CNN Model for Verification Codes Recognition

—— Project for CS6200

Zhenyuan Xi

August, 2018

Background

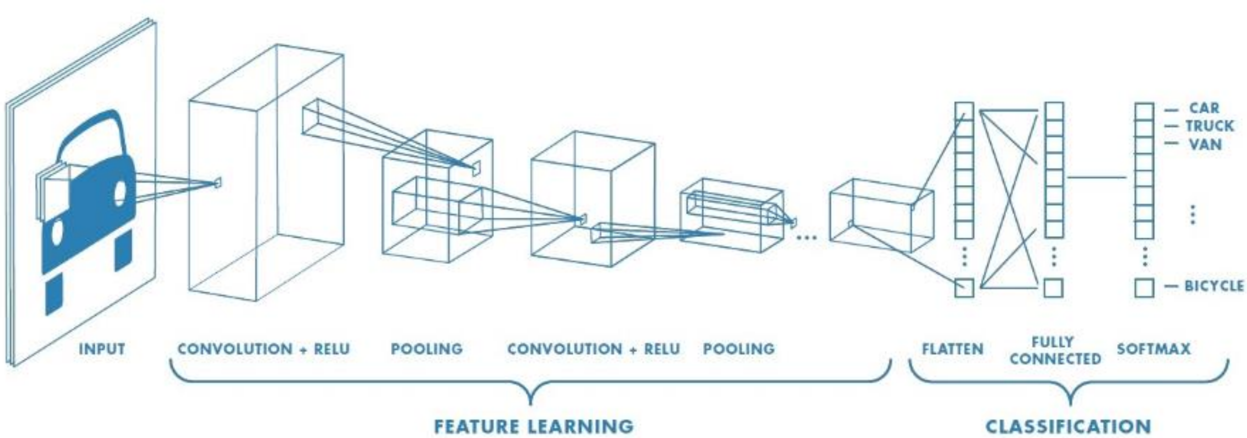
Verification code is widely used in daily life, especially when we login some websites, the administrators need it to recognize whether we are robot users or not. Here is a typical verification code:

This is actually an image with 5 places, in each place, there could be either upper case characters, lower case characters or digits. It's really easy to recognize what's in the picture with our eyes, but how can a machine do this job?

In neural networks, Convolutional neural network (CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications take an input image, process it and classify it under certain categories. Computer sees an input image as array of pixels and it depends on the image resolution.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



In my project, I plan to apply CNN modes to verification codes recognition. Verification code contains 4 places, all of them could be either digits from 0 to 9 or lower-case characters from a to z. My goal is to first generate all the image data, then split them into train and test sets. Then build several CNN models with different number of layers. After fitting training sets to the CNN model, evaluate the performance by precision.

Data Preprocessing

I choose to use Python Captcha library to generate image CAPTCHAs.

First, I initialize the two arrays representing the digits and lower-case characters and define the captcha length which is 4 and its height and width.

Then, I write two functions for generating the image CAPTCHAs. First one is to generate the captcha text which is a string type by randomly choose 4 items in the two arrays defined before. Second one is to generate image by applying captcha APIs to the randomly generated text string.

```
# initialize the arrays and constants

NUMBER = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
LOWER_CASE = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
              'n', 'o', 'p', 'q', 'r', 's', 't', 'u','v', 'w', 'x', 'y', 'z']
CAPTCHA_LIST = NUMBER + LOWER_CASE
CAPTCHA_LEN = 4
CAPTCHA_HEIGHT = 60
CAPTCHA_WIDTH = 160
```

```
# randomly choose four elements from captcha list which contains number and lower case characters

def random_captcha_text():
    captcha_text = [random.choice(CAPTCHA_LIST) for i in range(CAPTCHA_LEN)]
    return ''.join(captcha_text)
```

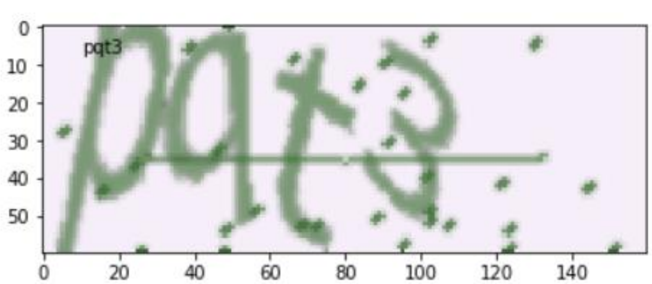
```
# use captcha to generate the text and image which is a numpy array

def gen_captcha_text_and_image():
    image = ImageCaptcha(width=CAPTCHA_WIDTH, height=CAPTCHA_HEIGHT)

    captcha_text = random_captcha_text()
    captcha = image.generate(captcha_text)

    captcha_image = Image.open(captcha)

    captcha_image = np.array(captcha_image)
    return captcha_text, captcha_image
```



Also, to better fit the image to the CNN models, I try to convert the RGB image to gray image since we do not need the color for the image recognition, my concern is the content in the image.

