机器学习毕业项目开题报告

题目: 猫狗大战项目
作者: Kyle Chen
版本: 20180507v1
日期: 20180507

项目背景

- 项目涉及的相关研究领域
- 在猫狗大战项目研究中, 重点研究了深度学习在图像识别中的应用. 猫狗大战是一个典型的二分类应用场景, 主要用于将图片中的猫, 狗区分出来. 在此项目中, 输入是一张相片, 相片中, 可以是任何猫或狗, 当其中出现猫时, 期望预测结果为猫. 如若为狗时, 期望预测结果为狗.
- 在现实生活中, 不乏很多多分类问题, 但是我们只有在对二分类问题非常了解的情况下, 才能对多分类问题有更深的理解, 也对往后处理分类问题落地起到了至关重要的作用.

问题描述

- 解决办法所针对的具体问题
- 在此项目中, 我们需要解决针对图像的训练与分类问题, 这是一个有监督学习的二分类问题. 首先, 要先对训练集中的数据进行训练; 在多次训练与学习中, 提升对数据预测的准确度; 其次, 在训练完成之后, 对测试集进行预测与评分.

输入数据

- 问题中涉及的数据或输入是什么
- 在此项目中, 输入应为一张图片. 图片中可以是猫或狗. 当出现狗时, 则期待分类器能将其归类为狗这一类; 否则, 我们期待我们的分类器能将其归类为猫这个类型;
- 在代码实现中, 可以使用keras.preprocessing中的image库加载RGB图像.

- 研究下kaggle给我们提供的样本:

```
➜ dogs-vs-cats-redux-kernels-edition ✗ ls -ahl train/ | grep -i cat |
head -n 3
-rw-r--r--      1 Kyle  staff    12K Sep 20  2013 cat.0.jpg
-rw-r--r--      1 Kyle  staff    16K Sep 20  2013 cat.1.jpg
-rw-r--r--      1 Kyle  staff    34K Sep 20  2013 cat.10.jpg

➜ dogs-vs-cats-redux-kernels-edition ✗ ls -ahl train/ | grep -i dog |
head -n 3
-rw-r--r--      1 Kyle  staff    31K Sep 20  2013 dog.0.jpg
-rw-r--r--      1 Kyle  staff    24K Sep 20  2013 dog.1.jpg
-rw-r--r--      1 Kyle  staff    12K Sep 20  2013 dog.10.jpg
```

不难发现, 样本中的Y, 就是文件的prefix = [cat | dog], 标签和样本是绑定在一起的, 这方便了我们对样本打乱.

- 接着我们统计下训练集中猫,狗的类型分布:

```
➜ dogs-vs-cats-redux-kernels-edition ✗ find train/ -name "cat*" | wc -
l
   12500
➜ dogs-vs-cats-redux-kernels-edition ✗ find train/ -name "dog*" | wc -
l
   12500
```

可以发现, 这里的猫狗是均匀分布的, 我们可以在训练集中取一部分来作为训练集, 另外一部分作为验证集.

- 在获得训练集, 验证集之前, 需要将train/目录下的文件打乱, 然后按照一定的比例将其划分到训练集, 验证集中.

解决办法

- 针对给定问题的解决方案
- 在一切开始前, 我们需要准备我们的数据集. 在猫狗大战这个项目中, 可以直

接从kaggle上下载DataSet. 在安装kaggle api之后, 我们可以直接在终端执行:

```
➜ Dogs_vs_Cats ✗ kaggle competitions download -c dogs-vs-cats-redux-kernels-edition
```

- 接下来, 我们需要将DataSet中的数据分成训练集, 验证集, 测试集. 由于测试集已经被单独存放到test/目录下, 仅需将train/目录下的文件分成训练集与验证集, 可以参考比例7:3来划分.
- 使用keras框架构建深度卷积神经网络, 这里我们使用Xception进行迁移学习训练, 在第一次调用时会自动到github上下载相关的训练好的特征权重模型, 供我们后面训练使用. 在构建模型时, 并不是直接将其加载进来就能直接使用, 我们需要将其嵌入我们需要训练的模型中去. 例如说, 猫狗大战, 是一个二分类问题, 所以我们需要将最终的预测结果修改为两类(猫/狗).
- 在训练前, 我们可以选择冻结/解冻训练模型中卷积层, 来选择我们需要训练哪些卷积层. 这里我们需要尝试多种方式, 可以将最开始的几层冻结, 或者将其全部冻结只训练输出层, 亦或者冻结其中某几层. 这个过程需要大量的测试来确保我们的模型能达到最优的结果.
- 在训练完成后, 我们选取最优的模型, 统计此模型在测试集中的成绩(需要将预测结果上传至kaggle获取评分), 作为最终得分.

