3.基于LSTM+CTC实现不定长文本图片OCR

周小叨 (/u/421852a21f52) + 关注
0.1 2018.12.18 21:11* 字数 1075 阅读 194 评论 0 喜欢 1
(/u/421852a21f52)

上一篇实现了图片CNN多标签分类（4位定长验证码识别任务）（地址：https://www.jianshu.com/p/596db72a7e00 (https://www.jianshu.com/p/596db72a7e00)）

本文继续优化，实现不定长文本图片的识别任务

下一篇考虑玩一玩GAN网络

本文所用到的10w不定长验证码文本数据集百度网盘下载地址（也可使用下文代码自行生成）：https://pan.baidu.com/s/11BzIvuT4pYw3B0aFCK0ndQ (https://pan.baidu.com/s/11BzIvuT4pYw3B0aFCK0ndQ)

利用本文代码训练并生成的模型（对应项目中的my-model文件夹）：https://pan.baidu.com/s/1AoKtZVyscWp3ZdOQU71qLA (https://pan.baidu.com/s/1AoKtZVyscWp3ZdOQU71qLA)

项目简介：
需要预先安装pip install captcha==0.1.1,pip install opencv-python,pip install flask, pip install tensorflow/pip install tensorflow-gpu）
本文采用LSTM+CTC实现1-10位不定长验证码图片OCR（生成的验证码由随机的1-10位大写字母组成），本质上是一张图片多个标签的分类问题，且每个图片的标签数量不固定（数据如下图所示）

0_PIY.png

1_BCAVDPXT.png

2_N.png

整体训练逻辑：

1，将图像传入到LSTM中获得sequence，和sequence的长度（大致的原理是：将图像的width看做LSTM中的time_step，将图像的height看做每个time_step输入tensor的size）

2，将真实的y_label转为稀疏矩阵张量（此处的sparseTensor是个重点，同学们可以把代码中的153行y_train_tmp打印出来观察一下）

3，损失函数采用tf.nn.ctc_loss，然后对以上两步获得的数据进行训练，最终使得损失函数尽可能的减小

关于ctc_loss的原理可以百度科普一下，它的主要作用可以大概理解为将上层网络预测出的AAABBBBCCDEE收敛成ABBCDE，这里面牵涉到AAA到底收敛为几个A，BBBB又收敛为几个B，这也是他的核心

整体预测逻辑：

1，将图像传入到LSTM中获得sequence，和sequence的长度

2，将sequence，sequence的长度输入到tf.nn.ctc_beam_search_decoder函数预测出稀疏矩阵张量

3，将第二步得到的稀疏矩阵张量反向转化为sequence，并最终解码成A~Z的大写字母并输出

后续优化逻辑：

1，可以在LSTM之前先采用CNN对图像特征进行一次提取

2，TF自带的ctc_loss可以换成百度开源的Warp_CTC

3，针对少量原始图片为AAA结果最终识别为AA，丢掉了一个A的情况，是否可以把原先的标签['A', 'A', 'A']扩充为['A-left', 'A-middle', 'A-right', 'A-left', 'A-middle', 'A-right', 'A-left', 'A-middle', 'A-right']将每个字由原先的1个标签扩充为三个标签，此处抛砖引玉，可以自行尝试优化

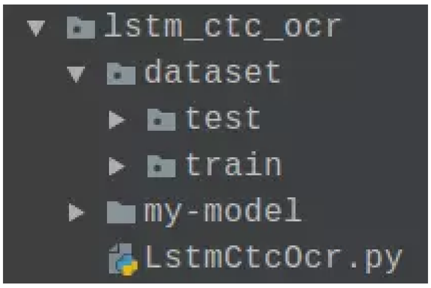
优缺点：

1，LSTM+CTC考虑了一行文本从左到右的序列关系，这一点上比CNN更强，同时可以轻松实现不定长的OCR

2，也正是由于RNN网络考虑了时序间的关系，所以运算量相对于CNN网络大幅增加，收敛比较慢，有条件的同学还是上一块好点的GPU吧，能提升很多效率

运行命令：
自行生成验证码训练寄（本文生成了10w张，修改self.im_total_num变量）：
python LstmCtcOcr.py create_dataset
对数据集进行训练：python LstmCtcOcr.py train
对新的图片进行测试：python LstmCtcOcr.py test
启动成http服务：python LstmCtcOcr.py start
利用flask框架将整个项目启动成web服务，使得项目支持http方式调用
启动服务后调用以下地址测试
http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/0_PIY.png
(http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/0_PIY.png)
http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/1_BCAVDPXT.png
(http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/1_BCAVDPXT.png)
http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/2_N.png
(http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/2_N.png)

