COMP6248 Reproduction challenge/ Street Image Recognition

Zhihan Wang & Shaunak Sen & Alejandro de Jess Alonso canales
Electronics and Computer Science
zw3u18@soton.ac.uk & ss8n18@soton.ac.uk & adac1n17@soton.ac.uk

Abstract

The main work of this assignment is to reproduce (2), mainly sector 5.1. The task in (2) is to recognize digits in the street images using convolutional neural network. Hence this paper could be split into two parts. One is the knowledge what we learned from (2) and another is the experiment about reproduction.

1 Introduction

Optical character recognition has been well studied on constrained domains, such as document processing, but is still challenging in unconstrained domains, such as natural photographs. In the paper (2), an equally hard sub-problem, arbitrary multi-character text recognition in photos captured at street level, has been discussed.

The paper employed DistBelief, a software framework that can utilize computing clusters with thousands of machines to train large models(1), in which large-scale deep neural networks are implemented on publicly available Street View House Numbers(SVHN) dataset. This model finally achieved over 96% accuracy in recognizing street numbers, and 97.84% accuracy on per-digit recognition tasks.

1.1 Architecture

There are three steps in traditional approaches to recognize multi-digit numbers from photos. They are localization, segmentation, and recognition respectively. The paper (2) proposed a unified model to integrate these three steps via the use of a deep convolutional neural network that operates directly on the image pixels and achieved an end-to-end prediction.

1.1.1 Probabilistic Approach

The task of street number recognition is that given an image, the numbers in the image should be identified. The basic method used here is to train a probabilistic model of a predicted sequence output given an image. Let **S** represent the output sequence and X represent the input image. The goal is to learn a model of $P(\mathbf{S}|X)$ by maximizing $\log P(\mathbf{S}|X)$ on the training dataset. The probability of a specific sequences **s**=$s_1, s_2, ..., s_n$ is given as below, in which n is the number of digits in the image.

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

At prediction time,

$$\mathbf{s'} = (l, s_1, s_2, ..., s_l) = \underset{L,S_1,S_2,...,S_L}{\arg\max} \log P(S|X)$$

1.1.2 CNN structure

The best model trained on the SVHM dataset in (2) is with 11 hidden layers, consisting of 8 convolutional layers, 1 locally connected hidden layers and 2 densely connected hidden layers. All connections are feedforward and there are no skipped layers. Each convolutional layer contains a convolutional part, a generalization unit and max pooling. Two methods to achieve generalization are used in the original paper, maxout units(3) and ReLU. All the max pooling parts are with 2×2

windows. Then the results are passed to the subtractive normalization which helps to decrease the variability using the window in size 3×3. The stride at each layer alternates between 1 and 2; therefore, zero padding is used to preserve representation size. The size of all the kernels is in 5×5. As significant overfitting can be seen, dropout units are applied to all hidden layers.

1.2 Performance

(2) shows that the performance of model increases with the depth of the convolutional network. Two experiments were used to prove this conclusion. The first one confirmed that depth is necessary for good performance and the second one with a accuracy graph demonstrated that smaller models even with more parameters cannot reach the same level of the performance as deeper models either.

1.3 Comparison with previous work

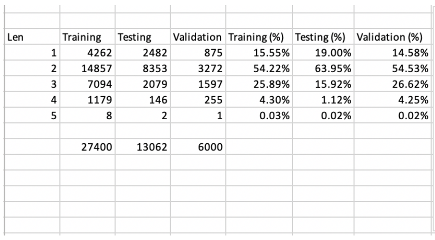
Images recognition networks trained in the previously published papers generally have 2 to 4 convolutional layers followed by 1 or 2 densely connected layers and classification layers. But the model in (2) used more convolutional layers, 8, as referred above. This is because earlier layers are used to solve localization and segmentation firstly, and then the results are processed to later layers to recognize. Therefore, the model can achieve an end-to-end prediction.

2 Reproduction

In this part, we are going to reproduce section 5.1(2), because this is the section which uses public dataset, and show the performance of the model trained by us on the public Street View House Numbers dataset. The detailed information is described as below.

