超速计算器

发表于 2021-01-04 分类于 Challenge ， 2019 ， UNCTF ， Misc
Challenge | 2019 | UNCTF | Misc | 超速计算器

点击此处获得更好的阅读体验

解题思路

问题分析

打开首页，是一道计算器的题目，需要计算表达式，并提交结果，如图。
因为表达式是图片，需要先识别图片，再执行表达式计算结果。如果要训练模型需要大量的标注数据，看看能不能自己生成验证码数据进行训练，会方便很多。

访问/robots.txt,看到有一个code.py文件禁止爬虫访问，访问code.py，是生成验证码的代码。在代码中有用到Chopsic.ttf,访问/Chopsic.ttf获取到字体文件。然后使用code.py就可以本地生成验证码。

验证码识别

使用现成的captcha项目生成模型，这里使用captcha_trainer进行识别,支持不定长字符的识别。 按照说明下载代码，安装依赖。

数据集的准备

使用python脚本生成图片文件，文件名为验证码图片的文字：

```
import os
from code import gen_exp_pic
def make_dataset(pic_path, count=10000):
    os.makedirs(pic_path, exist_ok=True)
    for i in range(count):
        r = gen_exp_pic()
        target_file = os.path.join(pic_path, r[1]+"_.jpg")
        r[0].save(target_file)
datasets_dir = "datasets/"
make_dataset(datasets_dir, count=5000)
```

生成dataset图片之后，再使用python make_dataset.py生成测试和训练数据集。在生成数据集之前要先配置模型信息:

```
# - requirement.txt - GPU: tensorflow-gpu, CPU: tensorflow
# - If you use the GPU version, you need to install some additional applications.
System:
DeviceUsage: 0.9
# ModelName: Corresponding to the model file in the model directory,
# - such as YourModelName.pb, fill in YourModelName here.
# CharSet: Provides a default optional built-in solution:
# - [ALPHANUMERIC, ALPHANUMERIC_LOWER, ALPHANUMERIC_UPPER,
# -- NUMERIC, ALPHABET_LOWER, ALPHABET_UPPER, ALPHABET,
ALPHANUMERIC_LOWER_MIX_CHINESE_3500]
# - Or you can use your own customized character set like: ['a', '1', '2'].
# CharMaxLength: Maximum length of characters,  used for label padding.
# CharExclude: CharExclude should be a list, like: ['a', '1', '2']
# - which is convenient for users to freely combine character sets.
# - If you don't want to manually define the character set manually,
# - you can choose a built-in character set
# - and set the characters to be excluded by CharExclude parameter.
Model:
Sites: [
'ocr3step'
]
ModelName: ocr3step
ModelType: 400x32
# 支持的字符集，这里要识别的运算符号只有+*-
CharSet: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '*', '-']
# 识别的最长字符数
CharMaxLength: 11
CharExclude: []
CharReplace: {}
ImageWidth: 400
ImageHeight: 32
# Binaryzation: [-1: Off, >0 and < 255: On].
# Smoothing: [-1: Off, >0: On].
# Blur: [-1: Off, >0: On].
# Resize: [WIDTH, HEIGHT]
# - If the image size is too small, the training effect will be poor and you need to zoom in.
# ReplaceTransparent: [True, False]
# - True: Convert transparent images in RGBA format to opaque RGB format,
# - False: Keep the original image
Pretreatment:
Binaryzation: -1
Smoothing: -1
Blur: -1
Resize: [400, 32]
ReplaceTransparent: True
# CNNNetwork: [CNN5, ResNet, DenseNet]
# RecurrentNetwork: [BLSTM, LSTM, SRU, BSRU, GRU]
# - The recommended configuration is CNN5+BLSTM / ResNet+BLSTM
# HiddenNum: [64, 128, 256]
# - This parameter indicates the number of nodes used to remember and store past states.
# Optimizer: Loss function algorithm for calculating gradient.
```

```
# - [AdaBound, Adam, Momentum]
NeuralNet:
CNNNetwork: CNN5
RecurrentNetwork: BLSTM
HiddenNum: 64
KeepProb: 0.98
Optimizer: AdaBound
PreprocessCollapseRepeated: False
CTCMergeRepeated: True
CTCBeamWidth: 1
CTCTopPaths: 1
WarpCTC: False
# TrainsPath and TestPath: The local absolute path of your training and testing set.
# DatasetPath: Package a sample of the TFRecords format from this path.
# TrainRegex and TestRegex: Default matching apple_20181010121212.jpg file.
# - The Default is .*?(?=_.*.)
# TestSetNum: This is an optional parameter that is used when you want to extract some of the test set
# - from the training set when you are not preparing the test set separately.
# SavedSteps: A Session.run() execution is called a Step,
# - Used to save training progress, Default value is 100.
# ValidationSteps: Used to calculate accuracy, Default value is 500.
# TestSetNum: The number of test sets, if an automatic allocation strategy is used (TestPath not set).
# EndAcc: Finish the training when the accuracy reaches [EndAcc*100]% and other conditions.
# EndCost: Finish the training when the cost reaches EndCost and other conditions.
# EndEpochs: Finish the training when the epoch is greater than the defined epoch and other conditions.
# BatchSize: Number of samples selected for one training step.
# TestBatchSize: Number of samples selected for one validation step.
# LearningRate: Recommended value[0.01: MomentumOptimizer/AdamOptimizer, 0.001: AdaBoundOptimizer]
Trains:
# 训练数据集的路径
TrainsPath: './dataset/ocr3step_trains.tfrecords'
# 测试数据集的路径
TestPath: './dataset/ocr3step_test.tfrecords'
# 生成的图片文件的路径
DatasetPath: [
"./datasets/"
]
TrainRegex: '.*?(?=_)' # 提取图片label的正则表达式
TestSetNum: 200
SavedSteps: 100
ValidationSteps: 500
EndAcc: 0.95
EndCost: 0.1
EndEpochs: 2
BatchSize: 30 # 根据本机性能调整
TestBatchSize: 15 # 根据本机性能调整
LearningRate: 0.001
DecayRate: 0.98
DecaySteps: 10000
```

