Image Captcha Recognition

Xiaoguang Wang	Qian Wang	Hao Zhang
A53246911	A53235276	A53234732
xiw503@ ucsd.edu	qiw018@ ucsd.edu	haz351@ ucsd.edu

Abstract

Image captcha has long been used as a common protective tool for some websites to tell computer and human apart. Here, we employed the pre-trained AlexNet and combined with joint training method to build a model that can automatically recognize the numbers and characters on the captcha image and received a recognition accuracy of 96% on the most complex occasion with numbers, upper and lower characters. Also, we tried to borrow some technology from traditional computer vision like median blur to denoise the captcha images and improved the performance of our recognition pipeline.

1 Introduction

Captcha here is an acronym for "**C**ompletely **A**utomatic **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part". It is useful since computers can hardly read the contents like numbers and characters from images like a human. However, thanks to the fast development in the research of computer vision and especially artificial neural network, machines now can better understand the specific semantic information from images. For example, LeNet [1] has proved great success in handwritten digit recognition.

Figure 1: Some examples of captcha images

Inspired from the above ideas and forerunners, we proposed a novel system which can automatically learn useful features from the captcha images and make predictions of the numbers and characters on each digit separately. Figure 2 provides the broad framework for our system.

The main **challenge** here is that we cannot regard this problem as a traditional image classification problem since we don't have specific labels or categories. Say a 4-digit caption set with simplest char set with only number from 0 to 9, if we want to "classify" this dataset, we have to specify 10^4 categories. This makes no sense obviously and can hardly adapt to larger char set with characters.

In this project, we proposed a novel captcha recognition network shown below and implemented the system on TensorFlow. The main contributions of our work are outlined as follows:

- Modified the existing AlexNet with multiple output softmax layers to make predictions for each digit of captcha code separately and used multi-task learning method to tune the model.
- Combined with some image smoothing methods from traditional computer vision to pre-process the input images to the network and improved performance.

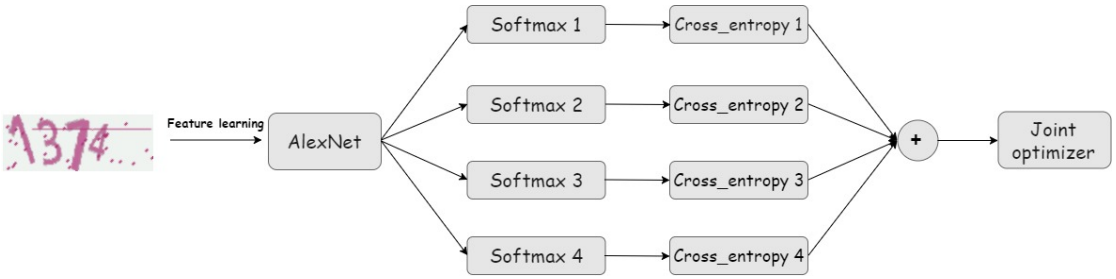

Figure 2: Network structure for the recognition system (4-digits captcha)

2 Approach

In this section, we will present more specific description of the designed network and some implementation details. Specifically, although the method proposed in this project can be easily adapted to any length captcha recognition, we implemented a system for 4-digit here for the convenience to verify the feasibility of our ideas.

2.1 Model architecture overview

Figure 2 contains the architecture of the model we built in this project. It is a transformation from traditional AlexNet.

Feature learning: We firstly used a pre-trained AlexNet for image feature learning since it has been well trained on a large image dataset – ImageNet and proved to represent images features well with lower dimension feature vectors. We leave the net parameters untouched during the training process. Also, we explored some more modern networks like GoogLeNet, but unluckily, the GPU server seems out of resources to run this large network well. So, we chose AlexNet at last and believe this network can extract necessary features well from the simple captcha images.