基准模型

- 用来与你的解决方案进行比较的一些简单的、过去的模型或者结果
- 在使用Xception训练前, 我们还需要将其与CNN做个比较, 在其不使用迁移学习的模型时, 是否还能有好的表现.
- 这里我们可以使用relu作为Hidden_nodes的激活函数, sigmoid作为输出函数, 在中间添加Conv2D, MaxPooling2D. 如若需要, 可以加入BN防止过拟合. 当然, 在没有实际代码经过多次测试的基础下, 暂且不能确定最优的模型框架.
- 拟合数据时需要加上validation_split = 0.3, shuffle = True两个参数.
- 这里使用LogLoss来为模型评估分数.
- 关于CNN部分可以参照可以下框架搭建:

```
cnn_model = Sequential()
shape_input = (len(data[0]), len(data[0][0]), len(data[0][0][0]))
cnn_model.add(Conv2D(filters=16, kernel_size=2, input_shape=shape_input))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(Dense(133, activation='relu'))
cnn_model.add(Conv2D(filters=32, kernel_size=2))
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(Dense(133, activation='relu'))
cnn_model.add(Conv2D(filters=64, kernel_size=2))
cnn_model.add(MaxPooling2D(pool_size=2, padding='valid'))
cnn_model.add(GlobalAveragePooling2D(dim_ordering='default'))
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.summary()
```

- 以上模型在猫狗大战中的训练过程如下:

```
Train on 17500 samples, validate on 7500 samples
Epoch 1/5
17500/17500 [==============================] - 1908s 109ms/step - loss: 0.6724 - acc: 0.5890 - val_loss: 0.6701 - val
_acc: 0.5653

Epoch 00001: val_loss improved from inf to 0.67014, saving model to saved_models/weights.best.cnn.hdf5
Epoch 2/5
17500/17500 [==============================] - 1749s 100ms/step - loss: 0.6535 - acc: 0.6191 - val_loss: 0.6427 - val
_acc: 0.6325

Epoch 00002: val_loss improved from 0.67014 to 0.64273, saving model to saved_models/weights.best.cnn.hdf5
Epoch 3/5
17500/17500 [==============================] - 1754s 100ms/step - loss: 0.6445 - acc: 0.6271 - val_loss: 0.6435 - val
_acc: 0.6259

Epoch 00003: val_loss did not improve
Epoch 4/5
17500/17500 [==============================] - 1775s 101ms/step - loss: 0.6400 - acc: 0.6304 - val_loss: 0.6318 - val
_acc: 0.6348

Epoch 00004: val_loss improved from 0.64273 to 0.63181, saving model to saved_models/weights.best.cnn.hdf5
Epoch 5/5
17500/17500 [==============================] - 1796s 103ms/step - loss: 0.6375 - acc: 0.6335 - val_loss: 0.6622 - val
_acc: 0.6156

Epoch 00005: val_loss did not improve

<keras.callbacks.History at 0x49f859470>
```

- 可以看到还有很大的优化空间, 我们还可以通过迁移学习来慢慢接近目标, 进入前10%, 也就是LogLoss分数达到0.06127以下.

评估指标

- 衡量你解决方案的标准
- 可以使用模型在测试集中的得分(LogLoss)来对指标评估.
- 在kaggle页面中, 也为我们提供了验证LogLoss函数:

$$LogLoss = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)]$$

$\hat{y}$ 表示我们预测出来的结果, y表示图片的正确归类, n表示样本个数.

- 当然, 很重要的一点, 在最后, 需要进入kaggle猫狗大战挑战中的前10%, LogLoss分数大概在0.06127以下.

设计大纲

- 你的解决方案如何实现， 如何获取结果
- 将猫狗图片分类, 从数据集中分割出训练集, 验证集, 测试集(已单独存放).
- 定义模型框架, 这里包含两个部分, 在一开始, 我们需要对CNN做一个测试(关于这个模型应该怎么搭建, 在基准模型中已经表明), 看看是否可以直接用CNN解决这个二分类问题, 在训练完成, 训练并获得验证集LogLoss分值, 对参数进行持续调优, 看看是否还有优化空间; 其次, 使用Xception搭建新的模型, 将其融入我们的二分类模型中, 训练并获得验证集LogLoss分值, 并对其持续调优, 以获得更低的LogLoss分值.
- 最后选取得分最高, 最好的模型, 作为最终模型.
- 提交最优模型到kaggle, 获取测试集中的LogLoss分值, 检查排名是否符合要求.
- 最后, 在上述基本模型上, 在加载图片时, 可以使用skimage对数据进行增强, 还可以加入模型融合, 类似于boosting, stacking的方法来获得更低的LogLoss分值.

引用

[1] Dogs vs. Cats Redux: Kernels Edition Rules: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/rule
[2] François Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, 4 Apr 2017: https://arxiv.org/abs/1610.02357
[3] Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2 Mar 2015: https://arxiv.org/abs/1502.03167
[4] Nitish Srivastava, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 11/13 2014: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

备注: 本论文无摘抄部分, 纯属原创.