“验证码识别”项目报告

翟润天　　林殷年　　李拙
1600012737　　1500016611　　1600012911

本项目由三位同学：翟润天、林殷年、李拙共同完成。其中翟润天同学负责项目总安排和编写测试服务器。林殷年同学负责设计学习模型和编写机器学习代码。李拙同学负责验证学习模型以及编写验证模块。

本项目 github 地址：https://github.com/RuntianZ/captcha.git

初期的项目研究（翟润天）

我们研究了大量使用图片验证码的网站，初步把验证码混淆技术分为四类：噪音、旋转、扭曲和艺术字。

噪音指的是验证码中包含一些像素点或花纹，覆盖在字符上方。例如：

Figure 1 **点噪音**

Figure 2 **点噪音和线噪音**

旋转指的是字符发生了随机小角度的整体旋转，例如：

Figure 3 **旋转**

扭曲指的是字符的某一部分发生了伸缩或旋转，例如：

Figure 4 **扭曲**

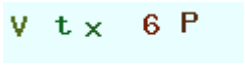
艺术字指的是验证码采用难以识别的字体，例如：

Figure 5 **空心字**

Figure 6 **手写体**

我们生成了一系列测试数据来测试我们的算法，并用 PHP 代码模拟了现今各大网站上较为流行的验证码生成算法，将验证码分为了几类，列举如下：

0、 无噪音的普通验证码



1、 少量点噪音的普通验证码

2、 大量点噪音、多字符的普通验证码

3、 少量点噪音、字符数量随机的验证码

4、 少量点和线噪音的普通验证码

5、 无噪音的旋转验证码

6、 少量点和线噪音的旋转验证码

7、 无噪音的空心验证码（部分字符可能有重叠）

8、 无噪音的手写体验证码

9、 少量点噪音的随机字体验证码

开始项目之前，我们对项目的文件系统大概做了规划如下：

./src 项目主要代码
    ./src 通用代码
    ./src/ml 机器学习代码
    ./src/test 单元测试代码
    ./src/server 服务器通信用代码
./test 样本相关代码
    ./test/src 初始学习样本生成代码
    ./test/server 服务器代码
./doc 项目说明文档

我们在 github 上建立了库用来进行版本和团队管理。

样本的生成和测试服务器的创建（翟润天）

为了进行有效的机器学习，我们需要大数据样本。我首先用 PHP 写了一些样本生成代码并在本地生成了每类验证码各 10000 份样本。

```php
/* The characters to choose from */
$chars = 'abcdefghijkmnpqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ123456789';
$maxn = 9;
$maximg = 10000;
$maxdot = 900;
$maxwidth = 180;
$maxheight = 30;
$ans = '';

for ($t = 0; $t < $maximg; $t++) {
    $string = '';
    for ($i = 0; $i < $maxn; $i++) {
        $rand = rand(0, strlen($chars) - 1);
        $string .= substr($chars, $rand, 1);
    }
    $ans .= $string."\r\n";
    $_SESSION['string'] = $string;
    $im = imagecreatetruecolor($maxwidth, $maxheight);
    $backcolor = imagecolorallocate($im, rand(220, 255), rand(220, 255), rand(220, 255));
    imagefilledrectangle($im, 0, 0, $maxwidth, $maxheight, $backcolor);

    /* Set dots */
    for ($i = 0; $i < $maxdot; ++$i) {
        $dotcolor = imagecolorallocate($im, rand(0, 255), rand(0, 255), rand(0, 255));
        $x = rand(0, $maxwidth);
        $y = rand(0, $maxheight);
        imagesetpixel($im, $x, $y, $dotcolor);
    }

    for ($i = 0; $i < strlen($string); ++$i) {
        $frontcolor = imagecolorallocate($im, rand(0, 120), rand(0, 120), rand(0, 120));
        imagestring($im, 10, rand(20 * $i + 1, 20 * $i + 10), rand(0, 5), substr($string, $i, 1), $frontcolor);
    }
```

Figure 7 初期的样本生成代码

后来为了进一步扩大样本量，我实现了一个样本生成服务器。只要向该服务器发送一个类别 id，服务器会自动生成一个验证码图片并返回它的 url 地址。然后，再向服务器发送一个字符串，服务器就会比对该字符串和真实的验证码，并返回 error_function 的结果（该函数在之后会详细讲述）。

