to the limitation of computational resources, we trained our model on part of the Street View House Numbers (SVHN) dataset, and our best model achieves 86.02% accuracy on the test dataset.

1. Introduction

Recognizing multi-digit numbers from street view picture is an important area in the research of deep learning, computer vision and online map producing industry. One important application of this technique is to identify the multi-digit numbers from the pictures of Google's Street View, which contains tons of geo-located images.

The original paper is indeed focused on studying multi-digit numbers from street view panorama images. Specifically, they applied a unique Convolutional Neural Network model on the Street View House Numbers (SVHN) datasets, and received 97.84% accuracy. One essential improvement of the original paper is that they did not apply recognition on the cropped images, which contain only the numbers from these images and discard the street views, and they instead recognize multi-digits number using uncropped street images directly.

In our work, we strive to reproduce the result of original paper using the same convolutional neural network structure and evaluating them on the SVHN datasets. The challenges are, however, the original paper suggests using millions of street view images as training data on a convolutional neural network with relatively deep structure, and therefore takes around 6 days for the model to converge to a satisfy result. Due to the limitation of computational resources, we decided to shrinkage the number of images and training sets to around 20% as to the original work suggest.

2. Summary of the Original Paper

In this section, we provide the methodology and key results of the original paper. The original paper inferred street view numbers using convolutional neural network, and achieved 96.03% accuracy on Street View House Numbers (SVHN) dataset.

2.1 Methodology of the Original Paper

The task of the original paper is a specific kind of question in sequence recognition. In detail, they are given an image of street view pictures which contain a number, and intend to identify the number in the image.

The approach of the original paper is to train a probabilistic model of sequences given input street view images. In other word, they train a model which takes an street view image as input, and output the inferred numbers produced by the model. Specifically, the sequence transcription model output the length of numbers, as well as the number of each digit in the images.

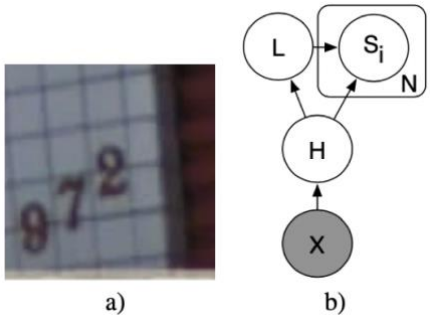

Figure 1: a) An example of input street view image, the correct output of this image is 972. b) The diagram of the sequence transcription model structure in the original work.

The model suggested by the original paper is a self-designed convolutional neural network. The overall structure of the network contains eight convolutional hidden layers, two fully connected layers, and one partially connected layer.

2.2 Key Results of the Original Paper

The model trained the convolutional neural network on public Street View House Numbers (SVHN) dataset, which contains around 200k street numbers, and about total 600k digits. After training, the best model of the original paper reaches 96.03% sequence transcription accuracy.

3. Methodology

We replicate the convolutional neural network structure of the original work, and train the network on partial Street View House Numbers (SVHN) dataset. In this section, we provide the objectives, challenges, and a detailed

description of the street view sequence transcription problem and our convolutional neural network model used.

3.1. Objectives and Technical Challenges

Since we are using the same network structure as the original work, we hope to achieve the same accuracy. However, the whole SVHN dataset contains 200 thousand images and may take a considerably long time to train the network.

Our solution is to use only partial of the SVHN dataset. Specifically, the SVHN dataset contains three parts – train, test, and extra – and we use train and test data to develop our neural network model.

3.2. Problem Formulation and Design

The street view sequence transcription problem starts with the input image, we take an image as input and output the correct number. In our paper, we use S as output sequence, X as input image, and H as out network model. In example Fig. 1a, the number we intend to identify is 972, so we want to produce $s = s_1, s_2, \dots, s_n$, where $s_1 = 9, s_2 = 7, s_3 = 2$.

To model S, we define S as a series of numbers S_1, $S_2, \dots$, S_N, where N is fixed to be 5. This is because most of the street view images contain number less than 5 digits. We also model the length of number by L, where L can be one of the 7 values – 0, 1, 2, 3, 4, 5, and "more than 5". Therefore, we have the probabilistic model:

$$P(S = s|X) = P(L = N|X) \prod_{i=1}^{n} P(S_i = s_i|X)$$

and our goal is to maximize the log probability on the training set to produce a well-functional convolutional neural network.

