Multi-Digits Recognition based on CNN

E4040.2017Fall.final project report
Jiahao Li jl4930, Weikang Wang ww2461
Columbia University

Abstract

Our project aims to design architecture that inputs images with multi-digits and outputs the identities of them. This model can be used in situations, such as map-making using images including building numbers. The main challenge is to find a suitable model to represent the multi-digit and we solved this by using a mixture probability model. We get a 73.998% testing accuracy on the SVHN dataset.

Key word: *multi-digits recognition, deep learning, CNN, image processing*

1. Introduction

Recognizing multi-digit numbers in pictures captured at street level is an important component of modern-day map-making. But this task is really difficult. Difficulties arises due to the wide variability in the visual appearance of text in the wild on account of a large range of fonts, colors, styles, orientations, and character arrangements. The recognition problem is further complicated by environmental factors such as lighting, shadows, specularities, and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs. In realizing this model, we also face technical difficulties such as the choice of model architecture, the training method and the time saving method.

In our project, we design to use mixture probability distribution to model the multi-digits in the original picture. We construct a CNN to extract features from the original images and use these features as the inputs of several softmax classifiers to obtain probability distribution. We multiple the probabilities as the output of this architecture. The digits that maximize this output is the identities of the original image.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

We will provide a network that can recognize multi-digits in natural pictures after training on the SVHN datasets. We will also provide with the loss of the training process of the training datasets, the accuracy of the training datasets and the accuracy of the validation datasets. By the way, a net graph of our model is also provided.

As for the original paper, the authors want to train a probabilistic model of sequences given images. Let S represent the output sequence and X represent the input image. The goal is to learn a model of P (S|X) by maximizing log P (S|X) on the training datasets.

The paper defines S as a collection of N random variables $S_1 \dots S_N$ representing the elements of the sequence and an additional random variable L that representing the length of the sequence. They assume the identities of every number are independent from each other, so the probability of a sequence $s = s_1 \dots s_n$ is given by:

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

The model can also detect the sequence whose length is larger than N. By adding a number to L that represents this situation can solve this problem.

So, in details, L has 7 values, which is 0,1,2,3,4,5 and 'more than 5'; each S_i has 10 values, which is 0,1,2,3,4,5,6,7,8,9. The paper uses convolutional neural network to extract a feature H from every element of the training datasets (image) and construct six softmax classifiers for L and five S_i separately. These softmax classifiers all use H as its input and output a probability distribution to identify the length of the multi-digits in original image and every digits number. The graph explains this as below.

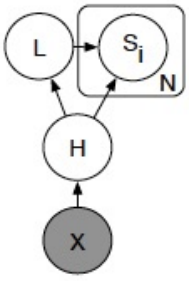

Figure 2-1-1

By using stochastic gradient method, the paper trains the network by maximizing log P (S|X).

At test time, the prediction is:

$$s = (l, s_1, \dots, s_l) = argmax_{L,S_1,\dots,S_N} \log P(S|X)$$

The flow chart of this process is:

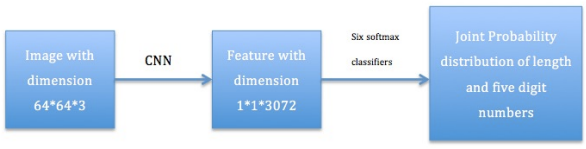

Figure2-1-2

2.2 Key Results of the Original Paper

The best model of the paper obtains a sequence transcription accuracy of 96.03%. And by using confidence thresholding, the paper obtains 95.64% coverage at 98% accuracy. The system also achieves a character-level accuracy of 97.84%.

3. Methodology

We first give the objectives of our project and the possible technical challenges in the first section, and then introduce the methods of how we solves the challenges and how we achieve the objectives. By the way, architecture of our model is also provided.

3.1. Objectives and Technical Challenges

The objectives of our project are to design an architecture that can recognize multi-digits in nature image without manure work.