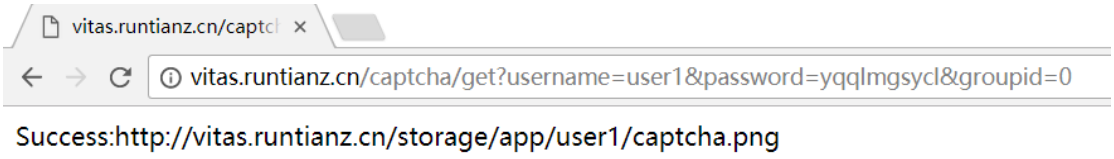


Figure 8 在浏览器上和服务器进行直接交互

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>> from server import *
>>> server.server_get('user', 'wrong_password', '1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "E:\captcha\src\server\server.py", line 33, in server_get
    raise ServerException(resp)
server.server.ServerException: Password wrong.
>>> server.server_get('user3', 'cptbtptp', '1')
'http://vitas.runtianz.cn/storage/app/user3/captcha.png'
>>> server.server_attempt('user3', 'cptbtptp', 'wrong')
0.0
>>> server.server_iterate('user3', 'cptbtptp', ['wrong1', 'wrong2'], [0])
[[0.0], [0.0]]
>>>
```

Figure 9 用 Python 进行封装过的交互

为了简化与服务器的通信，我编写了一个 server 包。通过这个包就可以实现登录服务器，从服务器获取验证码，以及测试验证码。

模型的选择和训练（林殷年）

训练集的准备

在构建 CNN 的时候，我们小组采用了多种的方法来生成/读取训练集：一是利用 python 的一个基于 Pillow 的名为 Captcha 的库，顾名思义，这个库的作用是用来生成给定大小和内容的验证码图片；二是利用实现事先生成好的图片作为训练集，大小约为 1GB；三是利用服务器实时生成，通过网络请求传输图片。三种方法各有优劣，其中，本地生成的训练集速度最快，所有操作都在内存完成，但是并不能控制验证码中噪音的类型，只能是点噪音+线噪音+旋转+重叠这个组合，而且由于文字内容也是随机生成的，所以下降速度有时需要看脸。利用事先生成好的训练集可以控制噪声的类型和文字内容，可以分开训练针对不同类型的验证码。利用服务器生成的训练数据非常慢，不推荐。

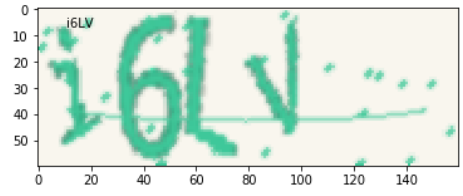


Figure 10 使用 Captcha 生成的验证码

图片的预处理

我们使用 CNN 进行验证码识别，而 CNN 的一大特点就是提取人难以理解的隐藏特征，所以在图片预处理这一部分我们的工作大大减少了。在我们的验证码识别系统中，要将一张图片转化成适合模型输入的，需要这几个步骤：图片 reshape。对每张输入图片，需要检查其是否符合规定的图片大小，因为如果图片的大小不固定的话，输入向量的大小就不确定。如

果输入图片不符合要求，我们会利用 Pillow 包将图片修改成符合要求的大小。Rgb 转灰度图。将原图的色彩信息去除，简化输入。色彩在验证码识别中没有什么作用，所以直接舍弃。
一维化。将灰度图片按像素点展开成一个一维向量。

```
52  def get_captcha_from_url(url, local):
53      if not local:
54          response = req.get(url)
55          image = Image.open(BytesIO(response.content))
56      else:
57          image = Image.open(url)
58      out = image.resize((160,60),Image.ANTIALIAS)
59      output = np.array(out)
60      return output
61
```

Figure 11 将图片转为需要的尺寸

```
105         text, image = wrap_gen_captcha_text_and_image()
106         image = convert2gray(image)
107
108         # flatten the image as CNN input. Ignore structure information.
109         batch_x[i,:] = image.flatten() / 255
```

Figure 12 一维化