2.1 SVHN dataset

The SVHN dataset is obtained from (4). It contains 200k street numbers with the location information of individual digit in each image which is stored in a .mat file. We used the Python to read the .mat file as there is an available API to the MATLAB in Python. The information of the labels is stored into numpy arrays, we found out that the 1st element tell us how many numbers are on that image (a range between 1-5), and the next n elements are the labels of the numbers within the image (a range between 1-10). The number 10 was used as padding for the elements not used. One inconsistency we discovered about the dataset was that the digit 0 was represented as 10 in the labels. We corrected this inconsistency. Before training models, we use a data pre-processing part to reconstruct the data.



Len	Training	Testing	Validation	Training (%)	Testing (%)	Validation (%)
1	4262	2482	875	15.55%	19.00%	14.58%
2	14857	8353	3272	54.22%	63.95%	54.53%
3	7094	2079	1597	25.89%	15.92%	26.62%
4	1179	146	255	4.30%	1.12%	4.25%
5	8	2	1	0.03%	0.02%	0.02%
	27400	13062	6000			

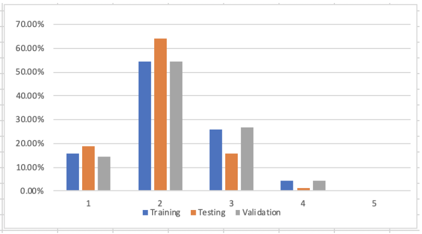


Figure 1: Data imbalance

Firstly, we find the rectangular bounding box that only contains an individual character. Then we expand the bounding box in both x and y direction by 30% to crop the image. Next, re size the output to 64×64 pixels. Then we analyzed the distribution of the amount of numbers on each image. The results are shown in fig1. We found out that there was a big imbalance on the classes. 54% of the training data are images with 2 numbers in it, followed by 25.89% with 3 numbers and 15.55 with one number, only 4.33% are images with 4 and 5 numbers on them. We trained the model based on this original dataset and then found that the model did not performed very well. Then we checked the accuracy in different class. We found the accuracy of the class, in which images were all with

2 numbers was the highest. Hence, we inferred that this circumstance was resulted from the data imbalance.

To solve imbalance problem, we decided to use data augmentation to create a more uniform data set for training and testing. From the 64×64 original image we will use a random 54×54 crop to create more data for the model. We will use this as many times as each class needs to be more balanced. For example, for the images with just 1 number we will create 3 new images using the shifted window, so from the 4262 images we will get around 12600 images for our model.

2.2 Architecture

The model used to train is a Convolutional Neural Network with 11 hidden layers. The first 8 hidden layers are convolutional ones followed by 2 densely connected hidden layers and 1 locally connected hidden one. We took reference from (5). Each convolutional layer is with a kernel filter in 5×5, a 2×2 window to achieve max pooling, a batch normalization unit and a generalization part. We used ReLU to achieve generalization. The architecture is shown in Fig2.

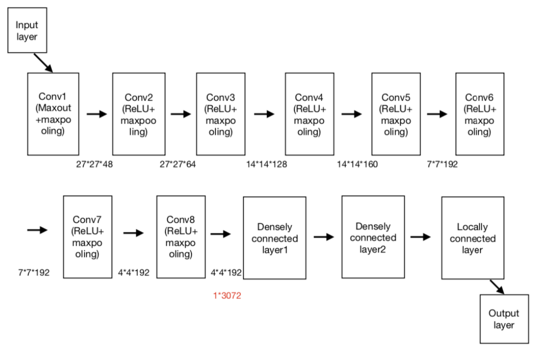


Figure 2: Model architecture. The output of the first convolutional layer is $32 \times 27 \times 27 \times 48$ where 32 is batch size, 48 is the number of units at each spatial location, and 27 is the output dimension. The output dimension of convolutional layer is calculated by $O = \frac{I-K+2P}{S} + 1$ where I is the input dimension, K is kernel size, P is padding and S is for strike. In the computation of the output of the first layer, $I = 54, K = 5, P = 2, S = 2$. And after max pooling, the dimension is calculated by $O = \frac{I-P}{S} + 1$ using the parameter value from last step. They are $I = 27, P = 1, S = 1$ respectively, so the final output dimension is 27. The red number part shows after the convolution and max pooling, the output matrix is required to be flattened to be proceeded into the densely connected layers.