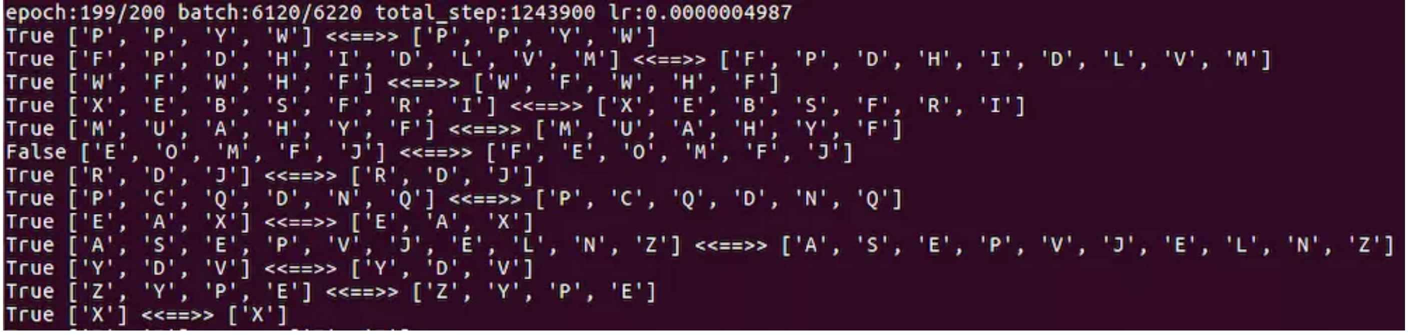
项目目录结构：



```
▼ lstm_ctc_ocr
  ▼ dataset
    ▶ test
    ▶ train
  ▶ my-model
    LstmCtcOcr.py
```

项目结构.png

训练200个epoch之后，可以看到model在val上的acc已经能达到84%了，后续大家可以自行修改学习率和增大epoch次数来提升精度（True表示预测正确，左边为预测值，右边为真实标签）：



```
epoch:199/200 batch:6120/6220 total_step:1243900 lr:0.0000004987
True ['P', 'P', 'Y', 'W'] <<==>> ['P', 'P', 'Y', 'W']
True ['F', 'P', 'D', 'H', 'I', 'D', 'L', 'V', 'M'] <<==>> ['F', 'P', 'D', 'H', 'I', 'D', 'L', 'V', 'M']
True ['W', 'F', 'W', 'H', 'F'] <<==>> ['W', 'F', 'W', 'H', 'F']
True ['X', 'E', 'B', 'S', 'F', 'R', 'I'] <<==>> ['X', 'E', 'B', 'S', 'F', 'R', 'I']
True ['M', 'U', 'A', 'H', 'Y', 'F'] <<==>> ['M', 'U', 'A', 'H', 'Y', 'F']
False ['E', 'O', 'M', 'F', 'J'] <<==>> ['F', 'E', 'O', 'M', 'F', 'J']
True ['R', 'D', 'J'] <<==>> ['R', 'D', 'J']
True ['P', 'C', 'Q', 'D', 'N', 'Q'] <<==>> ['P', 'C', 'Q', 'D', 'N', 'Q']
True ['E', 'A', 'X'] <<==>> ['E', 'A', 'X']
True ['A', 'S', 'E', 'P', 'V', 'J', 'E', 'L', 'N', 'Z'] <<==>> ['A', 'S', 'E', 'P', 'V', 'J', 'E', 'L', 'N', 'Z']
True ['Y', 'D', 'V'] <<==>> ['Y', 'D', 'V']
True ['Z', 'Y', 'P', 'E'] <<==>> ['Z', 'Y', 'P', 'E']
True ['X'] <<==>> ['X']
```