```
30  # Convert the image to a grey-scale map. Color is useless in CNN.
31  def convert2gray(img):
32      if len(img.shape) > 2:
33          gray = np.mean(img, -1)
34          return gray
35      else:
36          return img
37
```

Figure 13 转为灰度图

每个 batch 包含 64 张验证码图，使用两个二维数组分别存储每张图的输入数据和真实文本。我们验证码的文本由大小写字母和数字组成，共计 62 个字符，所以可以用 62 个整数来编码字符。

CNN 网络的搭建

利用 tensorflow 提供的 api，可以便利地构造 CNN。首先我们设计了三层卷积层。第一层的输入是 160x60 的原图，在 1 个输入通道下，进行 3x3 的卷积，这个 3x3 的大小是我们借鉴了一些成熟 CV 领域 CNN 模型的设计决定的。我们在第一层设计了 32 个卷积核，第一层中的步长 (strides) 都是 1，步长不宜过大，否则容易淹没一些局部特征。我们使用 tf 的 bias_add 函数加上了一个由 32 维符合正态分布的向量定义的偏置。而激活函数我们选取的是 Relu 激活函数，这是一个比较万能的激活函数，可以比较好地防止梯度弥散，实际上我们从头到尾用的都是它。卷积运算过后我们在子采样中使用 max pooling 的方法，我们采用的是 2x2 的池化窗口和 1 的步长。在第一层我们还要进行 dropout 的操作，在一定概率下，输出会变成 0 而这个概率是由我们来定的，这个操作可以非常有效地防止过拟合。
第二层卷积中，有 32 个输入通道，64 个 3x3 的卷积核和 64 个正态分布的偏置数据，这时要注意，进行卷积操作，被放入 con2d 函数的第一个参数的不再是输入的原图 X，而应该是上一层的输出，在我们的代码中它被命名为 conv1，激活函数、子采样层和 dropout 的策略和第一层都没有变化。

第三层卷积中，有 64 个输入通道，64 个 3x3 的卷积核和 64 个正态分布的偏置数据，其余的操作和上一层相同。
三层卷积过后，是全连接层，这时我们有 8*20*64 的 batch，或者说是一行 8*20*64 个数据的卷积，这样的卷积一共有 1024 个，所以我们有 1024 个偏置数据。然后将第三层池化结果构造成一个一行只有 8*20*64 个数据的向量，也就是把卷积层的输出碾平。这时卷积操作其实可以用矩阵乘法来实现而不是用 tf.nn.con2d 这个函数，这个函数会遍历相乘，tf.matmul 实现的是基本的矩阵乘法，自动认为是前行向量乘后列向量。同样的在这里我们需要 dropout 操作。
最后第五层是输出层，首先构造一个二维张量[1024, size] 其中 size 是我们验证码可能取的字符的个数，可以看做是分类问题里的类别。因为是多分类问题，所以这里可以用 softmax，也可直接用 wx+b 的形式，这里我们用的是后者。最后输出的是一个矩阵，一共有 64 行（和 batch 的数目一样），有 size 列。

模型的训练、保存和使用

定义好了 CNN 网络中的各个层，就可以开始训练模型。
首先要定义 loss 函数，我们使用交叉熵来计算 loss 函数，在 tensorflow 里有一个现成的　函数，叫 tf.nn.sigmoid_cross_entropy_with_logits，这里我们传入两个参数，一个是我们预测的结果，另外一个是正确答案，再对 batch 内作平均，就得到了最终的 loss。
在定义好了 loss 之后，我们又选择了 Adam 作为优化函数，AdamOptimizer 的特点是收敛比较快，我们也考虑过 sgd+momentum 的组合，但是速度慢了一点，也看不出哪里有性能的显著提高，所以还是用了 Adam。
完成了这两部分的定义之后，就可以开始训练了。在 tensorflow 中训练模型需要在 session 中进行并保存。
为了方便地对模型进行保存和恢复，我在模型上安装了输入输出流。每次进行训练或者识别，只需要设定好输入输出流，就可以使用指定的模型。并且在训练中，我们可以用输入输出流对模型进行缓存。这样即使训练过程中遇到故障也可以方便地恢复训练。

```
def set_model_path(input_path = "", output_path = ""):
    '''
    set_model_path -     Set the default model paths.
    :param input_path:   The new model input path.
    :param output_path:  The new model output path.