Because in the convolutional computation, the image should be converted into a matrix or vectors. I write a function to convert text string to vectors. Here I hardcode that by setting the index of the occurrence as 1 and other positions are all 0.

Based on the functions I defined before, I could combine them to a final generating next batch function which could generate the images, their flattened array and the vectors.

```
# convert 3D rgb to 1D grayscale for simplification

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

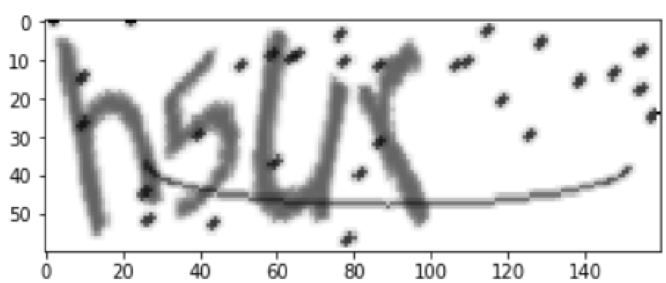
```
# convert text to vector for convolution

def text2vec(text):
    text_len = len(text)
    vector = np.zeros(CAPTCHA_LEN*len(CAPTCHA_LIST))
    for i in range(text_len):
        vector[CAPTCHA_LIST.index(text[i])+i*len(CAPTCHA_LIST)] = 1
    return vector
```

```
def next_batch(batch_size=100):
    batch_x = np.zeros([batch_size, CAPTCHA_HEIGHT * CAPTCHA_WIDTH])
    batch_y = np.zeros([batch_size, CAPTCHA_LEN * len(CAPTCHA_LIST)])

    for i in range(batch_size):
        text, image = gen_captcha_text_and_image()
        image = rgb2gray(image)
#         plt.imshow(image, cmap = plt.get_cmap('gray'))
#         plt.show()
        batch_x[i,:] = image.flatten() / 255 # standardize to 0-1 range since color uses 0-255 values
        batch_y[i,:] = text2vec(text)

    return batch_x, batch_y
```




```
[[0.98436078 0.31491373 0.31491373 ... 0.98436078 0.98436078 0.98436078]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Modeling

Use TensorFlow to define the CNN models.

1. Basically use tf.nn.conv2d(), tf.nn.bias_add() to add bias to value and compute a 2D convolution given 4D input and filter tensors. Also define the weight, strides and padding.

```
def weight(shape):
    initial = 0.01 * tf.random_normal(shape)
    return tf.Variable(initial)
```

```
def bias(shape):
    initial = 0.01 * tf.random_normal(shape)
    return tf.Variable(initial)
```

```
def conv(x, w):
    return tf.nn.conv2d(x, w, strides=[1, 1, 1, 1], padding='SAME')
```

2. I choose to use ReLU as the activation function to perform non-linear projection because ReLU has efficient computation and better gradient propagation.

3. Use pooling for non-linear down-sampling. I choose to implement pooling by the most common max pooling which would partition the input image into a set of non-overlapping rectangles and for each sub-region, outputs the maximum.

```
def max_pool(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

4. Also define dropout as a regularization technique for reducing overfitting by preventing complex co-adaptations on training data.

5. Define optimizer using sigmoid_cross_entropy rather than softmax to calculate the loss because sigmoid_cross is used for the case where every class is independent but not mutex while softmax_cross is used for that every class is independent and mutex.

```
def optimizer(y, y_conv):
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_conv, labels=y))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
    return optimizer
```

```
def accuracy(y, y_conv, width=len(CAPTCHA_LIST), height=CAPTCHA_LEN):
    predict = tf.reshape(y_conv, [-1, height, width])
    max_predict_idx = tf.argmax(predict, 2)
    label = tf.reshape(y, [-1, height, width])
    max_label_idx = tf.argmax(label, 2)
    correct_p = tf.equal(max_predict_idx, max_label_idx)
    accuracy = tf.reduce_mean(tf.cast(correct_p, tf.float32))
    return accuracy
```

6. Based on the functions defined before, build 2, 3 and 4 layers CNN models respectively.

