机器学习纳米学位-开题报告

王烨

2018年8月30日

项目背景

猫狗大战（Dog vs Cat）是Kaggle举办的一个娱乐型的竞技比赛。在这个比赛中需要编写计算机算法来实现分类图像中是猫还是狗。Kaggle举办这个比赛的原因是，原来在很多服务器网站 上为了防止机器人注册而使用了基于动物识别的验证码，这个验证是利用人能够很容易的区分猫狗，而机器不能，但是在人工智能发展之后，机器对猫狗的识别率已经大大提高了。本比赛是为了挑战计算机对猫狗的识别率。本项目是使用卷积神经网络（Convolutional Neural Network，CNN），来对图像进行识别。

项目选择的数据集是 Kaggle 竞赛提供的数据，训练集包括 12500 张被标记为猫的图片和 12500 张被标记为狗的图片，测试集包括 12500 张未标记图片。对于每一张测试集中的图 像，模型需要预测出是狗图像的概率（1 代表狗，0 代表猫）。

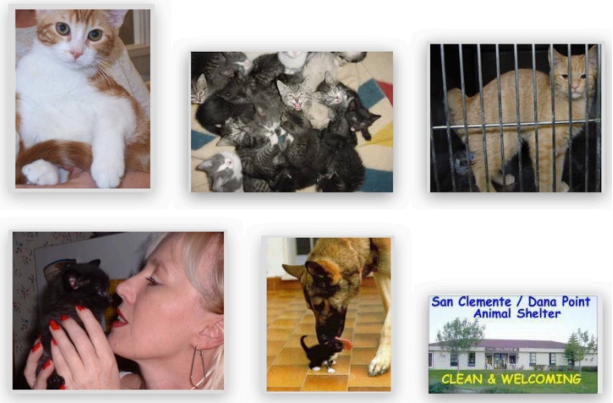
问题描述

1. 首先，这是一个分类问题。
2. 项目提供的数据是来源某收留流浪猫狗网站的真实的照片，背景复杂，分辨率不同，还有一些异常图片混在其中。

数据集分析

项目选择的数据集是 Kaggle 竞赛提供的数据，训练集包括 12500 张被标记为猫的图片和 12500 张被标记为狗的图片，测试集包括 12500 张未标记图片。对于每一张测试集中的图 像，模型需要预测出是狗图像的概率（1 代表狗，0 代表猫）。

首先，简单观察一下训练数据有哪些类型，照片来自真实的场景，可以看到场景丰富。有一只猫，关在笼子后的猫，多只猫，人和猫，猫和狗 同时现出现的；还有少量的异常值。



解决方案

基于前面的数据探索与分析可知，猫狗识别问题其实是一个分类问题。输入一张图片得到这张图片是猫还是狗的概率。分类算法就我们知道的有 决策树、支持向量机、逻辑回归、贝叶斯 、神经网络 等方法。基于经验， 神经网络 尤其适合图像的识别。

基准指标

此次的要求是在Kaggle上取得前10%的排名，截止目前2018/09/06为止，目前有4441名提交的成绩，按得分排序找到第444名，得分如下：

443	388025	Yujie	2017-02-13 14:24:33	0.05602
444	368495	Bojan Tunguz	2016-09-15 12:26:08	0.05603

也就是说这次项目的得分要小于等于 0.05603

评估指标

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

n **是测试集中的图像数量**

$\hat{y}_i$ **是图像是狗的预测概率**

y_i **图像如果是狗为1，如果是猫为 0**

$\log()$ **是以E为底的自然对数**

项目设计

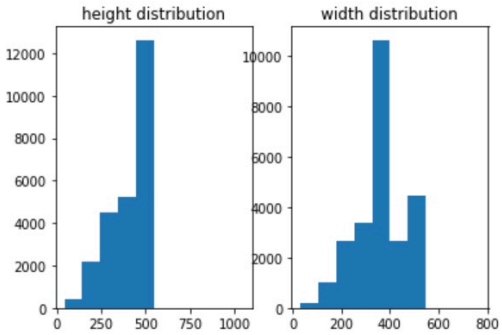
本项目将采用迁移学习，所谓迁移学习，就是将在某一个预训练好的模型通过简单的调整，使其适用于一个新问题。本次作者将采用，Xception和InceptionV3模型。

