The Application of Convolutional Neural Network in Security Code Recognition

Jingtian Gu

1College of Engineering, University of Washington-Seattle, U District, Seattle, WA, United State

Abstract. Security Code or CAPTCHA, which represents Completely Automated Public Turing test to tell Computers and Humans Apart, is used to determine whether the user is human or not. It is widely applied in website management to prevent people from maliciously registering and spreading spam. With the development of machine learning algorithms and artificial intelligence technologies, it is possible to recognize the security codes with machine so that people can have access to the website without registration limits. There are many traditional machine learning algorithms such as Support Vector Machine (SVM) and Random forest are used to recognize security codes, but they have various disadvantages like low efficiency and low learning ability, inability to extract features automatically and difficulty in handling 2-dimension pictures directly. In machine learning area, the convolution neural network (CNN) is famous for its strong learning ability and automatic feature extraction, as well as high learning ability and efficiency, which makes it suitable for 2-dimension image data. Thus, we construct a convolutional neural network for security code recognition. The proposed CNN model is made up with 3 convolutional layers, a flatten layer and a full-connected layer. With the proposed model, we achieve an accuracy of 80.5% in the validation set.

1. Introduction

Security codes have been widely applied in different websites like shopping websites, social media and even search engines. They take advantage of security code for different purpose. For instance, social media use it to filter excessively repetitive registration of advertisement and paid poster accounts to ensure the clean environment. Google gives user security codes when they have unusual behaviors, and ticket websites use these codes to prevent ticket resale. However, some security code recognition models have been built to avoid these codes and make it easier to login or conduct web crawler. Several softwares like automatic login assistant have been published for these purposes. Some even comes into commercial application, for example, 360 has deployed this technology in its software products to handle the security code for its users, which has received a lot of positive comment especially in ticket panic buying.

Most of these security code recognition models are based on template matching [1] algorithm or some traditional machine learning algorithms like SVM [2]. Template matching algorithm matches parts of a processing image with a template image at different levels from pixel level to higher level to judge the texts in the processing image. Two types of matching, feature-based matching and template-based matching, are usually chosen to implement the algorithm. However, the results of matching depend on the template images or features selected, which makes it difficult to apply when the characters are overmuch. When the image has strong features, feature-based matching is considered to be efficient, especially when the image has large resolution. But this method does not view the image as an entirety, so it is highly limited by the image factors. Template-based matching
partly fills up the inability of feature-based matching since it is used for the image without strong features. To ensure the accuracy, the amount of sampled data point should be very large, thus resolution of the image may be reduced to reduce the required data points. Generally, template matching algorithm has high requirement for the processing image including low noise, high data point quality, suitable size and resolution. As the security codes are becoming more and more complex and irregular, it is difficult for template match algorithm to deal with these codes. Some machine learning algorithms are also used in security code recognition. However, these algorithms are usually based on risk minimization principle and used in classification and regression analysis with 1-dimension data. When it comes to 2-dimension security code images, it is difficult to use these models directly. The images should be converted into gray images and then transformed into black-and-white image with thresholding. Moreover, the features of the image must be extracted to convert the 2-dimension image data into 1-dimension feature data, which can be handled with. Generally speaking, machine learning methods like SVM has a relatively higher accuracy than template matching algorithm when the training samples increase in terms of security code recognition. However, as I mentioned, these traditional machine learning algorithms require feature extraction [3] process. It is difficult to figure out what features of the target images should be picked, and it is also hard to ensure the picked features can represent different types of security codes. Moreover, a lot of useful information may be lost in the process of feature extraction, which may decrease both the efficiency and accuracy of these methods.

Deep learning [4] is a machine learning algorithm based on neural network [5], which imitates the structure of cerebral cortex. The basic element of the neural network is neuron, and each neuron receives input signals from several other neurons. All the input signals are combined together with a given weight and then compared with the bias of current neuron. Finally, it generates the output by activation function. All the neurons connect together by layers to form the neuron network. Throughout training, the weight and bias of each neuron will be adjusted to fit the requirement. Convolution neural network is a type of neural network that works effectively in image processing by extracting feature maps through several convolution layers automatically. With the increase of convolutional layers, the extracted features become more and more complex and suitable for recognition. With the shared weight structure in convolution neural network, the number of parameters to be estimated reduces dramatically reduces and be training expense decreases significantly. Compared with aforementioned image recognition models, convolutional neural network is more suitable for security code recognition, because of the automatic feature extraction and higher learning ability.