The structure of the convolutional neural network model consists eight convolutional layers, two fully connected layers, and one partially connected layer for each output. The outputs of this network would be the length of number L, and 5 digits S_1, $S_2, \dots$, S_5. If the length of the number n is less than 5, then the output number is S_1, $S_2, \dots$, S_n.

4. Implementation

In this section, we cover the model implementation. First, we give you a brief introduction to the dataset we use and how we preprocess our dataset. Then we talk about the architecture of our model, the parameters we use, and how we calculate the loss. After that, we walk you through the detail of our training process, as well as how we evaluate our training result.

4.1. Deep Learning Network

Here we would like to start with two figures to give you an idea of how we formulate this problem, and how we construct the architecture of our model.

The logic of our implementation can be summarized with this simple flow chart.

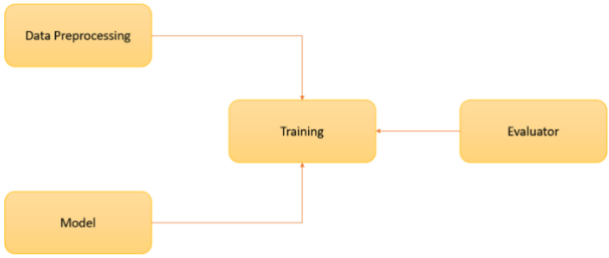

Figure 2: Implementation flow chart

The architectural block diagram is as follow.

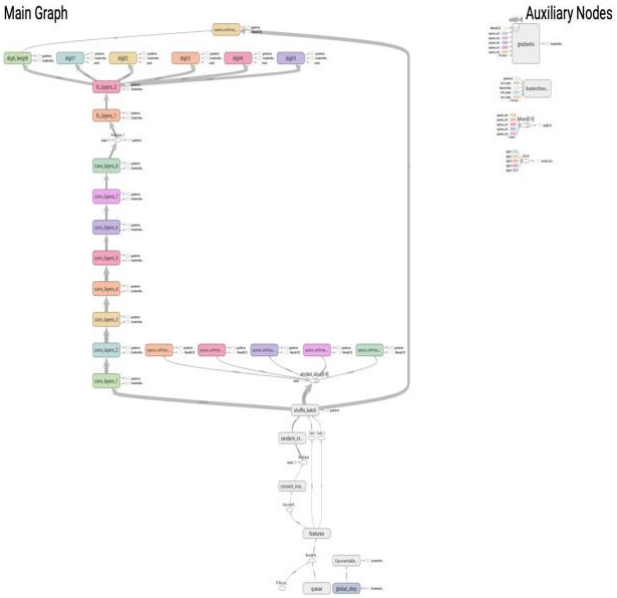

Figure 3: Convolutional neural network diagram

4.2. Software Design

In this part we give you a more detailed description of the algorithm we use and also present you with some of the code.

First let's take a look at the data we use. The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) is a dataset of about 200k street numbers, along with bounding boxes for individual digits, giving about 600k digits in total. In our model, we have 33,402 training samples, 3,068 validation samples, and 10,000 test samples. The class DataManager is used to preprocess the dataset. We preprocess the dataset in a way that is suggested in the paper – first we find the smallest rectangular bounding box that will contain individual character bounding boxes. We then expand this bounding box by 30% in both the x and the y direction, crop the image to that bounding box and resize the crop to 64 * 64 pixels. After that, we crop a 54 * 54 - pixel image from a

random location within the 64 * 64 - pixel image. This means that we generate several randomly shifted versions of each training sample for data augmentation.

Please see the following code for data preprocessing and augmentation.

```
@staticmethod
def expand_and_crop(image, bbox_left, bbox_top, bbox_width, bbox_height):
    cropped_left, cropped_top, cropped_width, cropped_height = (int(round(bbox_left - 0.15 * bbox_width)),
                                                                int(round(bbox_top - 0.15 * bbox_height)),
                                                                int(round(bbox_width * 1.3)),
                                                                int(round(bbox_height * 1.3)))
    image = image.crop([cropped_left, cropped_top, cropped_left + cropped_width, cropped_top + cropped_height])
    image = image.resize([64, 64])
    return image
```

```
@staticmethod
def augmentation(image):
    logger.info('--- augmenting images ---')
    image = tf.image.convert_image_dtype(image, dtype=tf.float32)
    image = tf.reshape(image, [64, 64, 3])
    image = tf.random_crop(image, [54, 54, 3])
    return image
```

Figure 4: Data preprocessing pseudo codes

In this project, we want to recognize both the length and digits. As is said in the paper, very few street numbers contain more than five digits, so we can assume the sequence length n is at most 5. We supplement the numbers with digits 10 so that each number has 5 digits. The code is as follow.

```
@staticmethod
def correct_labels(labels):
    digits = [10, 10, 10, 10, 10]
    for idx, label in enumerate(labels):
        digits[idx] = int(label if label != 10 else 0)
    return digits
```

Figure 5: Generate labels pseudo codes

Therefore, after all data processing work, examples of the images look like these.