    This method sets default input path and output path. The input path is used in initializing
    training model and performing recognition. The output path is used when saving trained
    model.
    '''
    global default_input_model_path, default_output_model_path
    default_input_model_path = input_path
    default_output_model_path = output_path
```

Figure 14 安装输入输出流
在此之后，我的想法是利用这个输入输出流，可以方便地将模型保存在远程服务器上。这样可以实现模型的分享，还可以服务器在不断训练模型的同时，本地端可以下载当前已经稳定的模型。我们为此更新了服务器代码，并写了一套网络通信方案。但是遗憾的是，由于模型过大，我们的服务器性能有限，因此不能很好地完成上述任务。这一点我们会在之后继续努力。

```php
public function upload(Request $request) {
    $checkinfo = $this->parse_login_info($request);
    if ($checkinfo != 'Success')
        return $checkinfo;
    $username = $this->username;
    $password = $this->password;

    if (!$request->has('filename'))
        return 'Argument error.';
    $file = $request->file('file');
    if (!$file->isValid())
        return 'File is not valid.';
    $filename = $request->get('filename');
    $filepath = 'storage/app/'.$username.'/';
    $filename = $filename.'.tar.gz';
    $file->move($filepath, $filename);
    return 'Success:'.$filepath.$filename;
}

public function download(Request $request) {
    $checkinfo = $this->parse_login_info($request);
    if ($checkinfo != 'Success')
        return $checkinfo;
    $username = $this->username;
    $password = $this->password;

    if (!$request->has('filename'))
        return 'Argument error.';
    $filename = $request->get('filename');
    $filepath = $username.'/'.$filename.'.tar.gz';
    if (!Storage::disk('local')->exists($filepath))
        return 'No such file';
    $filepath = 'storage/app/'.$filepath;
    return 'Success:'.$filepath;
}
```

Figure 15 服务器代码（节选）

模型的验证和单元测试（李拙）

error_function 的设计

在开始验证模型之前，我思考的问题是：什么样的模型称得上“好的模型”。
如果一个模型把"abcde"认成了"pqrst"，那么它当然不是一个好的模型。但是一个好的模型不一定每次都能够完全正确地识别每一个字符。例如把"xoxo"识别成"XOx0"的模型，未必不是好的。
这个问题对于验证环节是重要的，因为我们需要确定给予模型多大的奖励或惩罚。尤其是我们后期还打算使用增强学习，因此这一点就更为重要。将这一点抽象出来就是 error_function。这个函数接受两个字符串作为参数，返回的是它们的近似程度。这个近似需要考虑诸多方面，例如字符本身形状的相似（如 2 和 Z），以及字符和噪音之间的近似性（如把竖条状噪音识

别成了字符 l）等等。

```
/**
 * error_function           The error function that evaluates a recognition result.
 * @param recognize_result  The recognition result of an image.
 * @param answer            The real captcha of the same image.
 * @return                  A number in [0,1]. 0 means totally wrong, while 1 means correctly recognized.
 */
double error_function(char *recognize_result, char *answer);
```

Figure 16 error_function **的定义**

在开始编写这一函数之前，我们先写了一个测试函数，以达到测试驱动的目的。编写测试函数可以让我们知道所写的函数能否符合我们的需求。为了保证编写 error_function 函数不受到测试代码的干扰，两个不同的同学分别编写这两个函数。

```
char *test_error_function(double (*func)(char*, char*))
{
#define TASSERT(s) if (!(s)) break
    /* Test requirements that must be met */
    int flag = 0, cnt = 1;
    while (!flag) {

        /* 1. Must return a value between 0 and 1 */
        TASSERT(func("k", "l") >= 0 && func("k", "l") <= 1);
        TASSERT(func("", "uv123") >= 0 && func("", "uv123") <= 1);
        TASSERT(func("03f12", "") >= 0 && func("03f12", "") <= 1);
        TASSERT(func("p2vAB", "DC2fd") >= 0 && func("p2vAB", "DC2fd") <= 1);
        TASSERT(func("A77s9dQp", "ame0il") >= 0 && func("A77s9dQp", "ame0il") <= 1);
        TASSERT(func("s01WTdE", "1vxZ33p7W") >= 0 && func("s01WTdE", "1vxZ33p7W") <= 1);
        cnt = 2;

        /* 2. Must return 1 if strings are identical */
        TASSERT(func("", "") == 1);
        TASSERT(func("a", "a") == 1);
        TASSERT(func("1a2b", "1a2b") == 1);
        TASSERT(func("zzz", "zzz") == 1);
        cnt = 3;