The technical challenges concentrate on the following parts: architecture design challenge, training challenge and time challenge.

First, the architecture design challenge. According to the original paper, the authors state that the deeper the convolutional neural network is, the better the architecture works. But we cannot construct a infinitely deep CNN so we need try to find a suitable layer for our model. In the paper, the authors give a CNN architecture, which is 8 convolution layers and 3 MLP layers. But as for ourselves, we also need to combine with training datasets to choose our model structure.

The second challenge involves with the training datasets. The public training datasets is the SVHN datasets, which has 33402 images with labels. This is a really huge datasets and we need to design our training method carefully to avoid out-of-memory error.

The last one is the time challenge. Training a large CNN with large datasets will cost lots of time. We need to design a method that can use less time to get good accuracy in order to raise the efficiency.

3.2. Problem Formulation and Design

3.2.1. Problem formulation

First, we crop multi-digits from the original natural images by using the information in digit-struct.mat file. We resize every cropped image into a 64×64×3 image.

Then we design a convolutional neural network, which has 8 convolutional layers and 3 dense layers. The number of filters in the 8 convolutional layers is 48,64,128,160,192,192,192,192 and the max-pooling operation in each layer has stride 1 or 2 alternatively. The convolution filters in every layer have shape 5×5 and use zero padding to make the output and input has the same size of a convolution operation. And the max-pooling filters in every layer have shape 2×2 and also use zero padding to make the output and input has the same size. So after the convolution layers, the original 64×64×3 image becomes a 4×4×192 feature map. Then we flatten it and add two dense layers to the output of the flatten layer. Finally, we got a 3072-dimension feature from the architecture.

Finally, we use this 3072-dimension feature as the input of six softmax classifiers to get six probability distributions. These six softmax classifiers represent the length of the multi-digits in the image and five digits in the image. The softmax classifier that represents the length of the multi-digits has 7 entries, which means 0,1,2,3,4,5 and more than 5. The other five softmax classifiers all have 11 entries means 0,1,2,3,4,5,6,7,8,9 and none.

4. Implementation

We use CNN network to construct our model. There are four convolutional layers, 1 fully connected layer and 6 branch fully connected layers.

4.1. Deep Learning Network

4.1.1. Architectural block diagram(s)

In every two convolutional layers, max pooling will make the output picture become 1/4 size of the input picture. So, the 64*64*3 picture will be transformed into 4*4*192 with the knowledge that there are 192 filters in the last layer.

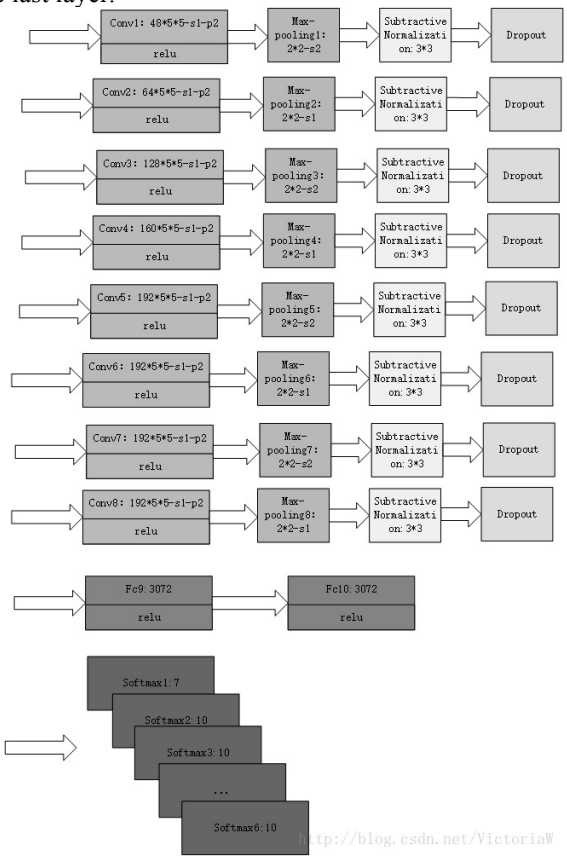

Figure 4-2-1

Then, we flatten the output of the convolutional neural network into shape (1*3720), and then use this as the input of the fully connected network in the graph below.

