AlphaJump - 如何用机器学习去玩微信小游戏跳一跳(二)

wstart / 2018-01-15 09:31:00 / 浏览数 4777

前言

上一篇文章，简单介绍了一下Python框架下如何去玩跳一跳 AlphaJump - 如何用机器学习去玩微信小游戏跳一跳(一)。

主要通过判断下一跳的位置和棋子的距离来计算屏幕按压时间，使得棋子可以精准的抵达下一跳的位置。

不同屏幕对应的系数并不一样，这里给出1080*1920下的参数和公式

- 单位 200 ms
- 系数 1.35
- 按压时间 = （棋子与下一跳的距离）x 1.35 x 200

这一篇主要介绍如何使用Tensorflow的物体识别的API去检测跳一跳中的物体以及搭建自己的识别别框架用于。

准备工作

由于之前不小心搞坏了linux环境以及最近特别多的漏洞需要各种研究分析一直没机会处理，最近终于用闲暇时间在window上搭建起来。赶紧把文补上。先来上一下基础环境

- 物理环境
  - CPU E3-1231
  - GTX 970 显存 4G
  - 内存 16G
- 软件环境
  - win7 64
  - 388.71 Nvida驱动
  - cuda_8.0.61_windows.exe
  - python 3.5
  - tensorflow GPU 1.4.0

在你准备好python3.5 以及 tensorflow 后。就可以开始安装object_detection
了。由于涉及的类和库非常的多，分离起来十分困难，直接下载tensorflow下的models。以后学习研究也非常方便。

安装配置 object_detection

官方的安装说明 installation

也可以按照我的来

下载 models https://github.com/tensorflow/models

- 保存为 models/
- 注：我们需要的API在models/research/object_detection 目录下

根据系统类型 下载 Protobuf 下载地址

- 把protobuf的目录添加环境变量
- 测试 shell>protoc
  Missing input file.

- pip 安装其余的库
  - sudo pip install pillow
  - sudo pip install lxml
  - sudo pip install jupyter
  - sudo pip install matplotlib
- protoc 转换协议
  在 models/research/ 目录下执行
  protoc object_detection/protos/*.proto --python_out=.

  添加PYTHONPATH环境变量

  - PYTHONPATH=models/research/slim

  测试

```
python models/research/object_detection/builders/model_builder_test.py
```

此时如果提示 ImportError: No module named 'object_detection' 在该文件头部添加以下三行

-

```
import sys
sys.path.append("E:/models/research/object_detection")
sys.path.append("E:/models/research")
```

- E:/models 改成 你放models的目录

- 输出

```
Ran 11 tests in 0.047s
 OK
```

测试通过 此时准备工作基本完整完成，接下来要搭建我们识别程序的框架。

搭建识别框架

框架结构

我在这篇文章中说过 使用TensorFlow自动识别验证码（一） 深度学习的基本思路是

- 采样
- 创建识别模型
- 生成训练样本和测试样本
- 训练样本和测试样本来训练识别模型
- 保存和验证

我们框架也是依据此来建立目录

- objectdecting_wechatjump 框架目录
  - imgandxml
    - 包含原图库 : 初始化的屏幕截图 [根据AlphaJump - 如何用机器学习去玩微信小游戏跳一跳(一) 获得大量的截图]
    - 采样数据 （ xml格式 ）
  - record
    - 测试数据集 和 训练数据集
  - modle
    - result 最终输出的模型
    - train 训练中的模型
    - train_set 识别模型以及模型训练设置
  - test
    - 验证图片

开始训练

采样数据

为了简单，我这里把图片的物体分为两种

- movtarget ：移动的棋子
- jumptarget ：可以跳跃平台面
  - 这里平台面就是目标位置的顶部部分

但 tensorflow 并不能直接识别图片，需要一种叫tfrecord格式的才可以进入模型训练。

在目录 models\research\object_detection\dataset_tools 下提供了多种的格式转换工具。

这里我选择的 datitran 使用的采样方式 。

- 使用labelimg给图片打标签生成xml
- 通过xml_to_csv的脚本转为 pascal voc的格式
- 再稍微修改dataset_tools下的create_pascal_tf_record.py 即可转为 tensorflow可以识别的record格式
- labelimg的使用非常简单
  - 下载回来后 打开->选择图片文件夹->一个个画框打标签
  - 打完标签后会在对应的目录下生成一个xml文件