项目实施步骤

1. 下载数据集 Kaggle Data

2. 数据探索

这些训练数据的尺寸，如下图：



训练集：median of height: 447.0 median of width: 374.0

将要使用的预训练网络模型为Xception和InceptionV3，这两个模型的图片预处理方式为将数据从（0，255）缩放到 （–1，1）区间内。Xception 和 InceptionV3 的默认输入分 辨率为299x299。分别使用： inception_v3.preprocess_input来进行xception.preprocess_input来做归一化操作。

3. 数据预处理

因为图片数据大小不一、场景复杂，所以决定用Keras的ImageDataGenerator函数进行统一的预处理，生成批次的带实时数据增益的张量图像数据。

ImageDataGenerator.flow_from_directory函数要求需要将不同种类的图片分在不同的文件夹中，因此我们需要对数据集进行分类，为了减少磁盘的消耗量，作者决定采用建立软链接的方式将图片分到不到目录中。

```
def sort_img(dir_path):
    '''判断图片是猫还是狗'''
    train_filenames = os.listdir(dir_path)
    train_cat = list(filter(lambda x:x.split('.')[0] == 'cat', train_filenames))
    train_dog = list(filter(lambda x:x.split('.')[0] == 'dog', train_filenames))
    return train_cat , train_dog
```

4. 剔除异常数据

因为数据中有少量的异常数据，可能会在一定程度上干扰训练效果。作者准备将异常数据剔除。作者使用keras提供的预训练网络模型Xception来剔除异常数据。首先从ImageNet上下载

动物的分类文件，并制作成csv文件，然后利用Xception识别出猫与狗，最后与训练集求差集，得到异常数据；最后在人工加入未检测出的异常数据：

```
n02085620,狗
n02085782,狗
n02085936,狗
...
n02125311,猫
n02127052,猫
```

*CSV*文件格式

```
image_net_classes = CUR_PATH + '/ImageNet_animal_Classes.csv'
img_size = (299, 299)
## 载入预先下载好的ImageNet分类csv文件
import csv

def get_imageNet_class(file_path):
    data_list = []
    with open(file_path,'r') as f:
        data_list = list(csv.reader(f))

    return list(n[0] for n in data_list if n[1] in ['猫','狗'])

def preprocess_image(img_path, model_preprocess):
    img = image.load_img(img_path,target_size=img_size)
    x = image.img_to_array(img)
    x = np.expand_dims(x,axis=0)
    x = model_preprocess(x) #归一化

    return (x)

def preprocess_image_dir(dir_list,model_preprocess):

    return [preprocess_image(fp, model_preprocess) for fp in tqdm(dir_list)]

imageNet_class = get_imageNet_class(image_net_classes)
# preprocess_imgs = preprocess_image_dir(train_data[:5000],xception.preprocess_input)
preprocess_imgs = preprocess_image_dir(train_data,xception.preprocess_input)
```

使用预训练模型*Xception*识别函数

```
# 获取能识别为猫狗类型的预测总数
def get_result_list(pred):
    print(type(pred))
    result = []
    for index, item in enumerate(pred):
        result.insert(index,0);
        for value in item:
            if value[0] in imageNet_class:
                result[index] = 1
                break;
    return result
```

对按相同索引对所有训练图片进行标志猫狗图片为*1*，其它为*0*。

```
manual_right_list = ['cat.10712.jpg']  #保留正确的图片
manual_exp_list = ['cat.4085.jpg','dog.1773.jpg',]  #人工增加未识别出的
```