In this paper, we constructed a convolutional neural network [6] for security code recognition. The security code images are generated by imitating the mechanism of the security code on the websites, and the image data is used for the training and testing of our proposed CNN model. Each security code image is made up of a combination of four characters or numbers, which are picked randomly. We also add noise lines and points to the image to make it more closed to the real-life security code. Before passing the image data into CNN model, we preprocessed the image by removing the noise point and line with coded filter and cut the image character by character. After preparing the training data set, we built the convolution neural network with three layers. After the three layers, we use a fully connected layer to flatten the image output from the previous layers and then summarize all the feature maps. A dropout layer is also adopted to avoid overfitting and increase the efficiency. After over 60000 iterations of training, we achieved an accuracy of about 99% for training set. Then we imported the test set and the accuracy is about 80% on the test set.

The rest of the article will discuss the following sections. In section 2, we will introduce how we generate our security code images for training and testing. In section 3, we will briefly talk about how we preprocess our image data before passing it into our CNN model. In section 4, we will discuss how the proposed convolution neural network for security code recognition is built. In section 5, we will go through the process of training our model and how it came out. Finally, we will discuss and evaluate our entire project. The related figures, tables and reference material will be provided.
2. Method/Result

2.1. Security Code Image Generation

To build and evaluate the image recognition model, we needed a large number of images. Although we can get security code images on the websites with the help of web crawler, it is time-consuming to label all the images with the true texts manually. Thus, we decided to imitate the generative mechanism of security code to generate images data, which makes it easier to control the parameters of our image and define the content in the security code images.

Taking the advantage of Python Image Library, it is easy to draw the security code images. A single security code image is made up of four random characters, each of which is either a number in 0-9 or letter in A-Z. For different images, we set different colors and fonts for the characters. Color is picked randomly out of RGB colors. We limited the character codes to be relatively darker in case it is too light to be recognized by human eyes, which is obviously inconsistent with the reality. The font is picked randomly from the system font library. We also added some noise lines and points on the image at random position. To make it closer to reality, we further adjusted the position range of noise line to make sure it can function as noise. The noise point with size of four pixels is set to be different colors randomly, so that it is visible enough. Finally, we affined and enhanced the images and they were ready for next progress.

![Figure 1. Example of security code image](image1)

2.2. Image preprocessing

In reality, the images are usually preprocessed to reduce the influence of noise such as line noise and point noise. We also preprocessed the image to remove the noise as much as possible to increase the predictability. As we introduced, the main noise involved in our image data is the noise lines and points. For noise points, we directly used a median filter from Python Image Library, which replaced each entry with the median value of neighboring entries. The example image after median filter is shown in the Figure 2, and we see that median filter can remove the noise points effectively.

![Figure 2. Image after median filter](image2)

Since noise line would cross the characters, a simple filter can’t remove it clearly. Thus, we used an algorithm to detect and remove these noise lines. First, we converted the image into binary image and append it into a data matrix to make it easier to process. Our algorithm will detect the two pixels...
above and two pixels right of each pixel in the image. If they are all black, the selected pixel will be identified as a component of a noise line, and it will be transferred to white. After passing through the filter algorithm, edge enhancement and smoothing will be executed to enhance the image. The outcome showed that most of the noise lines and points can be removed clearly without affecting the characters. The example image after the noise line removal is presented in the Figure 3. In some cases when the noise line nearly vertically crosses the characters, removal of it would cause the character to be separated into two parts. We fixed this drawback later in the character segmentation.

Figure 3. Image after noise line removal

To segment the images by characters, we set two stages: Foundletter and Inletter. Foundletter means the algorithm reaches the starting boundary of a single character, while Inletter means the cursor is currently in the character. Two stages are initially false. When starting to run the algorithm, we similarly pass through every pixel in the image from left to right. Once there is a black pixel and the current Foundletter is false, we change Foundletter to true and mark the position of the pixel as start. For Inletter stage, it is always true as long as the current pixel is black. Then as the algorithm going to right, if Inletter change to false because of white pixel and the Foundletter is still true for now. Foundletter will change to false and the current position will be marked as end meaning that it is the end of the character. Each pair of start and end will be stored in an array. After passing through the entire image, we used the array to identify the boundary of the characters.

