UDACITY

Return to "Deep Learning" in the classroom

Dog Breed Classifier

审阅

代码审阅

HISTORY

Meets Specifications

恭喜你通过审阅，真的是很棒的工作！　　审阅你的项目也给我带来了很好的体验！

希望你能仔细阅读我的一些注解和建议，这能给你的模型带来一些提升。

此外，这个项目只是较为直观的教会你CNN的基础思路与架构，想要从事Computer Vision领域相关研究和工作的话，除了image classification之外，还要学习image segmentation、object detection、image generation等。

最后，提一点建议：希望你的学习过程能够不局限于我们提供出的示范性代码，去看一些论文、博客，大胆地做各种各样的尝试，并且将你做过的各种尝试、效果、分析记录在notebook里（事实上这也是我们让你自己动手完成project的初衷　）。这样你会收获更多哦~

总之，继续学习、不要止步！加油！

推荐你阅读以下材料来加深对 CNN和Transfer Learning的理解:

- CS231n: Convolutional Neural Networks for Visual Recognition
- Using Convolutional Neural Networks to Classify Dog Breeds
- Building an Image Classifier
- Tips/Tricks in CNN
- Transfer Learning using Keras
- Transfer Learning in TensorFlow on the Kaggle Rainforest competition
- Transfer Learning and Fine-tuning
- Building powerful image classification models using very little data
- 简述迁移学习在深度学习中的应用
- 无需数学背景，读懂 ResNet、Inception 和 Xception 三大变革性架构

相关论文:

- [VGG16] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION
- [Inception-v1] Going deeper with convolutions
- [Inception-v3] Rethinking the Inception Architecture for Computer Vision
- [Inception-v4] Inception-ResNet and the Impact of Residual Connections on Learning
- [ResNet] Deep Residual Learning for Image Recognition
- [Xception] Deep Learning with Depthwise Separable Convolutions

提交文件

本次提交包含所有必需的文件。

提交了要求中的所有文件。

第一步：检测人类

提交包含狗狗与人脸数据集前 100 张图片中，检测出的人脸的百分比。

Well done!

> 人脸图片中人脸识别率为: 100.00%
> 狗狗图片中人脸识别率为: 11.00%

你可以通过以下书写方式来简化代码:

```
def detect(detector, files):
    # return np.mean([detector(f) for f in files]) # 或这种写法
    return np.mean(list(map(detector, files)))
print('human: {:.2%}'.format(detect(face_detector, human_files_short)))
print('dog: {:.2%}'.format(detect(face_detector, dog_files_short)))
```

表明 Haar 级联检测是否是一种合适的人脸检测技术。

很不错的分析!

实际上，这个问题需要分情况来探讨。一方面我们要提升自己的算法来应对各种情况的输入图像，提升用户使用体验。而另一方面，在一些特定场景下，我们也需要对用户提出要求，尤其是在安全相关的领域。比如，在人脸识别解锁、支付等场景下，为了保证安全性，我们需要要求用户正脸面对镜头，并且不能佩戴口罩等遮挡物。

进一步提升人脸识别的准确度，可以尝试HOG(Histograms of Oriented Gradients)或一些基于深度学习的算法，如YOLO(Real-Time Object Detection algorithm)、FaceNet、MTCNN等。

此外，你可以使用[imgaug]来对训练集进行增强、扩充，以增加训练集中的多样性。

补充阅读材料:

- Tutorial - Face Detection using Haar Cascades
- Face Detection using OpenCV
- OpenCV Face Detection in Images using Haar Cascades with Face Count
- YouTube video - Haar Cascade Object Detection Face & Eye
- Haar caascade classifiers
- YouTube video - VIOLA JONES FACE DETECTION EXPLAINED
- How can I understand Haar-like feature for face detection?
- A simple facial recognition api for Python and the command line
- 这个知乎专栏介绍了目前主流的基于深度学习的人脸识别算法。

第二步：检测狗狗

提交包含狗狗与人脸数据集前 100 张图片中，检测出的狗狗的百分比。

Perfect!

> 人脸图片中狗的识别率为: 0.00%
> 狗狗图片中狗的识别率为: 100.00%

你可以通过以下书写方式来简化代码：

```
print('human: {:.2%}'.format(detect(dog_detector, human_files_short)))
print('dog: {:.2%}'.format(detect(dog_detector, dog_files_short)))
```

第三步：建立一个CNN区分狗狗的种类（从头开始）

提交指明了一个 CNN 模型架构。

实现了模型并对模型进行了很棒的分析！

最后一层卷积层后，你连续使用了了多个池化层，实际上这四层最大池化加起来等价于一层`pool_size=16, strides=16`的最大池化。而我更推荐你使用`GlobalAveragePooling2D`来取代`Flatten`以及上面的那么多层池化，`GlobalAveragePooling2D`可以大量减少模型参数，降低过拟合的风险，同时显著降低计算成本，这也是现在主流的一些CNN架构的做法。