        /* 3. Must not return 1 if not identical */
        TASSERT(func("a", "A") < 1);
        TASSERT(func("x", "X") < 1);
        TASSERT(func("0", "O") < 1);
```

Figure 17 error_function **的测试代码**

编写 error_function 的同学不知道测试代码的具体内容，但是获得了一份需求清单：

两字符串相等时必须返回 1。例如：

error_function('abcde', 'abcde') == 1

error_function('A1z90f', 'A1z90f') == 1

两字符串不相等时，返回值必须小于 1。例如：

error_function('abcde', 'aacde') < 1

error_function('abcd', 'abcde') < 1

error_function('aaaa', 'aaaA') < 1

看上去两字符串完全不等时必须返回 0。例如：

error_function('', 'abcde') == 0

error_function('12345', 'abcde') == 0

error_function('abcde', 'fff') == 0

近似程度可以从以下几方面考虑：
一是字符形状本身相近。例如小写 x 和大写 X 可以认为是很相近的，因此必须有
error_function('abxde', 'abXde') > error_function('abcde', 'abXde')
又例如数字 2 和大写 Z 也比较相近，因此必须有
error_function('222', 'ZZZ') > error_function('hhh', 'ZZZ')

二是少了或多了若干字符。例如
error_function('abicde', 'abcde') > error_function('ab', 'abcde')
error_function('aaa', 'aaaa') > error_function('aa', 'aaaa')

编写 error_function 的同学就是在这份需求清单的指引下编写这一函数的，并且编写了不同的版本，以便在实际中测试这些函数真实的强度。

error_function 的不同实现

1）改进过的 Levenshtein 算法(编辑距离)(源文件见 error_funtion_1.c)

编辑距离是指由一个字符串转换成另一个字符串所需要的最少编辑操作，允许的编辑操作包括字符的替换，删除，添加。如果使用 Levenshtein 算法，那么完全相同的字符串编辑距离是 0，完全不同的字符串编辑距离是 max(l1, l2)，对不完全相同的字符串则是一个小于 max(l1,l2)的正数，因此可以用 1-Edit_distance/max(l1, l2)来作为函数的返回值，由此可以满足函数的大部分要求。

```
double fullmark = max(l1, l2) * 100;
double res = (fullmark - dp[l1][l2]) / fullmark;
```

待解决的一点是两个字符串形状相似的问题。

为了解决这个问题，对 Levenshtein 算法做一些改进：

对三种编辑操作分别赋予不同的权重，添加和删除的代价最大，赋予 100，而将字符 a 替换为 b 的代价视字符的相似程度而定，在这个函数中将替换分为五个等级：相同、完全不同和相似程度不同的三个级别。编辑代价如下：

```
#define del_penalty 100
#define level1 80
#define level2 50
#define level3 20
```

计算时使用动态规划的思想，转移方程：

$$\mathrm{dp[i][j]} = \min\left(dp[i-1][j-1] + trans[s1[i]][s2[j]], dp[i-1][j] + del_{penalty}, dp[i][j-1] + del\_penalty\right);$$

在实际使用中，可以根据训练的效果对 leveli 的值进行适当的调整，例如：

```
#define level1 90
#define level2 40
#define level3 10
```

2）杰卡德相似度

杰卡德相似度是指两个集合的交集在并集中占的比例，即：

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

相应地，由杰卡德距离的概念：

$$J_\delta = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

将字符串看作集合，杰卡德相似度可以反映两个字符串的相似程度。当字符串完全一致时，J(s1, s2)显然为 1，完全不同时为 0.计算 J(s1, s2)的代码主要如下：