Figure 6: Data preprocessing examples

Our model has the same architecture as the best model proposed in the paper. It consists of eight convolutional hidden layers and two densely connected hidden layers. All connections are feedforward and go from one layer to the next. We use ReLu as the activation function in each hidden layer. The number of units at each spatial location in each layer is [48, 64, 128, 160] for the first four layers and 192 for all other locally connected layers. The fully connected layers contain 3,072 units each. Each convolutional layer includes max pooling and batch normalization. The max pooling window size is 2 * 2. The stride alternates between 2 and 1 at each layer, so that half of the layers don't reduce the spatial size of the representation. All convolutions use zero padding on the input to preserve representation size. All convolution kernels are of size 5 * 5. We train with dropout probability 20%. Cross entropy losses are calculated for length and each digit, and are summed up to get the total loss.

We use stochastic gradient descent (SGD) to train the model. Each mini-batch has 32 samples. To converge to a minimum, the learning rate of the SGD decreases at some appropriate rate during training. This is because the SGD gradient estimator introduces noise (32 random training samples) which does not become 0 even when we arrive at a minimum. In our model we use exponential decay for the learning rate, where we set decay rate and decay steps as 0.9 and 10,000 respectively.

Please see the following pseudo code for SGD.

Algorithm 1 Stochastic gradient descent update at training iteration k

Require: Learning rate η_k.
Require: Initial parameter $\boldsymbol{\theta}$.
while Stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\mathbf{x}^1, ..., \mathbf{x}^m\}$.
  Set $\hat{\mathbf{g}} = \mathbf{0}$.
  for $i = 1$ to m **do**
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \frac{1}{m} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^i; \boldsymbol{\theta}), \mathbf{y}^i)$
  end for
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_k \hat{\mathbf{g}}$
end while

if m=1 -> single example SGD

Figure 7: Stochastic Gradient Descent (SGD) Algorithm

We print the loss every 100 iterations, and use the latest model to evaluate the validation set every 1,000 iteration. Furthermore, we introduce regularization into our model by early stopping. Instead of running our optimization algorithm until we reach a minimum of validation error, we run it until the error on the validation set has not improved for some amount of time. If the accuracy on validation set has not improved in 20 continuous evaluations, the training stops.

Please see the following pseudo code for early stopping.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_0$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Figure 8: Early Stopping Algorithm

We would also like to clarify how we evaluate the accuracy on validation set. When determining the accuracy of the model, we compute the proportion of the input images for which the length of the sequence and every element of the sequence is predicted correctly. There is no partial credit for getting individual digits of the sequence correct. We put the prediction of the length and each digit together, and construct a single string. Please see the following code for string construction.

```
# get prediction
predictions = tf.concat([tf.reshape(length_pred, [-1,1]), digits_pred], axis=1)
predictions = tf.reduce_join(tf.as_string(predictions), axis=1)
```

Figure 9: Generate predictions pseudo codes

If the string is exactly the same as the string we construct from the true label, then we say that the model gives an accurate prediction on this sample.

5. Results

In this section, we present the major results we obtained from our model classifying the Street View House Numbers. 5.1 describes our project results in terms of model loss history, validation accuracy, and test accuracy. 5.2 compares our results with that the reference paper claims, and shows differences in training time, loss, and test accuracy. Finally, 5.3 provides some critical thinking about those differences observed, and proposes plausible reasons.

5.1. Project Results

As discussed in previous sections, our model follows the best architecture provided in the reference paper, which consists of eight convolutional hidden layers, and two densely connected hidden layers, along with identical parameters used such as convolutional size and number of neurons. Thus, we claim our model perfectly replicate both the methodology and procedures performed as discussed in the reference paper.

The following graphs plot the loss history (saved for every 100 batches/steps) and validation accuracy (saved for every 1000 batches/steps) from our model.