```
def cnn_model(x, keep_prob, size, captcha_list=CAPTCHA_LIST, captcha_len=CAPTCHA_LEN):
    image_height, image_width = size
    x_image = tf.reshape(x, shape=[-1, image_height, image_width, 1])

    # layer 1
    w_conv1 = weight([3, 3, 1, 32])
    b_conv1 = bias([32])
    # ReLU
    h_conv1 = tf.nn.relu(tf.nn.bias_add(conv(x_image, w_conv1), b_conv1))
    # pooling
    h_pool1 = max_pool(h_conv1)
    # dropout
    h_drop1 = tf.nn.dropout(h_pool1, keep_prob)

    # layer 2
    w_conv2 = weight([3, 3, 32, 64])
    b_conv2 = bias([64])
    h_conv2 = tf.nn.relu(tf.nn.bias_add(conv(h_drop1, w_conv2), b_conv2))
    h_pool2 = max_pool(h_conv2)
    h_drop2 = tf.nn.dropout(h_pool2, keep_prob)

    # layer 3
    w_conv3 = weight([3, 3, 64, 64])
    b_conv3 = bias([64])
    h_conv3 = tf.nn.relu(tf.nn.bias_add(conv(h_drop2, w_conv3), b_conv3))
    h_pool3 = max_pool(h_conv3)
    h_drop3 = tf.nn.dropout(h_pool3, keep_prob)

    # full connected layer
    image_height = int(h_drop3.shape[1])
    image_width = int(h_drop3.shape[2])
    w_fc = weight([image_height*image_width*64, 1024])
    b_fc = bias([1024])
    h_drop3_re = tf.reshape(h_drop3, [-1, image_height*image_width*64])
    h_fc = tf.nn.relu(tf.add(tf.matmul(h_drop3_re, w_fc), b_fc))
    h_drop_fc = tf.nn.dropout(h_fc, keep_prob)

    # output layer
    w_out = weight([1024, len(captcha_list)*captcha_len])
    b_out = bias([len(captcha_list)*captcha_len])
    y_conv = tf.add(tf.matmul(h_drop_fc, w_out), b_out)
    return y_conv
```

7. Train the CNN model and for every 100 steps, test once and print out the current duration and accuracy to dynamically view the performance.

Performance

1. Two layers CNN model: total training time is 5 hours with 9400 steps, get 85% accuracy at 8600 steps, and after that, the accuracy fluctuated nearby 80%.

```
13:19:28 2018  step: 0  accuracy: 0.0225
13:23:39 2018  step: 100  accuracy: 0.0175
13:27:50 2018  step: 200  accuracy: 0.035
13:32:01 2018  step: 300  accuracy: 0.0275
13:36:09 2018  step: 400  accuracy: 0.01
13:40:17 2018  step: 500  accuracy: 0.03
13:44:32 2018  step: 600  accuracy: 0.0375
13:48:44 2018  step: 700  accuracy: 0.025
13:52:53 2018  step: 800  accuracy: 0.025
13:57:02 2018  step: 900  accuracy: 0.0275
14:01:11 2018  step: 1000  accuracy: 0.02

15:49:13 2018  step: 4000  accuracy: 0.5475
15:52:41 2018  step: 4100  accuracy: 0.51
15:56:05 2018  step: 4200  accuracy: 0.5875
15:59:36 2018  step: 4300  accuracy: 0.5575
16:03:05 2018  step: 4400  accuracy: 0.6025
16:06:35 2018  step: 4500  accuracy: 0.585
16:10:06 2018  step: 4600  accuracy: 0.595
16:13:36 2018  step: 4700  accuracy: 0.625
16:17:09 2018  step: 4800  accuracy: 0.6125
16:20:39 2018  step: 4900  accuracy: 0.56
16:24:10 2018  step: 5000  accuracy: 0.6075

17:40:43 2018  step: 8000  accuracy: 0.78
17:43:01 2018  step: 8100  accuracy: 0.7975
17:45:20 2018  step: 8200  accuracy: 0.785
17:47:45 2018  step: 8300  accuracy: 0.8225
17:51:45 2018  step: 8400  accuracy: 0.8175
17:56:02 2018  step: 8500  accuracy: 0.83
18:00:14 2018  step: 8600  accuracy: 0.85
18:04:29 2018  step: 8700  accuracy: 0.7825
18:08:42 2018  step: 8800  accuracy: 0.8
18:12:53 2018  step: 8900  accuracy: 0.7875
18:17:04 2018  step: 9000  accuracy: 0.8175
```

2. Three layers CNN model: total training time is 2.5 hours with 6100 steps to get the threshold 95% accuracy.