lstm-ctc-199-epoch.png

整体代码如下（LstmCtcOcr.py文件）：

```python
# coding:utf-8

from captcha.image import ImageCaptcha
import numpy as np
import cv2
import tensorflow as tf
import random, os, sys
import operator


from flask import request
from flask import Flask
import json
app = Flask(__name__)

class LstmCtcOcr:
    def __init__(self):
        self.epoch_max = 200  # 最大迭代epoch次数
        self.batch_size = 16  # 训练时每个批次参与训练的图像数目，显存不足的可以调小
        self.lr = 5e-5  # 初始学习率
        self.save_epoch = 5  # 每相隔多少个epoch保存一次模型
        self.n_hidden = 256  # 隐藏神经元个数

        self.im_width = 256
        self.im_height = 64
        self.im_total_num = 100000  # 总共生成的验证码图片数量
        self.train_max_num = self.im_total_num  # 训练时读取的最大图片数目
        self.val_num = 30 * self.batch_size  # 不能大于self.train_max_num  做验
        self.words_max_num = 10  # 每张验证码图片上的最大字母个数
        self.words = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
        self.n_classes = len(self.words) + 1  # 26个字母 + blank
        self.x = None
        self.y = None

    def captchaOcr(self, img_path):
        """
        验证码识别
        :param img_path:
        :return:
        """
        im = cv2.imread(img_path)
        im = cv2.resize(im, (self.im_width, self.im_height))
        im = np.array([im[:, :, 0]], dtype=np.float32)
        im -= 147
        pred = self.sess.run(self.pred, feed_dict={self.x: im})
        sequence = self.sparseTensor2sequence(pred)
        return ''.join(sequence[0])


    def test(self, img_path):
        """
        测试接口
        :param img_path:
        :return:
        """
        self.batch_size = 1
        self.learning_rate = tf.placeholder(dtype=tf.float32)  # 动态学习率
        self.weight = tf.Variable(tf.truncated_normal([self.n_hidden, self.n_
        self.bias = tf.Variable(tf.constant(0., shape=[self.n_classes]))
        self.x = tf.placeholder(tf.float32, [None, self.im_height, self.im_wi
        logits, seq_len = self.rnnNet(self.x, self.weight, self.bias)
        decoded, log_prob = tf.nn.ctc_beam_search_decoder(logits, seq_len, me
        self.pred = tf.cast(decoded[0], tf.int32)

        saver = tf.train.Saver()
        # tfconfig = tf.ConfigProto(allow_soft_placement=True)
        # tfconfig.gpu_options.per_process_gpu_memory_fraction = 0.3  # 占用显
        # self.ses = tf.Session(config=tfconfig)
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())  # 全局tf变量初始化

        # 加载w,b参数
        saver.restore(self.sess, './my-model/LstmCtcOcr-200')
        im = cv2.imread(img_path)
        im = cv2.resize(im, (self.im_width, self.im_height))
        im = np.array([im[:, :, 0]], dtype=np.float32)
        im -= 147
        pred = self.sess.run(self.pred, feed_dict={self.x: im})
        sequence = self.sparseTensor2sequence(pred)
        print(''.join(sequence[0]))


    def train(self):
        """
        训练
        :return:
        """
        x_train_list, y_train_list, x_val_list, y_val_list = self.getTrainDat

        print('开始转换tensor队列')
        x_train_list_tensor = tf.convert_to_tensor(x_train_list, dtype=tf.str
        y_train_list_tensor = tf.convert_to_tensor(y_train_list, dtype=tf.int

        x_val_list_tensor = tf.convert_to_tensor(x_val_list, dtype=tf.string)
        y_val_list_tensor = tf.convert_to_tensor(y_val_list, dtype=tf.int32)

        x_train_queue = tf.train.slice_input_producer(tensor_list=[x_train_li
        y_train_queue = tf.train.slice_input_producer(tensor_list=[y_train_li

        x_val_queue = tf.train.slice_input_producer(tensor_list=[x_val_list_t
        y_val_queue = tf.train.slice_input_producer(tensor_list=[y_val_list_t

        train_im, train_label = self.dataset_opt(x_train_queue, y_train_queue
        train_batch = tf.train.batch(tensors=[train_im, train_label], batch_s

        val_im, val_label = self.dataset_opt(x_val_queue, y_val_queue)
        val_batch = tf.train.batch(tensors=[val_im, val_label], batch_size=se

        print('准备训练')
        self.learning_rate = tf.placeholder(dtype=tf.float32)  # 动态学习率
        self.weight = tf.Variable(tf.truncated_normal([self.n_hidden, self.n_
        self.bias = tf.Variable(tf.constant(0., shape=[self.n_classes]))

        # self.global_step = tf.Variable(0, trainable=False)  # 全局步骤计数

        # im_width看成LSTM的time_step，im_height看成是每个time_step输入tensor的s
        self.x = tf.placeholder(tf.float32, [None, self.im_height, self.im_wi
        # 定义ctc_loss需要的稀疏矩阵
        self.y = tf.sparse_placeholder(tf.int32)

        logits, seq_len = self.rnnNet(self.x, self.weight, self.bias)

        # loss