As we mentioned, our noise line filter might separate a character into two parts, which would influence our segmentation algorithm. Thus, we added extra module by counting the number of pairs in the start and end array. If there are more or less than 4 pairs, it means that there exists erroneous segmentation. In this case, we used half of mean width of the pair length as reference and compare all the pairs in the array with it. If there are two continuous pairs that are smaller than the reference, they would be identified as the consequence of over segmentation. Then they would be merged together to form the complete pair. On the other hand, if there are fewer than four pairs, although this is less possible, the algorithm would segment the relatively long pair again to separate the joined characters. After segmentation, the preprocessing of the image is finished. The segmented characters will be saved to the related folder according to their content. Totally we have 36 folders for the 26 letters and 10 numbers. The exact code shown in the image is also saved for the subsequent classification in training and test.

2.3. Convolutional Neural Network

Our image recognition algorithm is based on Convolution Neural Network. It consists of three convolutional layers, a flatten layer and a full connection layer. The structure of our proposed CNN model is shown in Figure 4.

Before entering the network, the image would be reshaped to a 32×32 image matrix to unify the size of security code images. The first layer uses filters with the kernel size of 5×5, and there are totally 32 filters applied. For a single image passed into the first layer, there will be 32 feature maps generated from the first convolutional kernel. When executing the convolution, we set padding to be same so that the feature maps will be the same size as the initial image. The stride of convolution is set to be 1. Next step is the pooling of the generated feature maps. The kernel size of pooling is [1, 2, 2, 1]
and the strides is [1, 2, 2, 1]. The pooling of the feature maps will finally give us 32 16×16 feature maps which is the output of the first layer.

The filters of the second convolutional layer have the same size and other parameters are the same as those in the first layer. There are only 2 filters generated from the second convolutional kernel. The convolutional kernel in the second layer will increase the number of feature maps from 32 of the first layer to 64. Pooling and padding maintain the same setting as the first layer, which means that the output of the second layer are 64 8×8 feature maps.

![CNN structure diagram](image)

Figure 4. CNN structure diagram

Similarly, the third convolutional layer holds the filter with the same size and parameters. But there is only one filter applied in this layer. Pooling and padding maintain the same setting as the first layer. This means that after convolution and pooling, there are outputs of 64 4×4 feature maps from the third layer, which is the last convolution layer.

After three convolution layers, we built a fully connected layer to summarize all the output from the convolution layers. The first thing this layer does is to flatten the 64 4×4 feature maps from the third layer to vectors with the length of 1024, which in the other way is a 4×4×64 vector. Then all these vectors will be passed into activation function. A dropout layer also exists to randomly drop the neural unit from the CNN with a specific probability of 0.5. In Deep Learning, dropout layer is typically used to limit overfitting, which is a universal phenomenon for machine learning. Then the vectors will be transfer to probability vectors with the length of 36 in the following full-connected layer, and the result will be passed into the last activation function and give us the final output.

In our algorithm, calculation of loss and accuracy is included for us to monitor the training and testing process and determine the performance. For model training, we used TensorFlow [7] from Google as our library. First of all, we organized the input images into batches, each of which had 100 images. Totally, we iterated the training for 100000 times. Every 100 iterations, the current accuracy and loss were calculated so that we could monitor the real-time training result. When the accuracy reached 0.98, the model is potentially applicable, so we saved the existing model every time it iterated. When the accuracy reaches 0.995, it is highly possible that the model has reached its limited and the accuracy hardly increases. In that case, it is meaningless to keep iterating, so the algorithm stopped running, and the last saved model is our final product.

After the model training, we started to import different image set to test it. The test accuracy would be lower than the training accuracy because of overfitting. A validation set a potential adjustment that could be added to more limit overfitting [8].

### 2.4. Results

To analyze the process and result of training and testing, we used tensorboard [9] to visualize the learning trend, accuracy, loss and other related statistics of the network.
Figure 5 and 6 shows the accuracy and loss in the progress of training. We can see that the accuracy increased and the loss decreased rapidly within around first 15,000 iterations. Then the change tended to stop over the rest iterations. Finally, the accuracy is extremely close to 1 and the loss is almost down to 0.
Figure 7 and 8 on the other hand display the accuracy and loss for the testing set. As we can see, the general trend during the testing is similar to that of training. They had a rapid change at the beginning and when it got close to the limit, the change was negligible. What is different is that the upper limit of accuracy is around 0.8 and the floor level of the loss is around 0.1. There clearly exists gap between training and testing. This might because of overfitting in the training set. Overfitting means the model corresponds too exactly to a particular data set, which is our training data.