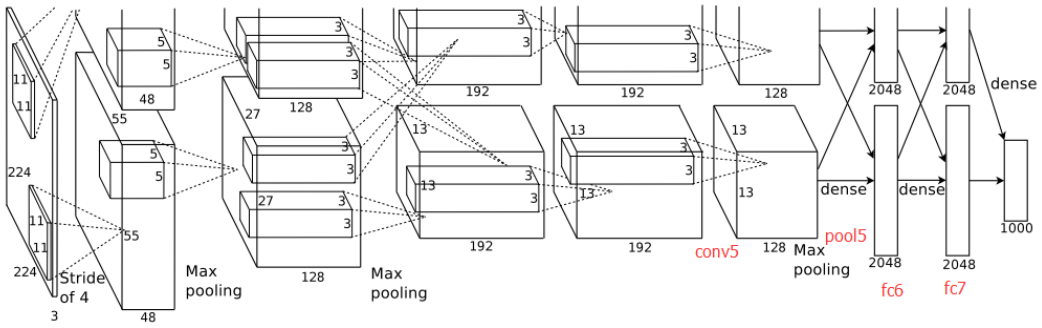

Figure 3: Original AlexNet structure for comparison

Captcha Prediction: we used one *softmax* layer to predict each digit separately (here we set

the captcha length to be 4, so we implemented 4 *softmax* layers). To train this model, we use a method called multi-task learning (we will give specific explanation in the next part).

2.2 Training (Multi-task learning)

Normally in a multiple categories classification problem, we use *softmax* layer to map image features from their feature space into a probability distribution. Then calculate and backpropagate the cross-entropy loss value to tune net parameters. However, here we have one output and get one cross-entropy loss value from each output *softmax* layer.

To handle this problem, we used a method from multi-tasking learning called **joint training**. That is in the training process, instead of backpropagating these loss values four times separately, we backpropagate the sum or the mean value of these loss values and assume that we can get the best prediction if we can minimize this "complex loss value" through some methods like gradient descent.

2.3 Image smoothing

Smoothing is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Here, we mainly tried Gaussian blur and median blur.

Gaussian blur: Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function. Since the Fourier transform of a Gaussian is another Gaussian, applying a Gaussian blur has the effect of reducing the image's high-frequency components; a Gaussian blur is thus a filter. The equation of a Gaussian function in one dimension is

$$G(r) = \frac{1}{\sqrt{2\pi\sigma^2}^N} e^{-r^2/(2\sigma^2)}$$

Median Blur: The median filter is a nonlinear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing. Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise, also having applications in signal processing.

Below is an example of applying the above two kinds of smoothing method.

Figure 4: effect of image smoothing (original, Gaussian blur and median blur from left to right)

We can see, median blur has a better performance on removing noise while preserve some useful details on the images, so we chose it for further exploration.

3 Experiment

We built a 4-digit captcha image recognition model to verify the feasibility of our design of the system. Also, the model is implemented with TensorFlow and ran on UCSD DSMLP.

3.1 Dataset

We searched the opening datasets on the Internet, but we cannot find a suitable dataset (one at least should have sufficient captcha images and corresponding labels to train our network). But luckily enough, there is a Python library called *captcha* (https://pypi.org/project/captcha/), we can use the library to build our own dataset (including training set and test set). Below is a demo of the generated dataset.

