Kaggle-Driver_state_detection

<Excerpt in index | 首页摘要>
使用Tensorflow搭建kaggle比赛，主要目的熟悉TensorFlow的各种常规操作
1、赛题介绍　　2、比赛思路　　3、优化改进　　4、代码
Github：Tensorflow
Resource:深度学习理论与实战（基于TensorFlow实现）

<The rest of contents | 余下全文>

比赛介绍

kaggle

In this competition you are given driver images, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc). **Your goal is to predict the likelihood of what the driver is doing in each picture.**

The 10 classes to predict are:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

读取数据

1、使用`tf.data`导入数据

数据集对象实例化：

```
dataset = tf.data.Dataset.from_tensor_slices(数据)
```

代码如下

```
dataset = tf.data.Dataset.from_tensor_slices(np.array([1.0, 2.0, 3.0,
4.0, 5.0]))
iterator = dataset.make_one_shot_iterator()
one_element = iterator.get_next()
with tf.Session(config=config) as sess:
    try:
        while True:
            print(sess.run(one_element))
    except tf.errors.OutOfRangeError:
        print("end!")
```

读入磁盘图片与对应label

考虑一个简单，但同时也非常常用的例子：读入磁盘中的图片和图片相应的label，并将其打乱，组成**batch_size=32**的训练样本，在训练时重复**10个epoch**

在这个过程中，dataset经历三次转变：

- 运行dataset = tf.data.Dataset.from_tensor_slices((filenames, labels))后，dataset的一个元素是(filename, label)。**filename是图片的文件名，label是图片对应的标签。**
- 之后通过**map**，将filename对应的图片读入，并缩放为28x28的大小。此时dataset中的一个元素是(image_resized, label)

- 最后，dataset.shuffle(buffersize=1000).batch(32).repeat(10)的功能是：**在每个epoch内将图片打乱组成大小为32的batch，并重 复10次。**最终，dataset中的一个元素是(image_resized_batch, label_batch)，image_resized_batch的形状为(32, 28, 28, 3)，而label_batch的形状为(32,)，接下来我们就可以用这**两个Tensor来建立模型**了。

TensorFlow tf.data 导入数据

blog

官方文档

- `tf.data.Dataset`：表示一系列元素，其中每个元素包含一个或多个 `Tensor 对象`。
- `tf.data.Iterator`：这是从`数据集中提取元素`的主要方法。

使用`TensorFlow tf.data`导入数据的基本机理:

- 1、使用 `tf.data.Dataset.from_tensors()` 或`tf.data.Dataset.from_tensor_slices()`构建**Dataset**
- 2、通过`tf.data.Dataset`对象上的方法调用将其转换为新的 **Dataset**,例如`Dataset.map()`（为每个元素应用一个函数），也可以应用多元素转换（例如`Dataset.batch()`）
- 3、构建**迭代器**，可以一次访问数据集中的一个元素`Dataset.make_one_shot_iterator()`。`tf.data.Iterator`提供了两个操作：`Iterator.initializer`，您可以通过此操作（重新）初始化迭代器的状态；以及`Iterator.get_next()`，此操作返回对应于有符号下一个元素的 tf.Tensor 对象。

```
def read(names, labels, batch_size=None, num_epoch=None, shuffle=False, p
hase='train'):
    def _read_img(name):
        #TODO
        # 给定图像名称tensor, 输出3维浮点值图像
        content = tf.read_file(name)
        image = tf.image.decode_image(content, channels=3)
        image.set_shape((None, None, 3))
        image = tf.cast(image, dtype=tf.float32)
        return image
    def _train_preprocess(img):
        #TODO
        # 对训练集图像预处理
        # 例如resize到固定大小,翻转,调整对比度等等
        img_resized = tf.image.resize_images(img, (256, 256))
        img_normed = tf.image.per_image_standardization(img_resized)
        return img_normed
    def _eval_preprocess(img):
        #TODO
        # 对验证集, 测试集图像预处理
        # 例如resize到固定大小等等
        img_resized = tf.image.resize_images(img, (256, 256))
        img_normed = tf.image.per_image_standardization(img_resized)
        return img_normed
```

```
    #TODO
    # 构造图像名称 dataset
    name_dataset = tf.data.Dataset.from_tensor_slices(names)
    #TODO
    # 通过 map 函数调用 _read_img 构造图像 dataset
    image_dataset = name_dataset.map(_read_img)
    if phase == 'train':
        #TODO
        # 通过 map 函数对训练集图像进行处理
        image_dataset = image_dataset.map(_train_preprocess)
    else:
        #TODO
        # 通过 map 函数对验证集,测试集图像进行处理
        image_dataset = image_dataset.map(_eval_preprocess)
    #TODO
    # 构造图像标签 dataset
    label_dataset = tf.data.Dataset.from_tensor_slices(labels)
    #TODO
    # 将图像以及图像标签 dataset 通过 zip 合并成一个 dataset
    dataset = tf.data.Dataset.zip((image_dataset, label_dataset))
    #TODO
    # 设置 dataset 的 epoch
    dataset = dataset.repeat(num_epoch)
    #TODO
    # 在需要 shuffle 时, 对 dataset 进行 shuffle
    if shuffle:
        dataset = dataset.shuffle(buffer_size=1000)
    #TODO
    # 在需要进行 batch 时, 对 dataset 进行 batch
    if batch_size is not None:
        dataset = dataset.batch(batch_size)      #顺序很重要
    #TODO
    # 构造 dataset 的迭代器
    iterator = dataset.make_one_shot_iterator()
    #TODO
    # 获取 dataset 的元素
    image, label = iterator.get_next()
    return image, label
```

`Dataset.shuffle()`转换会使用类似于 `tf.RandomShuffleQueue`的算法随机重排输入数据集：它会维持一个固定大小的缓冲区，并从该缓冲区统一地随机选择下一个元素。

所以shuffle的设置应该在batch的前面·

2、划分训练集和验证集

```
from sklearn.model_selection import  train_test_split
train_names,valid_names,train_labels,valid_labels = train_test_split(imag
e_names,image_labels,test_size = 0.2,random_state=42)
```

```
NUM_EXAMPLES_OF_TRAIN = len(train_names)
NUM_EXAMPLES_OF_VALID = len(image_names) - NUM_EXAMPLES_OF_TRAIN
```

加载预训练模型

TensorFlow预训练模型下载

1、找到所有需要 finetune 的变量

```
    if pretrained_model is not None:
        #TODO
        # 找到所有需要 finetune 的变量
        vars_to_finetune = {}

        for var in tf.model_variables():   # Returns all variables in the
 MODEL_VARIABLES collection.
            if 'logit' not in var.op.name:
                var_name_in_ckpt = var.op.name.replace('model', 'resnet_v
2_50')   # 将checkpoint文件中的变量名映射到图中的每个变量的字典
                vars_to_finetune[var_name_in_ckpt] = var

        vars_to_init = filter(lambda var: var not in vars_to_finetune.val
ues(), tf.global_variables())

        #TODO
        # 生成一个对上面变量的加载器
        restorer = tf.train.Saver(vars_to_finetune)
```

2、加载预训练模型

```
if pretrained_model is not None:
    #TODO
    # 使用加载器恢复变量的数值,并初始化其他变量
    sess.run(tf.variables_initializer(vars_to_init))
    restorer.restore(sess, pretrained_model)
else:
    #TODO
    # 初始化所有变量
    sess.run(tf.global_variables_initializer())
```

保存和恢复模型

【tensorflow】保存模型、再次加载模型等操作

1、保存模型

```
# 首先定义saver类
saver = tf.train.Saver(max_to_keep=4)
# 定义会话
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range(300):
        if epoch % 10 == 0:
            print "----------------------------------------------------------
"
            # 保存模型
            saver.save(sess, "model/my-model", global_step=epoch)
            print "save the model"
        # 训练
        sess.run(train_step)
```

