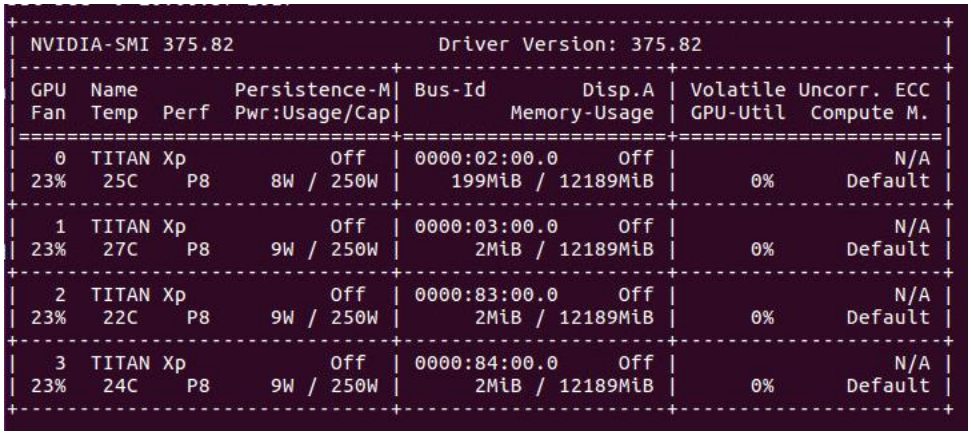
若对以下描述有任何疑问，请及时与我们联系。

邮箱: jiexin_zheng@qq.com
手机: 15918968665　郑杰鑫

1. 运行环境

Ubuntu 16.04　python 2.7.12　cuda8.0　cudnn6.0　tensorflow 1.3.0

GPU 4*TITAN XP



2. 从视频中截取出猪：

(1)为了排除背景数据对模型的影响，我们使用 yolo-9000 算法提取出视频中每一帧的猪，代码来源于 https://github.com/philipperemy/yolo-9000.

我们对其代码做了修改，将 yolo 解压包的代码解压后覆盖 darknet/src 下同名文件即可

(2)经观察后发现，虽然 yolo-9000 对猪的识别不一定会归于 hog 类，但是基本上所有的框都会以视频中的猪为主体，因此在取框的时候，我们不以 hog 类的框为输出图像，而是以置信度为参考标准。

(3)我们保留所有置信度大于 0.1 的窗口

(4)每个视频大约能得到一万多张 ROI 图片，我们按大小排序，选取大约前 4000 张图片，并剔除不相关的物体图片以及背景干扰较大的图片（比如没有框到猪身上，或者只框了极小部分的猪），将其作为训练集和验证集。

(5)最后得到 94677 张图片

3. 预处理以及生成数据集

(1)运行 raw_data/image_process.py，　将上一步得到的图片通过 padding 的方法变为正方形，保证在之后的步骤中 resize 操作不会扭曲图片

(2)运行 raw_data/get_data_txt.py，对数据进行分割，并且将数据分割成 50 个储存文件，存在 txt 文件中，方便之后大数据的分步读取

(3)运行 raw_data/create_h5_dataset.h5，将数据生成 h5 文件，这一步之后会得到 50 个储存训练集的.h5 文件，以及 50 个储存验证集.h5 文件

注意!!!!!!

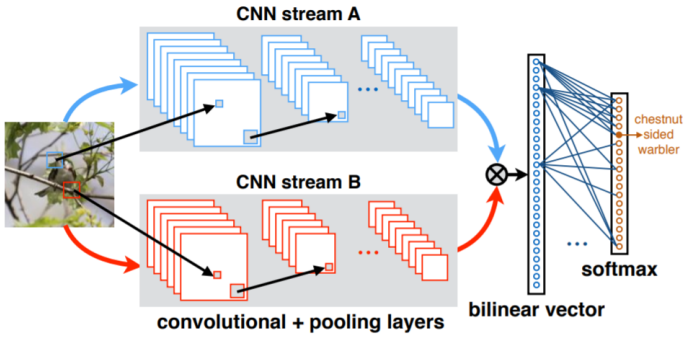
算法具有原创性，赛后计划继续研究发表论文。关于算法的细节请勿透漏给任何与比赛无关的人员

4. 模型

(1)本模型基于细粒度识别模型 bilinear cnn 做的改进，参考源码来自于 https://github.com/abhaydoke09/Bilinear-CNN-TensorFlow

参考论文 vis-www.cs.umass.edu/bcnn/docs/bcnn_iccv15.pdf

Bilinear cnn 是一个端到端的网络模型，该模型在 CUB200-2011 数据集上取得了弱监督细粒度分类模型的最好分类准确度。 模型如下：



(2)bilinear cnn 把最后一层卷积核的输出做了外积（实际是做内积），以此达到融合不同特征的目的。

(3)我们队伍受 resnet 结构的启发，对 bilinear cnn 算法做了改进，将最后一层卷积核的输出也和前面其他层的卷积核的输出做内积，以此达到融合不同层次的特征的目的。再把得到的 vector 和原来的 bilinear vector 融合。 我们增加了 conv4_1、conv5_1 对 conv5_3 的内积（只增加这两层是因为他们的 filter numbers 数量一致，pooling 之后就可以做内积了，不需要加额外的卷积核）

我们的思想是：不同卷积层关注的特征不同，且对应感受视野的大小也不同（即有高低层次之分），在识别类似图像时，单独考虑特征是不够的，还需要考虑他们之间的空间关系。

(4)加载预训练的 vgg 模型，先训练全连接层，之后再训练整个网络。预训练权重下载地址 https://www.cs.toronto.edu/~frossard/post/vgg16/

(5)训练过程中加入实时的数据增强，包括旋转、随机改变对比度、随机改变亮度、随机 crop. 训练时全连接层的 drop out 概率为 0.5

4. 结构

(1)train/read_data.py 是读取数据的结构。实现大数据的分次加载。

(2)train/resvgg_model.py 定义了网络结构，以及读取保存的权重的方法

(3)train/train_resvgg.py 定义了训练的过程

(4)train/predict_resvgg.py 输出预测结果

5. 加载预训练模型，微调

(1)在读取 resvgg 模型时， 令 finetune=False,实现只训练最后的全连接层。并且调用 load_initial_weights(sess)，读取预训练的 vgg 的卷积层的参数

(2)训练设置 optimizer = tf.train.MomentumOptimizer(learning_rate=0.2, momentum=0.5).minimize(loss)， 训练次数 50 次

(3)将过程中得到的最优模型保存下来

6. 全网络训练

(1)在读取 resvgg 模型时，令 finetune=True。 调用 load_own_weight(sess , model_path)，读取上一步得到的模型

(2)训练设置 optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)， 训练 200 次

(3)将过程中得到的最优模型保存下来

7. 后期调整

实际训练过程中，只有第一次会在所有数据上训练满 200 次。在得到保存下来的模型后，之后的调参过程只取大约 1/4 的数据进行继续训练

8. 预测

(1)运行 predict_resvgg.py 预测结果