We use batch normalization in every CNN layer to prevent gradient vanish or gradient explosion and we use drop out to prevent overfitting. We use Relu as our activation function.

In the fully connected construction, we use the dense layers.

Figure 4-2-1 is the structure of the architecture.

4.2. Software Design

4.2.1 Top level flow chart

Figure 4-2-2

Firstly, we need to abstract 5000 pictures from the compressed file train.tar.gz we downloaded as our training set. And we abstract 1000 pictures from the compressed file test.tar.gz as our testing set.

Then we cut down the unrelevant part of all 6000 pictures according to the digi_struct.mat file. After that, we need to reshape them all to shape 64 * 64 *3. These data will be the training data and testing data of our network.

We construct our network Using CNN and fully connected layer using tensorflow, we derive loss function from the logits derived by the network and our labels.

After that, we put training batches into the network to train it. Finally the model and the training and testing accuracy will be saved in files.

4.2.2 step-by-step implementation

1) Data abstraction

We write a function called getTrainData(number), it could help you get a given number of pictures from the folder "train". Similarly, a function getTestData(number) could get you a given number of pictures from the folder "test.

Then, we write a function called generalize5Label(y, y_length). This function is a little bit complicated so I give you an example.

Notice that there are 2 inputs y and y_length. If the picture is 19, 23, 250, then y would be 1,9,2,3,2,5,10 (Notice that 10 represent 0) and y_length would be 2,2,3. This function will generalize a matrix from y and y_length and the result would be:

1	9	0	0	0
2	3	0	0	0
2	5	10	0	0

Table 4-2-1

In this table, you could find that all the number was given 5 digits, if the length of a given number is smaller than 5, then the exceeded digits will be given 0.

2) Data process

As you could see in the figure below, the shape of the picture is irregular.

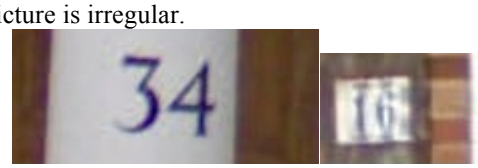

Figure 4-2-3

So we have to cut off those that are irrelevant. Fortunately, there is the information about the accurate position of the figure, so we don't have to detect the data by ourselves. However, the square sizes are different.

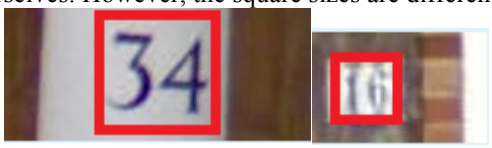

Figure 4-2-4

So, we have to use function region = im.crop(box).resize((64,64)) to resize them. And then, they would have the same shape.

3) network construction

It is not very difficult for us to build the network using tensorflow. For the reason that all the convolutional layer are similar with each other and all the fully connected layers are similar with each other too. We could just give examples for each type of layers.

```
with tf.variable_scope('hidden1'):
    conv = tf.layers.conv2d(tf_X, filters=48, kernel_size=[5, 5], padding='same')
    norm = tf.layers.batch_normalization(conv)
    activation = tf.nn.relu(norm)
    pool = tf.layers.max_pooling2d(activation, pool_size=[2, 2], strides=2, padding='same')
    dropout = tf.layers.dropout(pool, rate=drop_rate)
    hidden1 = dropout
```

Table 4-2-2

```
with tf.variable_scope('hidden5'):
    dense = tf.layers.dense(flatten, units=2560, activation=tf.nn.relu)
    hidden5 = dense
```

Table 4-2-3

4) train

In order to prevent memory insufficient. We have to read a batch of data each iteration and then put it into the model.