- 创建**saver**时，可以指定需要存储的**tensor**，如果没有指定，则全部保存
- 创建saver时，可以指定保存的**模型个数**，利用max_to_keep=4，则最终会保存4个模型（下图中我保存了160、170、180、190step共4个模型）。
- `saver.save()`函数里面可以设定`global_step`，说明是哪一步保存的模型。
- 程序结束后，会生成四个文件：存储网络结构`.meta`、存储训练好的参数`.data`和`.index`、记录最新的模型`checkpoint`。

2、模型文件

Tensorflow模型保存后有四个文件:

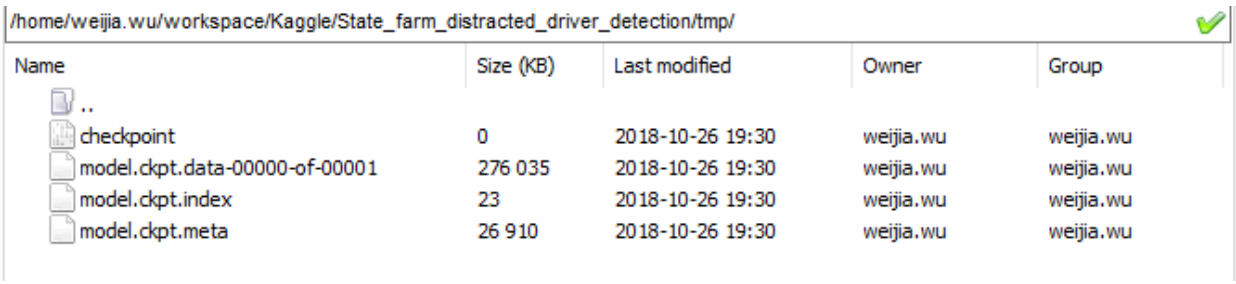
a) Meta graph：

这是一个协议缓冲区，它保存了完整的**Tensorflow图形**;即所有变量、操作、集合等。该文件以.meta作为扩展名。

b) Checkpoint file：

这是一个二进制文件，它包含了所有的**权重、偏差、梯度和其他所有变量的值。**这个文件有一个扩展名.ckpt。然而，Tensorflow从0.11版本中改变了这一点。现在，我们有**两个文件，而不是单个.ckpt文件**:

`存储网络结构.meta、存储训练好的参数.data和.index、记录最新的模型checkpoint。`



3、加载模型

```
def load_model():
    with tf.Session() as sess:
        saver = tf.train.import_meta_graph('model/my-model-290.meta')
        saver.restore(sess, tf.train.latest_checkpoint("model/"))
```

- 首先`import_meta_graph`，这里填的名字`meta`文件的名字。
- 然后`restore`时，是检查`checkpoint`，所以只填到checkpoint所在的路径下即可，不需要填checkpoint，不然会报错`“ValueError: Can’t load save_path when it is None.”`。