我这里一共手工标注了250张的图片。 把这些图片和XML分成两份，一份34，一份216 分别放到

- 216 用于训练模型 放到 objectdecting_wechatjump\imgandxml\train
- 34 用于训练中的检测 放到 objectdecting_wechatjump\imgandxml\eval

把 在 datitran 下载回来的 xml_to_csv.py 文件放到 objectdecting_wechatjump 目录下

- 把 objectdecting_wechatjump\imgandxml\train 和 eval 目录 传入到 xml_to_csv.py 中的函数xml_to_csv 调用 即可转换完成
- 把保存文件名分别命名为 train.csv 和 eval.csv
- 放到 objectdecting_wechatjump\record\ 目录下

把 models\research\object_detection\dataset_tools 下的 create_pascal_tf_record.py 稍作修改

- 把所有的地址参数全部使用FLAGS去传递 方便适应我们的识别模型目录
  - csv_input CSV文件地址
  - output_path 输出record文件地址
  - img_path 原图地址
- 把 def class_text_to_int(row_label): 函数里面识别物体种类改为刚才打标签的种类
  - row_label == 'movtarget': return 1
  - row_label == 'jumptarget': return 2
  - 其他返回 None
  - 可以用于检测是否有异常字段
- 重新命名为 generate_tfrecord.py
- 执行 python generate_tfrecord.py
  - 依次输入 csv_input,img_path
  - output_path 的路径保存在 objectdecting_wechatjump\record 下
- 成功后 可以看到 record中已经保存了tensorflow可以识别的数据

至此 我们的采样数据完成了，接下来是创建一个识别模型

创建识别模型

这里我们不需要自己创建识别模型，因为我们用的就是tensorflow基于COCO数据集提供的几种识别模型 下载地址

- ssd_mobilenet_v1_coco
- ssd_inception_v2_coco
- faster_rcnn_inception_v2_coco
- faster_rcnn_resnet50_coco
- faster_rcnn_resnet50_lowproposals_coco
- rfcn_resnet101_coco
- faster_rcnn_resnet101_coco
- faster_rcnn_resnet101_lowproposals_coco
- faster_rcnn_inception_resnet_v2_atrous_coco
- faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco
- faster_rcnn_nas
- faster_rcnn_nas_lowproposals_coco

这里我们选择最快的 ssd_mobilenet_v1_coco
下载回来后解压,文件列表如下

```
saved_model[dir]
checkpoint
frozen_inference_graph.pb
model.ckpt.data-00000-of-00001
model.ckpt.index
model.ckpt.meta
```

这是一个识别模型最终的输出结果。我们最后训练也会导出同样的文件结构。
其中frozen_inference_graph.pb就是训练结果，里面已经包含了多种的识别。

`model.ckpt` 前缀的文件就是训练模型，我们把

- model.ckpt.data-00000-of-00001
- model.ckpt.index
- model.ckpt.meta

三个文件放到 objectdecting_wechatjump\modle\train_set目录下作为我们训练的初始模型。

- 有网站说 model.ckpt.data-00000-of-00001 要改为 model.ckpt。 这里不需要改。新版的object_detection可以直接读，改了反而出错

在train_set 新建 object-detection.pbtxt
仿照 models\research\object_detection\data 下的pbtxt格式
写上刚才的物体标签

```
item {
 id: 1
 name: 'movtarget'
}

item {
 id: 2
 name: 'jumptarget'
}
```

保存为 object-detection.pbtxt 也放到 objectdecting_wechatjump\modle\train_set 目录下。

到models\research\object_detection\samples\configs 目录下 把训练配置参数配置文件 ssd_mobilenet_v1_pets.config 复制到 objectdecting_wechatjump\modle\train_set 下

- 如果你不是选 ssd_mobilenet_v1_coco 训练模型 那得复制对应的 训练参数文件

修改以下参数

- num_classes: 37 -> 2 多少个训练物体就有多少个
- train_input_reader
  - input_path :train_record路径
  - label_map_path pbtxt文件路径
- eval_input_reader
  - input_path
  - label_map_path pbtxt文件路径
  - num_readers: 3 有多少个测试参数就写多少个 本来要写34 但是为了稳妥 改成3.慢慢加