```
for (int i = 0; i < l1; ++i)
        a[recognize_result[i]]++;
for (int i = 0; i < l2; ++i)
        b[answer[i]]++;
int intersec = 0, unio = 0;
for (int i = 0; i < 256; ++i) {
        intersec += min(a[i], b[i]);
        unio += max(a[i], b[i]);
}
```

从思考和代码的角度都能发现一个问题，这种方法丢失了字符串中字符的顺序的信息，可能对不完全相同的字符串（例如"abc"和"cba"）返回 1.因此需要其他的方法弥补丢失的信息。

这里给杰卡德相似度(intersec / unio)乘以一个[0,1]的权重 LCS / min(l1,l2).LCS 表示两个字符串的最长公共字串的长度。在杰卡德相似度的基础上用 LCS / min(l1,l2)可以一定程度上反映两个字符串的顺序的信息。解决了杰卡德相似度为 1 但字符串不相同时返回 1 的问题。

但是这个方法无法解决字符的相似的问题，如果要优化，可以通过对余弦相似度进行类似 1）的改进，此处没有实现。

反观这个函数的实现，LCS 容易受字符的冗余和缺少的情况的干扰，而杰卡德相似度则可以弥补这一缺陷，因此将两者结合。

3）公共字串法

类似 2）的思想，考虑用 LCS 反映两个字符串的相似程度，需要解决的仍是字符串的冗余和缺少的情况，这里减弱对 LCS 的要求，计算允许一个字符的差的最长公共字串 LCS1。仍然用动态规划的方法：

```
for (int i = 1; i < l1; ++i)
        for (int j = 1; j < l2; ++j)
                if (recognize_result[i] == answer[j]) {
                        dp[i][j][0] = dp[i - 1][j - 1][0] + 1;
                        LCS0 = max(LCS0, dp[i][j][0]);
                        dp[i][j][1] = dp[i - 1][j - 1][1] + 1;
                        LCS1 = max(LCS1, dp[i][j][1]);
                }
                else {
                        dp[i][j][1] = max(dp[i - 1][j - 1][0], max(dp[i - 1][j][0], dp[i][j - 1][0])) + 1;
                        LCS1 = max(LCS1, dp[i][j][1]);
                }
```

返回的结果如下：

```
if (LCS0 == max(l1, l2))
        return 1;
return (double)LCS1 / max(l1, l2) * LCS0 / min(l1, l2);
```

这个实现与 2)的缺陷类似，无法反映字符的相似的情况。

3.测试情况与小结

在使用 test_error_function 测试 1）时，函数通过了所有的基本测试，在相似性的测试中，效果也很好。唯一的不足是没有体现字符串整体的性质，比如"vvv","000"的相似度应比"vvv","123"更高。而 2）和 3）对相似字符的情况处理都不好。

总的来说，实现 1）优于实现 2）和 3）.

4.遇到的问题

在实现 error_function 时遇到了以下问题：

① 如何决策哪些字符时相似的，并对相似度进行界定。也就是对 trans 矩阵的赋值的困难。这里采用的简单策略是根据主观判断进行赋值。

② 如何反映字符串的整体性质。思考可能要通过 hash 来实现，暂时没有想到好的解决方案。

③ 实现 2）和 3）怎么处理相似字符的问题。个人觉得可能还是要通过类似 1）的字符转移代价来解决。

使用 error_function 进行验证

首先我们要注册所写的函数。我们写了一套注册接口，当一个函数被注册后，就可以被服务器所用，来对模型进行验证。

```
def register_ef(func, docstr):
    '''
    register_ef -     Register an error function.
    :param func:      The function to be registered.
    :param docstr:    The doc string of the function.
    :return:          The internal version of this error function.
    '''
    func.__doc__ = docstr
    error_function_lib.append(func)
    error_function_docstr.append(docstr)
    return len(error_function_lib) - 1


def register_error_function(sofile, funcname, docstr):
    '''
    register_error_function - Register an error function from a shared library.
    :param sofile:            The shared library file name.
    :param funcname:          The function name.
    :param docstr:            The doc string of this function.
    :return:                  The internal version of this error function.

    Error functions are called using their internal version numbers. The version
    number 0 is reserved for the default error function. Other error functions
    can be registered using this function. The function should have the following
    header (in C):
    double new_function(char *result, char *ans)

    For example:
    >>> register_error_function('./ef.so', 'my_error_function', 'This is my function.')
    1
```

Figure 18 验证函数注册接口

接下来，修改我们的服务器，使其可以使用不同的验证函数对同一组验证码进行验证。这也有助于我们判断哪些验证函数是好的，哪些是需要改进的。

```
@cache_login
def server_iterate(username, password, results, versions):
    '''
    server_iterate -  To test a group of results using a group of error functions.
    :param username:  Username.
    :param password:  Password.
    :param results:   A list of results that will be tested.
    :param versions:  A list of error function versions that will be used.
    :return:          A list. Each item contains the results of different error
                      functions that test the same result.