        self.loss = tf.nn.ctc_loss(self.y, logits, seq_len)
        # cost
        self.cost = tf.reduce_mean(self.loss)
        # optimizer
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_r


        # 前面说的划分块之后找每块的类属概率分布，ctc_beam_search_decoder方法,是每次找
        # 还有一种贪心策略是只找概率最大那个，也就是K=1的情况ctc_ greedy_decoder
        decoded, log_prob = tf.nn.ctc_beam_search_decoder(logits, seq_len, me
        self.pred = tf.cast(decoded[0], tf.int32)
        self.distance = tf.reduce_mean(tf.edit_distance(self.pred, self.y))

        print('开始训练')
        saver = tf.train.Saver()  # 保存tf模型
        with tf.Session() as self.sess:
            self.sess.run(tf.global_variables_initializer())
            coordinator = tf.train.Coordinator()
            threads = tf.train.start_queue_runners(sess=self.sess, coord=coor

            batch_max = len(x_train_list) // self.batch_size
            print('batch:', batch_max)
            total_step = 0
            for epoch_num in range(self.epoch_max):
                lr_tmp = self.lr * (1 - (epoch_num / self.epoch_max) ** 2)  #
                print('lr:', lr_tmp)
                for batch_num in range(batch_max):
                    # print(epoch_num, batch_num)
                    x_train_tmp, y_train_tmp = self.sess.run(train_batch)
                    y_train_tmp = self.sequence2sparseTensor(y_train_tmp)  #
                    self.sess.run(self.optimizer, feed_dict={self.x: x_train_

                    if total_step % 100 == 0 or total_step == 0:
                        print('epoch:%d/%d batch:%d/%d total_step:%d lr:%.10f
                        # train部分
                        train_loss, train_distance = self.sess.run([self.cost

                        # val部分
                        val_loss_list, val_distance_list, val_acc_list = [],
                        for i in range(int(self.val_num / self.batch_size)):
                            x_val_tmp, y_val_tmp_true = self.sess.run(val_bat
                            y_val_tmp = self.sequence2sparseTensor(y_val_tmp_
                            val_loss, val_distance, val_pred = self.sess.run(
                            val_loss_list.append(val_loss)
                            val_distance_list.append(val_distance)
                            val_sequence = self.sparseTensor2sequence(val_pre
                            ok = 0.
                            for idx, val_seq in enumerate(val_sequence):
                                val_pred_tmp = [self.words.find(x) if self.wo
                                val_y_true_tmp = [x for x in y_val_tmp_true[i

                                is_eq = operator.eq(val_pred_tmp, val_y_true_

                                if idx == 0:
                                    print(is_eq, [self.words[n] for n in val_

                                if is_eq:
                                    ok += 1
                            val_acc_list.append(ok / len(val_sequence))
                        val_acc_list = np.array(val_acc_list, dtype=np.float3

                        print('train_loss:%.10f train_distance:%.10f' % (trai
                        print('  val_loss:%.10f   val_distance:%.10f val_acc:
                        print()
                        print()

                    total_step += 1

                # 保存模型
                if (epoch_num + 1) % self.save_epoch == 0:
                    saver.save(self.sess, './my-model/LstmCtcOcr', global_ste

            coordinator.request_stop()
            coordinator.join(threads)


    def rnnNet(self, inputs, weight, bias):
        """
        获取LSTM网络结构
        :param inputs:
        :param weight:
        :param bias:
        :return:
        """
        # 对于tf.nn.dynamic_rnn，默认time_major=false，此时inputs的shape=[batch_
        # (batch_size, im_height, im_width) ==> (batch_size, im_width, im_hei
        inputs = tf.transpose(inputs, [0, 2, 1])

        # 变长序列的最大值
        # seq_len = np.ones(self.batch_size) * self.im_width
        seq_len = np.ones(self.batch_size) * self.im_width

        cell = tf.nn.rnn_cell.LSTMCell(self.n_hidden, forget_bias=0.8, state_

        # 动态rnn实现输入变长
        outputs1, _ = tf.nn.dynamic_rnn(cell, inputs, seq_len, dtype=tf.float

        # (self.batch_size * self.im_width, self.hidden)
        outputs = tf.reshape(outputs1, [-1, self.n_hidden])

        logits = tf.matmul(outputs, weight) + bias  # w * x + b
        logits = tf.reshape(logits, [self.batch_size, -1, self.n_classes])
        logits = tf.transpose(logits, (1, 0, 2))  # (im_width, batch_size, im