训练模型

生成数据集之后就是训练了，使用上面的模型配置，运行`python train.py`直接训练。使用GeForce GTX 1050 Ti跑了3分钟，完成训练。

使用模型预测

修改`predict_testing.py`,添加一次预测一张图片的函数，保存为`predict.py`，代码如下:

```python
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
# Author: kerlomz <kerlomz@gmail.com>
import io
import cv2
import numpy as np
import PIL.Image as PIL_Image
import tensorflow as tf
from importlib import import_module
from config import *
from constants import RunMode
from pretreatment import preprocessing
from framework import GraphOCR
def get_image_batch(img_bytes):
    def load_image(image_bytes):
        data_stream = io.BytesIO(image_bytes)
        pil_image = PIL_Image.open(data_stream)
        rgb = pil_image.split()
        size = pil_image.size
        if len(rgb) > 3 and REPLACE_TRANSPARENT:
            background = PIL_Image.new('RGB', pil_image.size, (255, 255, 255))
            background.paste(pil_image, (0, 0, size[0], size[1]), pil_image)
            pil_image = background
        if IMAGE_CHANNEL == 1:
            pil_image = pil_image.convert('L')
        im = np.array(pil_image)
        im = preprocessing(im, BINARYZATION, SMOOTH, BLUR).astype(np.float32)
        if RESIZE[0] == -1:
            ratio = RESIZE[1] / size[1]
            resize_width = int(ratio * size[0])
            im = cv2.resize(im, (resize_width, RESIZE[1]))
        else:
            im = cv2.resize(im, (RESIZE[0], RESIZE[1]))
        im = im.swapaxes(0, 1)
        return (im[:, :, np.newaxis] if IMAGE_CHANNEL == 1 else im[:, :]) / 255.
    return [load_image(index) for index in [img_bytes]]
def decode_maps(charset):
    return {index: char for index, char in enumerate(charset, 0)}
def predict_func(image_batch, _sess, dense_decoded, op_input):
    dense_decoded_code = _sess.run(dense_decoded, feed_dict={
        op_input: image_batch,
    })
    decoded_expression = []
    for item in dense_decoded_code:
        expression = ''
        for char_index in item:
            if char_index == -1:
                expression += ''
            else:
                expression += decode_maps(GEN_CHAR_SET)[char_index]
        decoded_expression.append(expression)
    return ''.join(decoded_expression) if len(decoded_expression) > 1 else decoded_expression[0]
if WARP_CTC:
    import_module('warpctc_tensorflow')
graph = tf.Graph()
tf_checkpoint = tf.train.latest_checkpoint(MODEL_PATH)
sess = tf.Session(
    graph=graph,
    config=tf.ConfigProto(
        # allow_soft_placement=True,
        # log_device_placement=True,
        gpu_options=tf.GPUOptions(
            allocator_type='BFC',
            # allow_growth=True,  # it will cause fragmentation.
            per_process_gpu_memory_fraction=0.01
        ))
)
graph_def = graph.as_graph_def()
with graph.as_default():
    sess.run(tf.global_variables_initializer())
    # with tf.gfile.GFile(COMPILE_MODEL_PATH.replace('.pb', '_{}.pb'.format(int(0.95 * 10000))), "rb") as f:
    #     graph_def_file = f.read()
    # graph_def.ParseFromString(graph_def_file)
    # print('{}.meta'.format(tf_checkpoint))
    model = GraphOCR(
        RunMode.Predict,
        NETWORK_MAP[NEU_CNN],
        NETWORK_MAP[NEU_RECURRENT]
    )
    model.build_graph()
    saver = tf.train.Saver(tf.global_variables())
    saver.restore(sess, tf.train.latest_checkpoint(MODEL_PATH))
    _ = tf.import_graph_def(graph_def, name="")
dense_decoded_op = sess.graph.get_tensor_by_name("dense_decoded:0")
x_op = sess.graph.get_tensor_by_name('input:0')
sess.graph.finalize()
def predict_img(img_bytes):
    batch = get_image_batch(img_bytes)
    return predict_func(
        batch,
        sess,
        dense_decoded_op,
        x_op,
    )
```