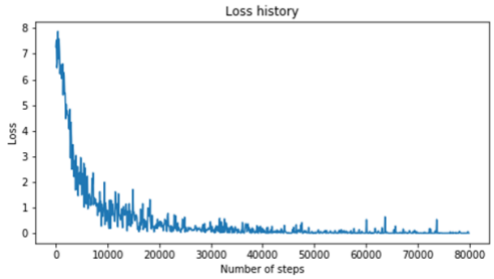

Figure 10: Loss history

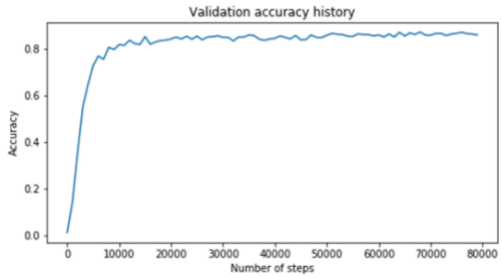

Figure 11: Accuracy history

The average loss value and validation accuracy we achieved, after 30K steps of batch training are completed, are 0.0570 and 81.997%. The best validation accuracy (corresponds to best model saved) and final validation accuracy (corresponds to last validation accuracy before finishing training) we finally obtained are 87.398% and 86.175% respectively. The loss values descend exponentially at first 10K steps, and then exhibit much slower descent in the following batch training.

Then we restore our best model from checkpoint file and apply the model to the test dataset which consists 10K images of street view house number. The following plot shows how test accuracy is improved as we apply different up-to-time best models from our checkpoint file.

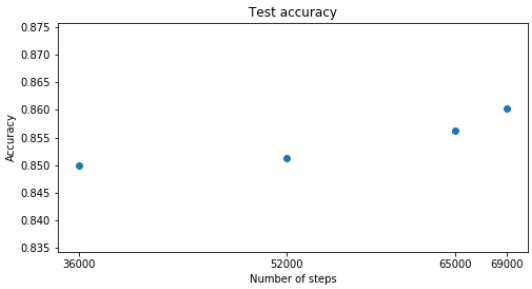

Figure 12: Test accuracy

The test accuracy we achieved with our best model is 86.02%. It's worth noticing that test accuracy increases as validation accuracy improves, and that test accuracy is indeed close to our validation accuracy (note that our validation dataset consists approximate 3K samples and test dataset consists 10K independent samples).

To better visualize the model performance, we randomly generate 9 samples from our test dataset, and then use the best model to predict digits. The following graph describes the real digits length and labels as well as the predicted digits length and labels. It's exciting to find that our trained model can accurately label all the 9 samples even for the hard-to-recognize image (e.g. the image in second row and third column, with a single digit 2), though the sample size is small.

Figure 13: Example of images in the test set

5.2. Comparison of Results

5.2.1 Sample size

The original paper uses the public Street View House Numbers (SVHN) dataset for its training, validation and test, which is about 200K street numbers total. While in our project, we obtain our house numbers data from http://ufldl.stanford.edu/housenumbers that lists both the train, test and extra datasets for downloading, and the size of the former two datasets are 33,402 and 13,068 respectively. To reduce the heavy computational power, we decide only to use the train and test datasets in our project. Thus, we have about 46K street numbers total, which is only a small fraction (23%) compared to the sample size used in the original paper.

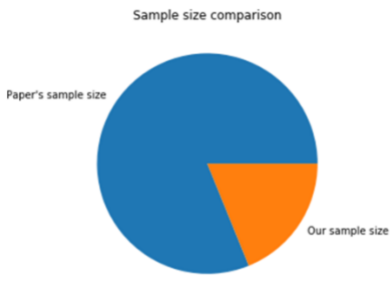

Figure 14: Sample size comparison

5.2.2 Training time

The original paper says they used 10 replicas in DistBelief and it took 6 days to train their model. In our case, we use a highest-configurated MacBook Pro to train our model and we set a stop criterion to finish the training process (stop training when at least 80K steps finished or no better validation accuracy achieved in 20K consecutive steps). Finally, it took us about 32 hours to train.

Figure 15: Training time comparison

5.2.3 Test accuracy

The original paper claims their best model obtained a sequence transcription accuracy 96.03% using their whole datasets. Due to the limit of sample size and computational power, we achieve a test accuracy of 86.02% for our best model.