Other than the analysis of result, we also recorded other detailed statistics of deep learning model, including means, standard deviation, maximum value and minimum value of bias and weights. In the Figure 9 and 10, we show the corresponding statistics for the full-connected layer.
As we can see from the diagrams that the maximum value and standard deviation of bias increased, while mean and minimum value decreased as the training going through. Otherwise, maximum value of weight changed anomalously, and other parameters just share similar patterns with bias.

Figure 10. Min and standard deviation statistics of full connected layer

Figure 11. Distribution of bias

Figure 12. Distribution of final output
Figure 13. Distribution of weights

In the Figure 11, 12 and 13, we further present the distribution of bias, final output and weights in the full-connected layer. From the histogram, we can directly see the change trend and distribution of all the parameters of the model. Bias scattered over training while weights stayed in a relatively stable distribution. The final output, on the other hand, became more concentrated.

Such diagrams are really helpful to monitor the training process of our model. In most of cases, the neural network is like a black box. We set up how it should learn but we do not know what it finally becomes. These diagrams create a crevice on the black box so that we can peep inside instead of just pass data in.

3. Discussions and Conclusions

The initial goal of this project is to build a security code recognition algorithm that could be used for login and web crawler. Finally, based on Convolution Neural Network, a recognition model was born that can deal with security code images with noise lines and noise points. During the entire process, we did face several problems. For instance, we did not expect the error in segmentation at first because we ignored the potential influence the noise line could bring to segmentation. Thus, we had to add extra methods in the algorithm to fix the mistakes. Another typical negligence happened during training. As is well known, a complex deep learning model requires strong hardware and long enough time to train and the final accuracy it could reach is hardly predictable. We had a high expectation on the accuracy so that we set the break accuracy level to be too high, but we forgot to apply automatic model saving before the accuracy reached our expectation. The result was that the growing speed was tiny before the accuracy got the level, which made the training took an extremely long time to finish and we could not manually stop it because the model would not be saved. Finally, we had to start over to make the condition of the training more reasonable. These problems could be good reminders for deep learning model development. Except for these setbacks, the outcome of our model development turns out to be good. We achieved a training accuracy of 0.9902 and a testing accuracy of 0.805, which are high enough for basic security code recognition. Otherwise, this model still maintains high potential. This model only consists of 3 convolution layers. There are more layers that could be applied to increase the complexity of the network. Other than the structure of the network, some self-adjust functions could be used to change the learning and dropout rates to fit more situations and promote the behavior. The images we feed the model were basic security code images, there are different security code with more twists and noise in many websites. For now, we generated our own images, but web crawler is also an opinion to collect more types of security code images to make the model more comprehensive. Security code recognition is just a small application of image recognition by deep learning. There are so many potentials that we can excavate. Facial recognition, video recognition or even traffic recognition for automatic driving are coming to people’s life. The future of this area is bright and hopeful.

References
[1] Hanebeck U D. Template matching using fast normalized cross correlation[J]. Proceeding of Spie
on Optical Pattern Recognition XII, 2001, 4387:95-102.

[2] Schuldt C, Laptev I, Caputo B. Recognizing Human Actions: A Local SVM Approach[C] International Conference on Pattern Recognition. IEEE, 2004:32-36 Vol.3.

[3] Yuille A L, Cohen D S, Hallinan P W. Feature extraction from faces using deformable templates[C] Computer Vision and Pattern Recognition, 1989. Proceedings CVPR '89. IEEE Computer Society Conference on. IEEE, 1989:104-109.

[4] Schmidhuber J, rgen. Deep learning in neural networks[M]. Elsevier Science Ltd. 2015.

[5] Yan W Q, Wang J, Kankanhalli M S. Neural network-based face detection.[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 1998, 20(1):23-38.

[6] Li H, Lin Z, Shen X, et al. A convolutional neural network cascade for face detection[C]// Computer Vision and Pattern Recognition. IEEE, 2015:5325-5334.

[7] Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning[J]. 2016.

[8] Schaffer C. Overfitting avoidance as bias[C]// Sfi Tr. Santa Fe Institute, 1993:267--289.

[9] Gad A F. TensorFlow: A Guide to Build Artificial Neural Networks using Python[J]. 2017.