人工增加未识别出的错误图片及识别错误的图片

5. 作者准备利用Xception和InceptionV3预训练网络模型，先分别利用这两个单模型对训练集进行训练，然后再使用混合模型对训练集进行训练并查看效果。具体示例如下：

I. 保存预训练模型特征向量函数

```
    gen = ImageDataGenerator()
    train_generator = gen.flow_from_directory(TRAIN_DIR, image_size, shuffle=False,
                                              batch_size=16)
    test_generator = gen.flow_from_directory(TEST_DIR, image_size, shuffle=False,
                                             batch_size=16, class_mode=None)

    base_model = MODEL(input_tensor=x, weights='imagenet', include_top=False)
    plot_model(base_model,to_file=MODEL.__name__+'.png')
    model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))
    model.summary()

    train = model.predict_generator(train_generator)
    test = model.predict_generator(test_generator)
    remove_dir(h5_full_path)

    with h5py.File(h5_full_path) as h:
        h.create_dataset("train", data=train)
        h.create_dataset("test", data=test)
        h.create_dataset("label", data=train_generator.classes)
```

代码17～19行，构建基本模型，并在后面追加一层平均池化。因为在后面代码中模型会使用全连接层，因此在这里添加一层AveragePooling。
代码26～29行，为保存向量到磁盘文件中。

II. 分别使用INCEPTIONV3和XCEPTION预训练模型输出特征向量并保存

```
write_gap(InceptionV3, (299, 299), inception_v3.preprocess_input)
write_gap(Xception, (299, 299), xception.preprocess_input)
```

6. 分别导入两个单模型及混合模型的权重向量的函数

```
def load_eigenvector(path_list=[]):
    x_train = []
    x_test = []
    for file_path in path_list:
        with h5py.File(file_path, 'r') as fp:
            x_train.append(np.array(fp['train']))
            x_test.append(np.array(fp['test']))
            y_train = np.array(fp['label'])

    x_train = np.concatenate(x_train, axis=1)
    x_test = np.concatenate(x_test, axis=1)
    x_train, y_train = shuffle(x_train, y_train)
    return x_train, y_train, x_test
```

7. 增加dropout层和全连接层并编译模型

```
def full_connect_layer(x_train, dropout_rate = 0.2):
    input_tensor = Input(x_train.shape[1:])
    model = Model(input_tensor, Dropout(dropout_rate)(input_tensor))
    model = Model(model.input, Dense(1, activation = 'sigmoid')(model.output))
    # 编译模型
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
    return model
```

代码3行，增加dropout层。
代码4行，增加全连接层，并将$Sigmod$函数做为激活函数。
代码6行，编译模型,优化器选择$Adam$算法。训练模型

```
def model_fit(model,x_train,y_train,chk_point_file,epochs = 10,batch_size = 128):
    remove_dir(chk_point_file)
    history_train = model.fit(x_train, y_train, validation_split = 0.2,epochs = epochs,
                              batch_size = batch_size, verbose=1,
                              callbacks=[ModelCheckpoint(filepath=chk_point_file, verbose=1, save_best_only=True)])
    return history_train
```

两个单模型及一个混合模型各训练一次，一共训练三次；训练模型同时对训练集进行分割，`validation_split = 0.2` 20%作为验证集，80%作为训练集；训练结果通过回调函数保存成文件。

8. 参数调优

```
epochs = 10
dropout_result= []
for rate in np.arange(0.1, 1.0, 0.1):
    model =  full_connect_layer(x_train, dropout_rate=rate)
    train_result = model.fit(x_train, y_train, batch_size=128, epochs=epochs, validation_split=0.2)
    dropout_result.append((rate,train_result.history['val_loss'],train_result.history['val_acc']))
```

通过对不同dropout丢弃率，从0.1～0.9尝试.

9. 结果分析展示
最后通过保存的checkpoint数据来用图像展示，并分析结果。

参考

[1] https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition#evaluation