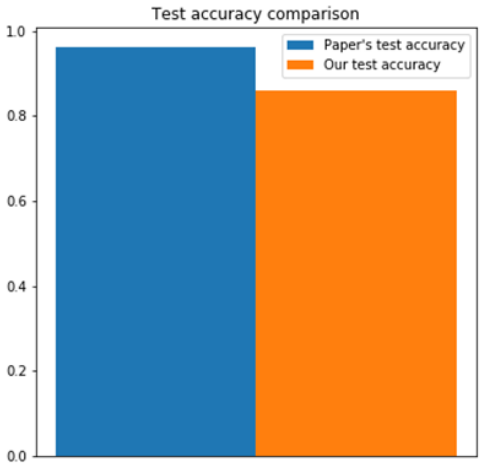

Figure 16: Test accuracy comparison

5.3. Discussion of Insights Gained

There are a couple of reasons that can cause differences in our results compared to those claimed in the paper.

Below are several most plausible and highly possible reasons.

5.3.1 The use of smaller samples and less training time

As we can see in last part, our samples only accounts for 23% of total samples the reference paper used. In contrast with other similar digits recognition tasks which try to crop individual digits from an image first and then recognize them, our task is far more challenging because we take the original images containing multiple digits and recognize them all simultaneously. Since one combination of digits in a single image can be quite different from another, we have much more possible patterns, or more specifically, features to recognize than that in single digit recognition. Hence this task requires tons of samples in order to generalize well. This should be the strongest factor that could have impact on our final test results.

Also, another important reason could be that we pre-defined a stop criterion to shorten our training time as we observe that loss error descends much slower as steps accumulate. But in facts, we can still improve the model performance by involving more training epochs, if computational power is no longer a concern.

5.3.2 Mis-labelled images in datasets

To verify if there are any labelling errors in our datasets, we randomly generate another 9 samples from our training datasets and print them out below.

Figure 17: Examples of mislabeled images

The label given by our dataset for image (0, 2) is 3, however we can visually observe that the "true" label should be 25 and our model can accurately predict the digits. This could be a serious problem since at this step the algorithm is not "correcting" the errors by updating the parameters, it instead messes with our parameters by adding some noises, which can harm our training process. Due to this fact, we expect that we can further improve our model performance once we can devise a way to remove those mis-labelled images from our datasets, or we include more data in our training process to force our model "ignore" those outliers.

6. Conclusion

In this project, we apply deep convolutional neural network to recognize multi-digits from street view images using the public Street View House Numbers (SVHN) dataset. We aim to reproduce the performance of our reference paper by using the same convolutional neural network structure and evaluating them on our datasets. However, we can't replicate the high accuracy as claimed in original paper, which is 96.03% in sequence transcription, due to computational limit. Our best model obtains 86.02% accuracy on the test dataset in the end, and we propose two most important reasons that could potentially drag our results: one is that we only use a fraction of available data with reduced training time, and the second is that the mis-labelled data in our datasets have introduced some noises.

This attempt to recognize multi-digits from images simultaneously is indeed a big step-ahead in computer vision, and CNN can be a powerful tool to achieve a reasonably high accuracy. However, this more challenging job requires a large scale of data to guarantee robust result, and depth is another crucial secret for success of the task. We expect in the future to explore more data, especially those hard-to-recognize ones, into our training process and include more layers in our model to generalize better.

7. References

Include all references - papers, code, links, books.
[1] Github link:
https://github.com/cu-zk-courses-org/e4040-2019fall-project-ZXXZ-rx2166-wz2466-yz3075
[2] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet, "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks", ICLR 2014.

8. Appendix

8.1 Individual student contributions in fractions - table

	rx2166	wz2466	yz3075

Last Name	Xie	Zhuo	Zhao
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Coding part: write “training”, “model”, and “meta” modules.	Coding part: Composed preproc.py module for data preprocessing; Composed jupyter notebook; Conducted training and testing.	Coding part: write “evaluator” module, revise “model” and “training” modules. Write README file.
What I did 2	Report part: Introduction, Summary of the Original Paper, and Methodology.	Report part: Results and Conclusion.	Report part: Implementation and Software Design.

8.2 Additional reference source:

[1] Reference code link:
https://github.com/potterhsu/SVHNClassifier
[2] Raw data link:
http://ufldl.stanford.edu/housenumbers/
[3] Additional reference: Columbia University ECBM E4040 Neural Networks and Deep Learning Fall 2019 Lecture Notes.