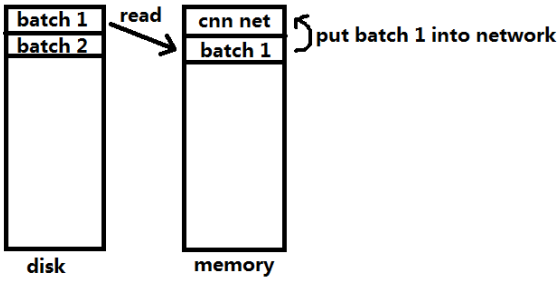

Figure 4-2-5

Finally, we use a txt file to write down all the training accuracy and testing accuracy.

4.2.3 Pseudo code for each section of the implementation

1) data abstraction

```
all_train_picture = []
all_test_picture = []

For i in range(training number):
      picture(i) = read_picture
      all_picture.append(picture(i))

For i in range(testing number):
      picture(i) = read_picture
      all_picture.append(picture(i))

read all label from mat file #for example 19,23,250 with label 1,9,2,3,2,5,10 and length label 2,2,3

change label structure # for example [1,9,0,0,0],[2,3,0,0,0],[2,5,10,0,0]
```

Table 4-2-4

2) data processing

```
train_data_after_process = []
test_data_after_process = []

for i in range(training number):
      read box from struct mat file
      cut picture into squares(all_train_picture(i), box)
      data_after_process = resize(square(i))
      train_data_after_process.append(data_after_process)

for i in range(testing number):
      read box from struct mat file
      cut picture into squares(all_test_picture(i), box)
      data_after_process = resize(square(i))
      test_data_after_process.append(data_after_process)
```

Table 4-2-5

3) build network

```
generate 4 convolutional layer

def one convolutional layer:
      conv = convolutional layer(input)
      norm = batch_normalization(conv)
      activation = relu(norm)
      pool = max_pool(activation)
      dropout = dropout(pool)
      output = dropout

def fully connected layer:
      output = dense(input)

      loss_digit_1 = mean(sparse softmax cross entropy(labels, logits)
            ...
      loss_digit_5 = mean(sparse softmax cross entropy(labels, logits)

total_loss = loss_length + loss_digit_1 + loss_digit_2 + loss_digit_3 + loss_digit_4 + loss_digit_5

accuracy = accuracy + correct_length && correct_digit_1 && correct_digit_2 && correct_digit_3 && correct_digit_4 && correct_digit_5
```

Table 4-2-6

4) train

```
batch_size = a given number
for iteration in range(a given number):
            for i in range(total_number/batch_size):
                  x_batch = read_batch_data
                  y_batch = read_batch_data from label after process_batch
                  run.train(x_batch, y_batch)
                  print(loss, training accuracy, test accuracy)

save model
save accuracy and loss
```

Table 4-2-7

5. Results

5.1. Project Results

The loss of the training process, the training accuracy and the validation accuracy is shown below:

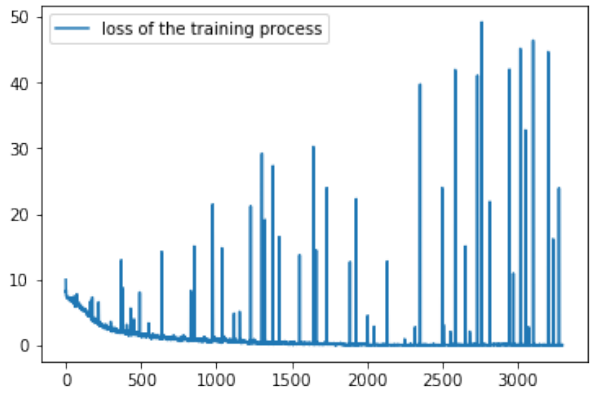

Figure 5-1-1 the loss of the training process

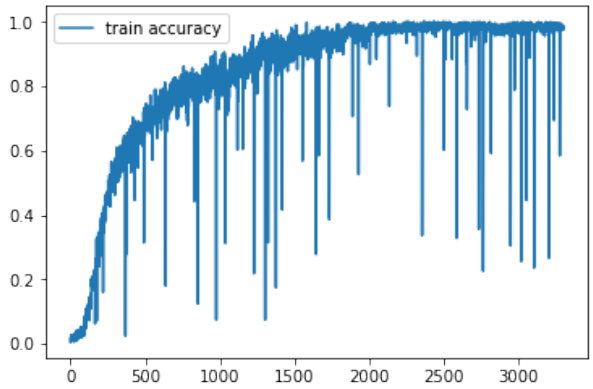

Figure 5-1-2 training accuracy

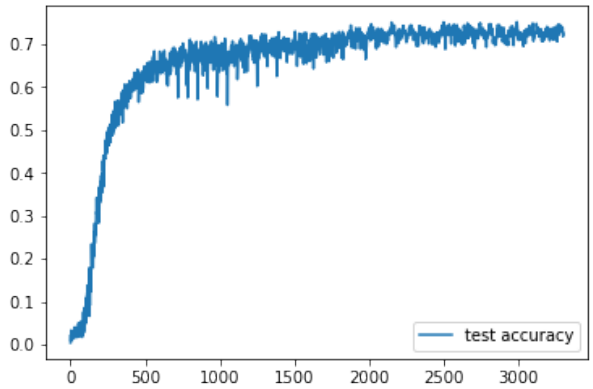

Figure 5-1-3 testing accuracy

As we can see from the plots, that the testing and training accuracy is growing faster in the first 500 iterations. The training accuracy is then slowly approaching 100% and the testing accuracy is slowly approaching 73%.
At the last 2000 iterations, we can see that, as training accuracy grows, testing accuracy grows very little. This means the model is already over-fitting.
As shown in the plots of Figure 5-1-1 and Figure 5-1-2, there are lots of periodic spikes in the plots. This phenomenon is probably due to the same batch of training data with wrong training label.

5.2. Comparison of Results

Our result is shown in the section of 5.1. In the original paper, the authors just give the final test accuracy and the training time. We show the comparison in the following table.

	Training Time	Test Accuracy
Original Paper	7 days	96.03%
Our project	13 hours	73.998%

Tabel 5-2-1

We can see that the training time of the original paper is much longer than ours and the testing accuracy is also bigger than ours. This could be because they use much more data to train the model and use different techniques to tuning the parameters of the model. In our project, we just use the training datasets to train the model and have not used the extra datasets.

5.3. Discussion of Insights Gained

we understand the theory about how to recognize a multi-digits based on CNN. And we use python and tensorflow to implement this model successfully. We find that in this problem, in general, the more complex the model is, the better the training result will be.

we also understand when the data become larger, it is really important to just read a little batch of data from disk to memory each time, because if you want to read all data from the disk in one time, the memory will overflow and the programme will shut down.

6. Conclusion

We understand the theory of multi-digit recognition based on deep learning and we complete our task by using convolutional neural network, and our accuracy is 73.998%.

We still know when the data set is very large, it is a clever way to use small data set several times to train the model and then save the model and train another subset of data. This could help us prevent memory become exhausted.

7. References

Include all references - papers, code, links, books.
[1]https://bitbucket.org/ecbm4040/2017_assignment2_ww2461/src
Note: My teammate's bitbucket link is in his own report
[2] H. Li, "Author Guidelines for CMPE 146/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

8. Appendix

8.1 Individual student contributions - table

	CUID1 jl4930	CUID2 ww2461
Last Name	Jiahao Li	Weikang Wang
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Dataset processing	Paper analyzing
What I did 2	Model Construction	Help constructing model
What I did 3	Training and Testing model	Training and Testing model
What I did 4	Report	Report