        return logits, seq_len


    def sequence2sparseTensor(self, sequences, dtype=np.int32):
        """
        序列 转化为 稀疏矩阵
        :param sequences:
        :param dtype:
        :return:
        """
        values, indices= [], []
        for n, seq in enumerate(sequences):
            indices.extend(zip([n] * len(seq), range(len(seq))))
            values.extend(seq)
        indices = np.asarray(indices, dtype=np.int64)
        values = np.asarray(values, dtype=dtype)
        shape = np.asarray([len(sequences), np.asarray(indices).max(0)[1] + 1
        return indices, values, shape


    def sparseTensor2sequence(self, sparse_tensor):
        """
        稀疏矩阵 转化为 序列
        :param sparse_tensor:
        :return:
        """
        decoded_indexes = list()
        current_i = 0
        current_seq = []
        for offset, i_and_index in enumerate(sparse_tensor[0]):
            i = i_and_index[0]
            if i != current_i:
                decoded_indexes.append(current_seq)
                current_i = i
                current_seq = list()
            current_seq.append(offset)
        decoded_indexes.append(current_seq)
        result = []
        for index in decoded_indexes:
            result.append(self.sequence2words(index, sparse_tensor))
        return result


    def sequence2words(self, indexes, spars_tensor):
        """
        序列 转化为 文本
        :param indexes:
        :param spars_tensor:
        :return:
        """
        decoded = []
        for m in indexes:
            str_tmp = self.words[spars_tensor[1][m]]
            decoded.append(str_tmp)
        return decoded


    def dataset_opt(self, x_train_queue, y_train_queue):
        """
        处理图片和标签
        :param queue:
        :return:
        """
        queue = x_train_queue[0]
        contents = tf.read_file('./dataset/train/' + queue)
        im = tf.image.decode_jpeg(contents)
        tf.image.rgb_to_grayscale(im)
        im = tf.image.resize_images(images=im, size=[self.im_height, self.im_
        im = tf.reshape(im[:, :, 0], tf.stack([self.im_height, self.im_width]
        im -= 147  # 去均值化
        return im, y_train_queue[0]


    def getTrainDataset(self):
        train_data_list = os.listdir('./dataset/train/')
        print('共有%d张训练图片， 读取%d张：' % (len(train_data_list), self.train_
        random.shuffle(train_data_list)  # 打乱顺序

        y_val_list, y_train_list = [], []
        x_val_list = train_data_list[:self.val_num]
        for x_val in x_val_list:
            words_tmp = x_val.split('.')[0].split('_')[1]
            words_tmp = words_tmp + '?' * (self.words_max_num - len(words_tmp
            y_val_list.append([self.words.find(x) if self.words.find(x) > -1

        x_train_list = train_data_list[self.val_num:self.train_max_num]
        for x_train in x_train_list:
            words_tmp = x_train.split('.')[0].split('_')[1]
            words_tmp = words_tmp + '?' * (self.words_max_num - len(words_tmp
            y_train_list.append([self.words.find(x) if self.words.find(x) > -

        return x_train_list, y_train_list, x_val_list, y_val_list


    def createCaptchaDataset(self):
        """
        生成训练用图片数据集
        :return:
        """
        image = ImageCaptcha(width=self.im_width, height=self.im_height, font
        for i in range(self.im_total_num):
            words_tmp = ''
            for j in range(random.randint(1, self.words_max_num)):
                words_tmp = words_tmp + random.choice(self.words)
            print(words_tmp, type(words_tmp))
            im_path = './dataset/train/%d_%s.png' % (i, words_tmp)
            print(im_path)
            image.write(words_tmp, im_path)




if __name__ == '__main__':
    opt_type = sys.argv[1:][0]

    instance = LstmCtcOcr()

    if opt_type == 'create_dataset':
        instance.createCaptchaDataset()
    elif opt_type == 'train':
        instance.train()
    elif opt_type == 'test':
        instance.test('./dataset/test/0_PIY.png')
    elif opt_type == 'start':
        # 将session持久化到内存中
        instance.test('./dataset/test/0_PIY.png')

        # 启动web服务
        # http://127.0.0.1:5050/captchaOcr?img_path=./dataset/test/1_BCAVDPXT
        @app.route('/captchaOcr', methods=['GET'])
        def captchaOcr():
            img_path = request.args.to_dict().get('img_path')
            print(img_path)
            ret = instance.captchaOcr(img_path)
            print(ret)
            return json.dumps({'img_path': img_path, 'ocr_ret': ret})

        app.run(host='0.0.0.0', port=5050, debug=False)
```