```
01:01:16 2018  step: 0  accuracy: 0.0125
01:03:54 2018  step: 100  accuracy: 0.025
01:06:37 2018  step: 200  accuracy: 0.025
01:09:17 2018  step: 300  accuracy: 0.02
01:11:56 2018  step: 400  accuracy: 0.0175
01:14:35 2018  step: 500  accuracy: 0.0325
01:17:14 2018  step: 600  accuracy: 0.0175
01:19:54 2018  step: 700  accuracy: 0.03
01:22:33 2018  step: 800  accuracy: 0.035
01:25:13 2018  step: 900  accuracy: 0.0325
01:27:53 2018  step: 1000  accuracy: 0.0375

02:21:04 2018  step: 3000  accuracy: 0.6775
02:23:48 2018  step: 3100  accuracy: 0.685
02:26:33 2018  step: 3200  accuracy: 0.71
02:29:13 2018  step: 3300  accuracy: 0.74
02:31:46 2018  step: 3400  accuracy: 0.785
02:34:25 2018  step: 3500  accuracy: 0.8025
02:36:53 2018  step: 3600  accuracy: 0.835
02:39:17 2018  step: 3700  accuracy: 0.78
02:41:41 2018  step: 3800  accuracy: 0.7925
02:44:01 2018  step: 3900  accuracy: 0.885
02:46:16 2018  step: 4000  accuracy: 0.81

03:10:50 2018  step: 5100  accuracy: 0.8875
03:13:04 2018  step: 5200  accuracy: 0.9275
03:15:19 2018  step: 5300  accuracy: 0.915
03:17:33 2018  step: 5400  accuracy: 0.93
03:19:49 2018  step: 5500  accuracy: 0.9225
03:22:03 2018  step: 5600  accuracy: 0.92
03:24:18 2018  step: 5700  accuracy: 0.91
03:26:32 2018  step: 5800  accuracy: 0.9275
03:28:45 2018  step: 5900  accuracy: 0.915
03:30:59 2018  step: 6000  accuracy: 0.9175
03:33:12 2018  step: 6100  accuracy: 0.96
```

3. Four layers CNN model: total training time is 3 hours with 7300 steps to get the threshold 95% accuracy.

```
20:14:54 2018  step: 0  accuracy: 0.03
20:17:05 2018  step: 100  accuracy: 0.0175
20:19:21 2018  step: 200  accuracy: 0.0275
20:21:39 2018  step: 300  accuracy: 0.0425
20:23:58 2018  step: 400  accuracy: 0.035
20:26:17 2018  step: 500  accuracy: 0.0275
20:28:36 2018  step: 600  accuracy: 0.025
20:30:55 2018  step: 700  accuracy: 0.0175
20:33:13 2018  step: 800  accuracy: 0.0375
20:35:31 2018  step: 900  accuracy: 0.025
20:37:48 2018  step: 1000  accuracy: 0.015

21:22:58 2018  step: 3000  accuracy: 0.0375
21:25:12 2018  step: 3100  accuracy: 0.0225
21:27:26 2018  step: 3200  accuracy: 0.0375
21:29:40 2018  step: 3300  accuracy: 0.0475
21:31:55 2018  step: 3400  accuracy: 0.15
21:34:09 2018  step: 3500  accuracy: 0.1875
21:36:23 2018  step: 3600  accuracy: 0.2625
21:38:37 2018  step: 3700  accuracy: 0.33
21:40:51 2018  step: 3800  accuracy: 0.46
21:43:29 2018  step: 3900  accuracy: 0.5
21:46:09 2018  step: 4000  accuracy: 0.6

22:49:55 2018  step: 6300  accuracy: 0.925
22:52:58 2018  step: 6400  accuracy: 0.94
22:55:54 2018  step: 6500  accuracy: 0.93
22:58:49 2018  step: 6600  accuracy: 0.9225
23:01:44 2018  step: 6700  accuracy: 0.935
23:04:32 2018  step: 6800  accuracy: 0.9125
23:07:43 2018  step: 6900  accuracy: 0.9275
23:10:53 2018  step: 7000  accuracy: 0.9425
23:14:03 2018  step: 7100  accuracy: 0.935
23:17:14 2018  step: 7200  accuracy: 0.9475
23:20:28 2018  step: 7300  accuracy: 0.9575
```

Conclusion

1. All these CNN models take ~3min to train for every 100 steps, which means the cost or efficiency of training different layers CNN models might be the same. The trade-off for more layers is generally the overfitting, but I think there could be more, here I found the cost of time is not a good one.

2. All three CNN models get a huge increasing in the accuracy around 2500 steps from ~0.02 to ~0.3, I think the probable reason might be at this step, CNN model becomes to learn some simple digits and characters, especially those with good symmetry, like 1 and o.

3. Three layers CNN model is the optimal one for the recognition of 4-place verification codes (CAPTCHA), it gets the 95% accuracy in 2.5 hours. In general, more layers mean more accurate and faster to get the expected accuracy, but it depends on the datasets, since here the datasets is not large (36^4=167,000), more layers might not lead to more efficient. So, for my project, 3 layers may be the best one.