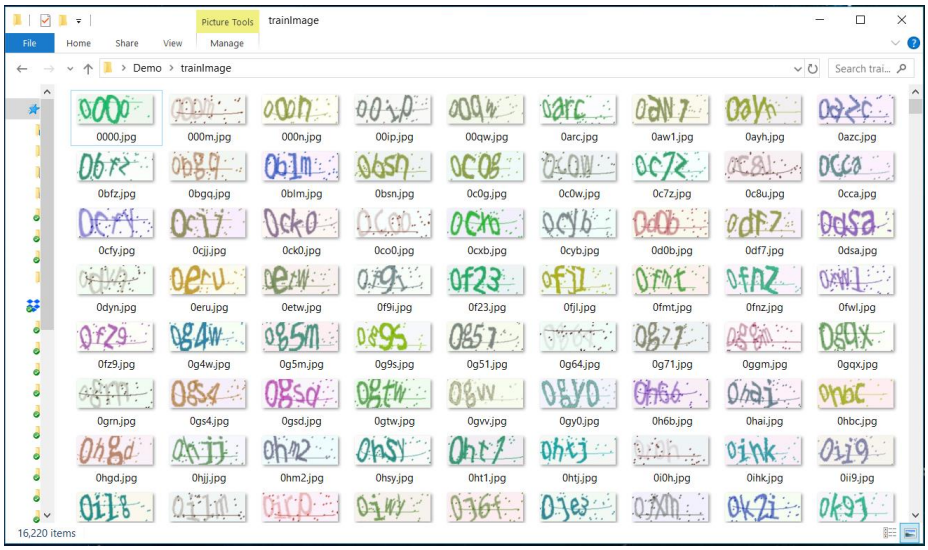

Figure 5: Running demo for Python captcha

We can see for each captcha image, we set their file name to be the content on the image, which serve as the supervised information(labels) during the training process. Also, since the dataset is generated and can be controlled strictly on the image format and dataset size, which is very convenient to verify our ideas and model. In this project, we generated 10000 images for each model and used 500 for testing and the rest for training.

3.2 Evaluation Protocol

For image captcha recognition, what we concern about mostly is the prediction accuracy. So here, we use prediction accuracy on each of the 4 digits as the only metrics to evaluate out models. Besides this, we plot the accuracy and loss value on the whole training set to help us monitor the training process.

3.3 Hyperparameters

More importantly, let's consider another important problem for our model – robustness from the following two aspects:

- **Complexity of the code**: here, we explored different levels of complexity of the generated captcha code from numbers only (char set size: 10) to numbers + lower case characters (char set size: 36) and lastly the most complex case with numbers + lower and upper-case characters (char set size: 62).

- **Noise**: As an improvement to our system, here instead of inputting the generated captcha images directly into the model, we borrowed some technology from traditional computer vision and pre-process the images with some noise cancellation operations like Gaussian

blue and median blur. We compared the effect of image smoothing on our model.

3.4 Results

In this part, we will present results we get on our model with different char set complexity: numbers only, numbers + lower case characters, number + lower + upper case characters and a comparison between the results of with or without smoothing operation. Typically, recognition accuracy on each digit can reach a level higher than 90%.

3.4.1 numbers only

We only used digits from 0 to 9 to compose a 4-digits captcha image. Below are the accuracies on test set and mean loss value on training set we get during the training process.

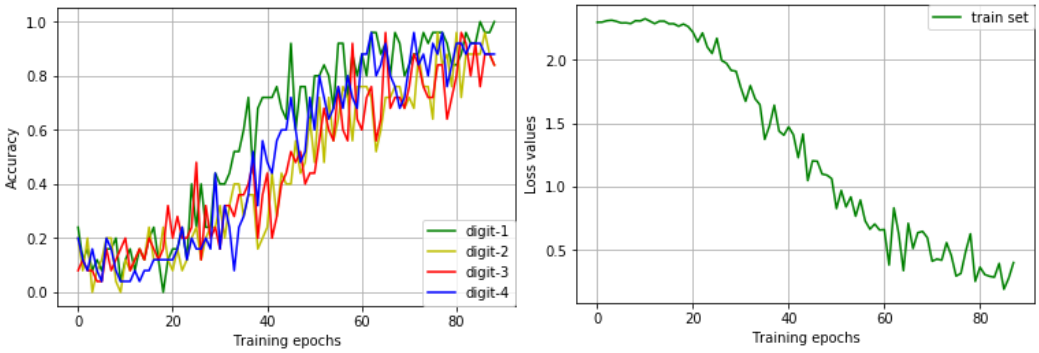

Figure 6: Accuracies and loss value during the training process (numbers only)
(sampling rate： 20 iterations/sample)