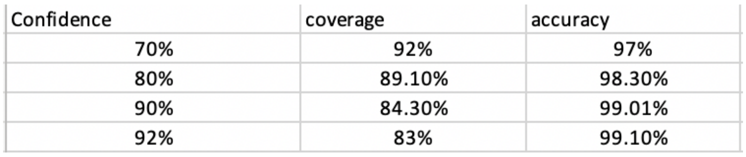
The stride used at each convolutional layer alternates between 2 and 1. Therefore, half the convolutional layers do not reduce the representation size. In this case, we use zero padding in the inputs of all the convolutional layers to keep spatial size of representation. To avoid the overfitting case, dropout units are used in the training part. When it comes to computing the loss and model training, we used the SGD to optimize the parameters.

Another interesting point is that we use a parameter named $coverage$ in the model. The $coverage$ is defined as the proportion of inputs which is not discarded(2). And we use ”confidence threshold” to decide what to discard. For example, we set up a ”confidence threshold” as 0.8 at first. Then any output matrix obtained from softmax unit would be discarded if there is a maximum value of the component in each row lower than 0.8.

Obviously, the value of $coverage$ will decrease if we set up a higher ”confidence threshold”. Therefore, at the beginning, we will set up a higher ”confidence threshold” to help the model achieve a certain accuracy level. And when it performs well, we will increase the value of $coverage$ by reducing the ”confidence threshold”. Hence the model would be much robuster.

2.3 Experiment results

The programming code has been uploaded in [1]. We took reference of the preprocessing steps from here from (6). We set up the rate of dropout as 0.2 and learning rate in the SGD as 0.03. The value of coverage was 100% here as we set up the confidence threshold as 0 initially. Before data augmentation, the accuracy of the prediction on the test set was 93.35%. The accuracy of per-digit was over 95%. And iterating after 20 epochs, the final overall accuracy reached 94.5%. Then implemented ”confidence threshold” to the data set. The detailed results are show in fig3. The fig3 shows we obtain an almost same results with the original paper.



Confidence	coverage	accuracy
70%	92%	97%
80%	89.10%	98.30%
90%	84.30%	99.01%
92%	83%	99.10%

Figure 3: Confidence-coverage-accuracy. At first, we set up the threshold as 0.92 and obtain a coverage value, 83% and 99.1% accuracy. Then slightly change the threshold to 0.90, 0.80, and 0.70.

3 Evaluation

In our reproducibility challenge, we found that the model on per-digit recognition performed very well and the value of accuracy is the highest. The overall accuracy is lower than human level. In addition, after we trained the model, we found that the performance of model would increase if the neural network was much deeper. These conclusion were the same as the results in paper(2). However, the final accuracy achieved by our model was always slightly lower than the original paper. We think about a few reasons why this circumstance happened. Firstly, the detailed information about the parameters setting is not very clear in (2), for example, in which proportion to split train and test data set, or the number of epochs to iterate. Besides, (2) worked with multiple CNN architectures, while we only implemented one CNN architecture in the model.

For future works, we can use multiple GPUs to achieve this work and add some little tricks to our model like using maxout(3) instead of ReLU. We can augment the data set to solve the imbalance problem as well, using the method we referred in section 2.1.

4 Conclusion

In this reproducibility challenge, we trained a CNN model with 11 hidden layers to recognize the digits from the street view imagery. We have obtained some results similar to the original paper which means we made a good reproduction.

References

[1] Dean, Jeffrey, et al. ”Large scale distributed deep networks.” Advances in neural information processing systems. 2012.

[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] Goodfellow, Ian J., et al. ”Maxout networks.” arXiv preprint arXiv:1302.4389 (2013).

[4] http://ufldl.stanford.edu/housenumbers/

[5] https://github.com/potterhsu/SVHNClassifier

[6] https://ryannng.github.io/2016/12/20/Street-View-House-Numbers-Recognition-Using-ConvNets/

[1] https://github.com/COMP6248-Reproducability-Challenge/Multi-number-recognition-street-view