    Use this method if you want to test on multiple results and use more than one
    error function. This function returns compare results in their input order.

    For example: If we have the following result
    Result     Version     Error function return
    'abc'      0           0.0
    'abc'      1           0.4
    'abc'      2           0.7
    'def'      0           0.0
    'def'      1           0.6
    'def'      2           0.9

    >>> server_iterate('user', 'password', ['def', 'abc'], [0, 2, 1])
    [[0.0, 0.9, 0.6], [0.0, 0.7, 0.4]]
    '''
```

Figure 19 用服务器进行验证

对已训练的模型进行评估

我们对用上述验证函数验证过的模型进行评估，结果如下：

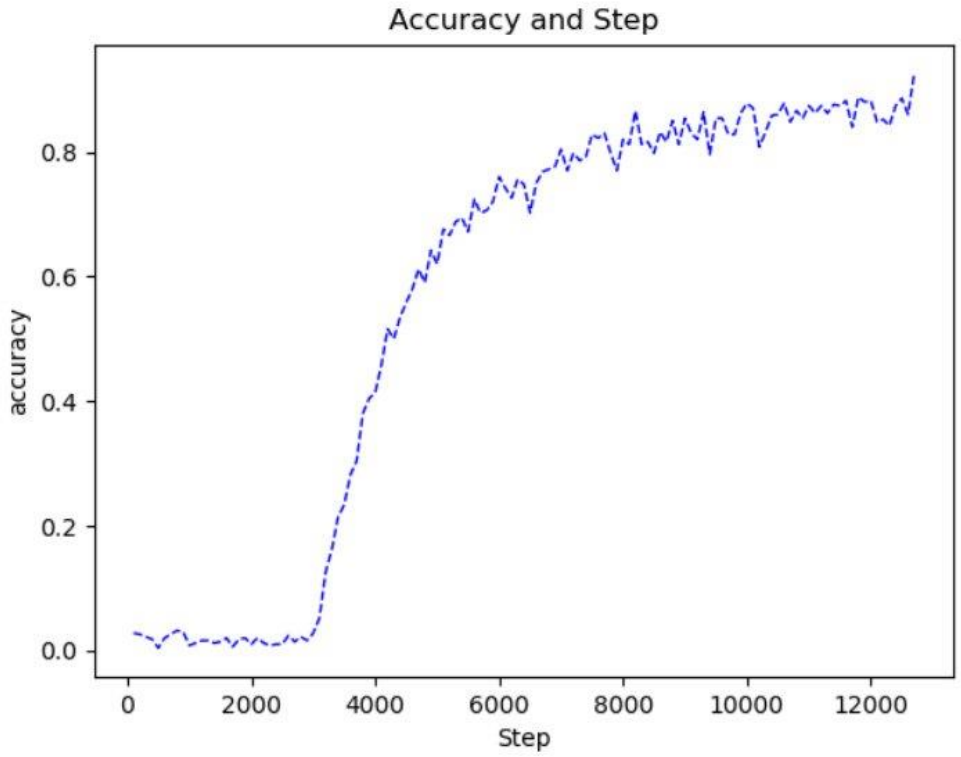


Figure 20 对模型进行评估

我们目前得到的最好模型是：

（1） 对于初始的四字符验证码和有轻微噪音的验证码可达到 98%准确率

（2） 对于有旋转和多噪音的验证码可达到 92%准确率

项目的单元测试

我们很注重项目的单元测试。对于我们所编写的每一个代码文件，我们都会编写相应的测试代码，并放置在专门的文件夹内。当我们修改代码时，就要重新运行测试代码，确保所写代码的正确性。

```
from server import *

server.server_login('user3', 'cptbtptp')
t = server.server_upload(path = 'model/model.ckpt', model_name = 'first')
print(t)
```

Figure 21 用于测试服务器的测试代码（节选）

项目的后期推进

- 我们尝试改进我们的服务器，使得它可以接受模型文件，并实现模型文件的分析和分享。
- 我们尝试使用 GPU 加速，以及多核加速技术。事实上，我们正向北大超算中心申请使用超级计算机进行我们的模型训练。
- 我们尝试使用其它的模型，以及增强学习技术。