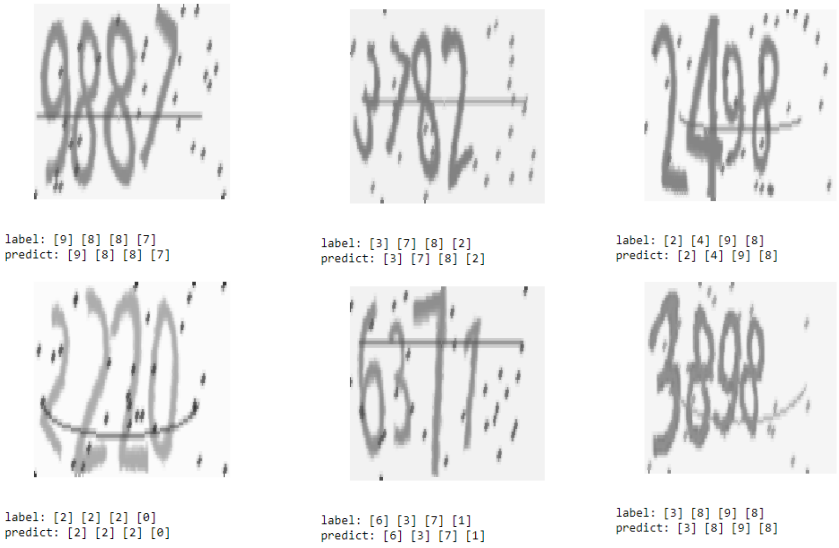

Figure 7: Test demos for image captcha recognition (numbers only)

3.4.2 numbers + lower case characters

Here, we used digits from 0 to 9 and lower-case characters from ‘a’ to ‘z’ to compose a 4-digits captcha image. Below are the accuracies on test set and mean loss value on training set we get during the training process.

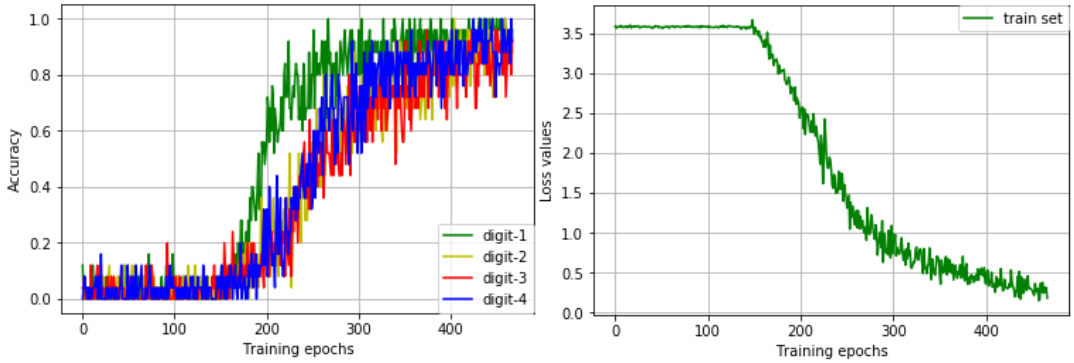

Figure 8: Accuracies and loss value during the training process (numbers + lower case)
(sampling rate: 20 iterations/sample)

Figure 9: Test demos for image captcha recognition (numbers + lower case)

3.4.3 numbers + lower and upper-case characters

Here, we used digits from 0 to 9 and lower-case characters from ‘a’ to ‘z’ and upper-case characters from ‘A’ to ‘Z’ to compose a 4-digits captcha image. Below are the accuracies on test set and mean loss value on training set we get during the training process.