然后重新生成一个图片进行测试:

```
from code import gen_exp_pic
from predict import predict_img
from PIL import Image
import io
def image_to_byte_array(image:Image):
  imgByteArr = io.BytesIO()
  image.save(imgByteArr, format="jpeg")
  imgByteArr = imgByteArr.getvalue()
  return imgByteArr
r = gen_exp_pic()
# (<PIL.Image.Image image mode=RGB size=400x32 at 0x7F49A37E02B0>, '843+479*161', 77962)
img = image_to_byte_array(r[0])
predict_img(img)
# '843+479*161'
```

可以看到识别结果还是比较准确的。

计算表达式并提交

使用代码获取验证码进行识别，并提交计算结果，获取flag,代码如下：

```
#!/usr/bin/env python
# coding=UTF-8
import re
import time
import hashlib
import base64
import json
import requests
from predict import predict_img
# 代理设置
proxy = 'http://127.0.0.1:8080'
use_proxy = False
MY_PROXY = None
if use_proxy:
    MY_PROXY = {
        # 本地代理，用于测试，如果不需要代理可以注释掉
        'http': proxy,
        'https': proxy,
    }
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36",
    'Upgrade-Insecure-Requests': '1',
    'Accept-Encoding': 'gzip, deflate',
    'Accept-Language': 'en,ja;q=0.9,zh-HK;q=0.8',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
}
def md5(data):
    md5 = hashlib.md5(data.encode('utf-8'))
    return md5.hexdigest()
def http_req(url, data=None, method='GET', params=None, json=False, cookies=None, proxies=MY_PROXY):
    if json:
        method = 'POST'
        json = data
        data = None
    if method == 'GET':
        params = data
        data = None
    r = requests.request(method, url, headers=headers, verify=False, json=json,
                         params=params, data=data, cookies=cookies, proxies=MY_PROXY)
    return r
def calc_req(url, data=None):
    global my_cookie
    result = http_req(url, data=data, cookies=my_cookie)
    my_cookie = result.cookies
    return result
calc_url = "http://127.0.0.1:8800/"
calc_pic = calc_url + "imgcode"
calc_check = calc_url + "checkexp"
def print_round(txt):
    round_txt = re.search("round.*", txt)
    if round_txt:
        print(round_txt[0])
my_cookie = {
}
r = calc_req(calc_url)
print_round(r.text)
# 由于10次图片识别不一定每次都正确，采用循环直到发现flag
while True:
    pic = calc_req(calc_pic)
    exp = predict_img(pic.content)
    result = eval(exp)
    time.sleep(0.3)
    r2 = calc_req(calc_check, {'result': result})
    print_round(r2.text)
    if len(r2.history) == 0:  # 没有302重定向，则输出结果
        print(r2.text)
        break
```

结果如下，有可能输出的round不同，因为有时验证码会识别错误，重新开始计算round:

```
round: 1 / 10
round: 2 / 10
round: 3 / 10
round: 4 / 10
round: 5 / 10
round: 6 / 10
round: 7 / 10
round: 8 / 10
round: 9 / 10
round: 10 / 10
this is what you want: flag{9cd6b8af2cad231c1125a2c7ce8f3681}
```

Flag

```
flag{9cd6b8af2cad231c1125a2c7ce8f3681}
```


 # Challenge # Misc # 2019 # UNCTF
贝斯的图
长安十二时辰