可视化

1、grad-CAM

具体原理及实现参考blog

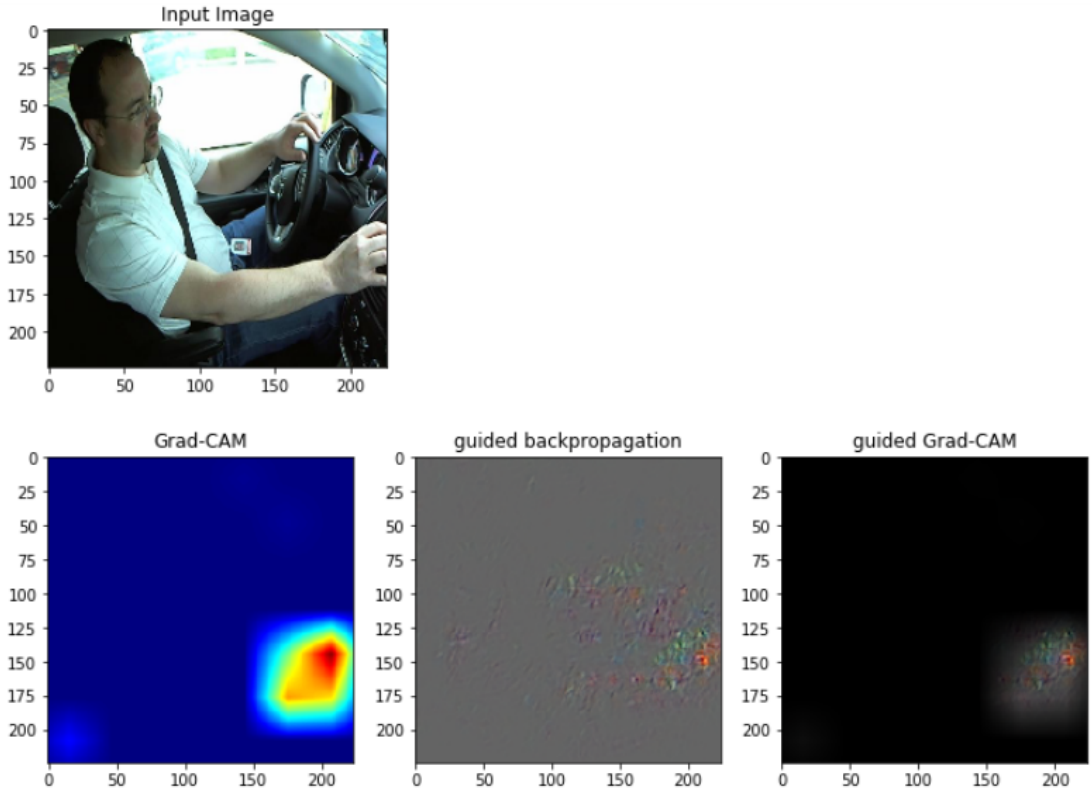
```
cost = (-1) * tf.reduce_sum(tf.multiply(labels, tf.log(prob)), axis=1)

# Get last convolutional layer gradient for generating gradCAM visualizat
ion
target_conv_layer = end_points['model/block4/unit_3/bottleneck_v2/conv3']

# gradient for partial linearization. We only care about target visualiza
tion class.
y_c = tf.reduce_sum(tf.multiply(net, labels), axis=1)
target_conv_layer_grad = tf.gradients(y_c, target_conv_layer)[0]

# Guided backpropagtion back to input layer
gb_grad = tf.gradients(cost, images)[0]
print('gb_grad:', gb_grad)
```

2、结果



模型提升

1、Evaluation

常见的模型评价指标：

- **1.对数损失函数(Log-loss)**

对数损失, 即对数似然损失(`Log-likelihood Loss`), 也称逻辑斯谛回归损失(`Logistic Loss`)或交叉熵损失(`cross-entropy Loss`), 是在概率估计上定义的.

对数损失通过**惩罚错误的分类**,实现对分类器的准确度(`Accuracy`)的量化. 最小化对数损失基本等价于最大化分类器的准确度.为了计算对数损失, **分类器必须提供对输入的所属的每个类别的概率值, 不只是最可能的类别**. 对数损失函数的计算公式如下:

$$L(Y,P(Y|X))=-\log P(Y|X)=-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}y_{ij}\log(p_{ij})$$

其中, `Y` 为**输出变量**, `X` 为**输入变量**, `L` 为**损失函数**. `N` 为输**入样本量**, `M` 为**可能的类别数**, `yij` 是一个**二值指标**, 表示类别 `j` 是否是输入实例 `xi` 的真实类别. `pij` 为模型或分类器预测输入实例 `xi` 属于**类别** `j` 的**概率**.

- **2.精确率-召回率(Precision-Recall)**
- **3.AUC(Area under the Curve(Receiver Operating Characteristic, ROC))**

AUC和logloss比accuracy更常用

2、数据增强

有时间来完成

3、模型融合

有时间来完成