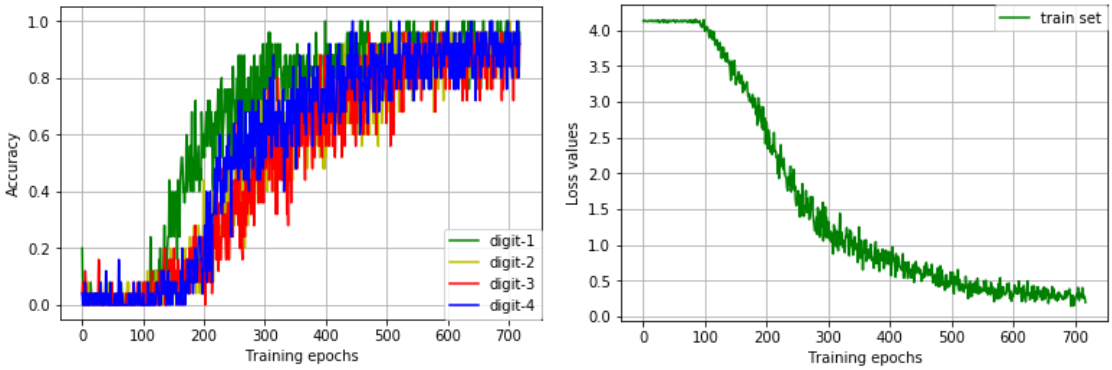

Figure 10: Accuracies and loss value during the training process

(numbers + lower, upper case)

Figure 11: Test demos for image captcha recognition (numbers + lower, upper case)

3.3.4 Median blur

In this part, we before feeding captcha images into AlexNet, we preprocessed them with median blur to remove noise like the lines and spots on the images. Below are the accuracies on test set and mean loss value on training set we get during the training process. (We compare with part 3.3.2, captcha built from numbers and lower case character)

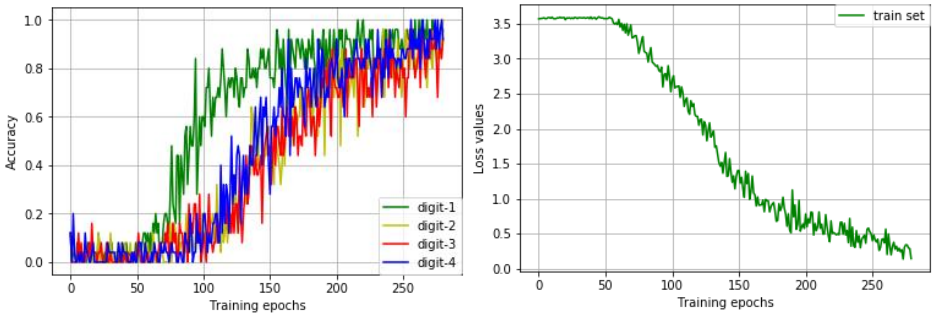

Figure 12: Accuracies and loss value during the training process (numbers + lower case)

(with image smoothing, sampling rate： 20 iterations/sample)

As we can see, after applying image smoothing, accuracy of the model doesn't get improved much as we expected since we have got a high recognition accuracy in the previous part, but our model can converge fasted compared with experiments in the previous part without denoising (same learning rate). Here, it took 5620 iterations for convergence, but in part 3.3.2 without smoothing, it took nearly 9000 iterations for convergence.

4 Conclusions and Discussions

Trough the whole project, first we proposed a novel idea of using convolutional neural network to recognize captcha images. Here, different from traditional image classification problems (since the simplest 4-digit number captcha has 10^4 kinds of labels), we proposed a joint training method to train the model and predict the character on each digit with a SoftMax layer separately. Below are some interesting conclusions we get from the results from the previous part.

● **Accuracy**: Our system can reach a high accuracy of more than 95% on each digit on test set generally.

● **Training process**: During the training process, since we randomly initialize the parameters in the SoftMax layer, we suffered a problem called "cold start", that is, during the initial hundreds of iterations, accuracies and losses would stay flattened. And then accuracy on the first digit would began to raise (the green line above) ahead of other digits, which is because of the joint training method we use.

● **Drawbacks**: From the running demos above we can see, although our model has reached a high accuracy on test set, but there are still some mistakes like 0-o, 9-g, 7-T etc. This may because the characters are not written in a clear and neat format, and even for a human, it is still a little difficult to distinguish these pairs sometimes. Also, denoise operation help little with this situation.

Reference

[1] Yann LeCun. (Proc. IEEE 1998) Gradient-Based Learning Applied to Document Recognition.

[2] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. (2012) ImageNet Classification with Deep Convolutional Neural Networks.