本来此处，我们的训练模型已经配置完成，可以进入下一步的训练阶段。
但实际上，开始训练的时候往往训练一阵子就内存耗光直接奔溃了程序，无法继续。
查阅了好多资料和源码后，最终发现配置项中需要添加和修改如下参数：
在 train_config: 中

- 添加 batch_queue_capacity: 2
- 添加 prefetch_queue_capacity: 2
- 修改 batch_size的大小
- batch_queue_capacity，prefetch_queue_capacity可以慢慢增加
- batch_size 原本是24，在我的祖传970的4G显存上只能写12个
- 这可能是由于我训练的图片是1080*1920的
- SSD原本的训练集好像最大 600*600 。我记得某一篇文档上说过，而且他们训练使用Titan V

至此，辛苦的活都做完了。接下来就是训练模型阶段。

训练模型

我们先来看一下当前目录的情况

我们的识别框架基本成型，训练模型非常简单，直接调用原生的训练文件train.py即可

执行 python models/research/object_detection/train.py

- 几乎所有参数 需要用--使用，例如 --logtostderr
- 参数
  - logtostderr 为空即可
  - pipeline_config_path : 训练配置文件
    - objectdecting/modle/train_set/ssd_mobilenet_v1_pets.config
  - train_dir=objectdecting/modle/train 训练目录

开始后会显示步数和lost信息

使用 tensorboard --logdir="objectdecting_wechatjump/modle/train" 可以看到界面的训练情况

在目录 objectdecting/modle/train 下 我们已经可以看到模型生成了

- 一开始的数字是 model.ckpt-0000
- 训练可以随时停止
- 重新开始的话 tensorflow会读取最新的训练模型开始
- 每隔一段时间会记录一个训练模型，model.ckpt-XXXX 不同就是不同的模型。
- 我们最终会根据某一个训练模型 model.ckpt-XXXX 导出我们的模型

在我的970训练了1个多小时后，步数抵达到了24425，来试试导出模型 测试一下

导出模型和验证

把 models\research\object_detection\sexport_inference_graph.py的导出文件 拷贝到
objectdecting_wechatjump目录下 。
执行 python export_inference_graph.py
参数

- input_type:image_tensor
- pipeline_config_path 训练模型参数文件
  - objectdecting_wechatjump/modle/train_set/ssd_mobilenet_v1_pets.config
- trained_checkpoint_prefix 训练中的模型 这里选一个最大数字就行 :
- objectdecting_wechatjump\modle\train\model.ckpt-24425"
- output_directory 模型导出目录
  - objectdecting_wechatjump\modle\result

导出完成后，如图，和我们下载的ssd_mobilenet_v1的模型一样的结构

接下来我们写一下验证文档，总体思路是

- 读取模型
- 读取设置文件
- 读取图片
- 验证

保存为checkmodel.py 。 具体我就不贴代码了 ，稍后会在我的github上同步。github.com/wstart

测试结果如下

基本上完整识别了所有的目标。只需要取Y轴最高的jumptarget和movtarget的中心点，然后通过时间计算系数，就可以完美的跳了。
最高纪录好像22000，后来就停了，不过微信已经把我加入黑名单了，没办法更新分数。

总结

使用tensorflow物体识别可以快速定位移动物体。但是不是最后的一步，定位到物体后还需要检测一下物体是否标准，再去确定中心点的位置。否则很容易因为识别误差

要获取大量的图样，前期需要采集足够多的样本,训练的进度才高。除了一开始的基础框架要跳的足够远以外，起码要500以上，才会出现小圆点，还可以通过破解小程序

tensorflow使用的属于 RCNN 这种算法还是源于CNN。原理可以参考 使用TensorFlow自动识别验证码（三）--- CNN模型的基础知识概述以及模型优化

最新的物体检测算法是YOLO（You only look once），号称实时物体检测，等待测试。环境部署十分简单。 等下一次小游戏上线再测试。

后续代码 测试图 模型等会更新到我的github上， wechat_AlphaJump

上一篇：Misc 总结 ----流量分析 ... 下一篇：Apache FOP-XXE—【C...
1. 1 条回复

tb03****82_11 2018-01-24 10:28:39

巨强

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板