我建议你增加一些Dropout [Ref]层来避免模型过拟合，或添加BatchNormalization [Ref]层来降低Covariate Shift并加速运算过程，这也是主流CNN架构中的常见做法。

要注意的是，使用`BatchNormalization`层时，我建议你在每个`Conv2D`或`Dense`层后、`Activation`前进行添加。这个视频演示了`BatchNormalization`是如何工作的。

如果你决定使用`BatchNormalization`层，可以参考以下代码：

```
model.add(Conv2D(16, (3, 3), strides=(1, 1), padding='valid'))
model.add(MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

将`MaxPooling2D`提至`BatchNormalization`和`Activation`前和放在它们后面是等价的，但是放在前面可以减少模型运算量。

更进一步，你可以尝试不同的模型结构，如更多的卷积层和全连接层、更多的节点数、使用不同类型的正则化层（`Dropout`、`BatchNormalization`等）、使用不同的权值初始化方案（`truncated_normal`、`xavier`等）、使用不同的激活函数（`LeakyReLU`、`eLU`等）、抉择使用`Flatten`还是`GlobalAveragePooling2D`等。

在实际应用中，你需要根据场景的不同来设计不同的模型架构、使用不同的超参数。对比各类结构和超参数给模型带来的影响，有助于你更好的理解模型的结构。

补充阅读材料:

- Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python
- Keras tutorial – build a convolutional neural network in 11 lines
- Image Classification using Convolutional Neural Networks in Keras
- 斯坦福大学的cs231n课程介绍了CNN结构功能以及参数选择的相关知识。

训练过的模型在测试数据集上至少达到 1% 的准确度。

测试集准确率达到了11.7225%，太棒了！

根据我个人实验，采用如下代码训练可以达到50%以上的准确率，不过需要很长的训练时间，建议在有GPU的设备上进行训练。你如果想要继续优化网络可以参考一下。

```
# 如果你的环境配置了tensorflow-gpu，可以使用如下代码调用GPU进行运算，速度会快很多
import keras.backend.tensorflow_backend as KTF
```

```
import tensorflow as tf
KTF.set_session(tf.Session(config=tf.ConfigProto(device_count={'gpu':0})))
```

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=[3, 3], input_shape=[224, 224, 3
]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=[3, 3]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=[3, 3]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters=64, kernel_size=[3, 3]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=[3, 3]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters=128, kernel_size=[3, 3]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=[3, 3]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters=128, kernel_size=[3, 3]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(GlobalAveragePooling2D())
model.add(Dropout(0.5))
```

```
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(133, activation='softmax'))

model.summary()
```

```
epochs = 100
batch_size = 32

train_datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
train_generator = train_datagen.flow(train_tensors, train_targets, batch_s
ize=batch_size)

val_datagen = ImageDataGenerator()
validation_generator = val_datagen.flow(valid_tensors, valid_targets, batc
h_size=batch_size)

checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.from_sc
ratch.hdf5',
                               verbose=1, save_best_only=True)

earlystopping = EarlyStopping(monitor='val_loss', min_delta=0.001, patienc
e=20, verbose=1)

model.fit_generator(
        train_generator,
        steps_per_epoch=len(train_files) // batch_size,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=len(valid_files) // batch_size,
        callbacks=[checkpointer, earlystopping])
```

提交指出了训练算法所用的 epochs 数。

作为实验，你可以适当提升epochs数来对模型进行更进一步的训练，直到验证集误差不再有所提升为止，看看你设计的这个模型的极限能到多少吧~

如果你想让算法自动选择epoch参数，并且避免epoch过多造成过拟合，我推荐你使用Keras中提供的early stopping callback（提前结束）方法。early stopping可以基于一些指定的规则自动结束训练过程，比如说连续指定次数epoch验证集准确率或误差都没有进步等。你可以参照[Keras' callback]官方文档来了解更多。

```
keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0,
 verbose=0, mode='auto', baseline=None, restore_best_weights=False)
```

`patience`参数代表了模型的valid loss连续多少回合没有提升就停止训练，默认设置为0，但是在实际使用中我并不推荐如此设置。因为在这种情况下只要验证集loss没有下降，训练立即就结束了，没有任何缓冲。而实际情况是模型可能只是陷入了一个局部最优，需要多几个epoch才能跳出来并且继续下降。所以我们一般的做法是将其设置为10左右，根据训练任务和速度有时还会更大一些，比如50、100，是具体情况而定。

更多阅读材料：

- How to train your Deep Neural Network
- Number of epochs to train on

第五步：建立一个CNN区分狗狗的种类

提交下载了对应于 Keras 与训练模型（VGG-19, ResNet-50, Inception, or Xception）的关键特征。

选择了ResNet，很不错！

更进一步，你可以尝试不同的特征提取模型，如VGG19、InceptionV3或Xception，然后对比这些模型的性能与优劣，并分析这些模型的性能为什么不同。因为代码模板都是一致的，这并不会占用你多少时间。

除此之外，我更推荐你尝试Xception，它在众多图像识别领域中拔得头筹。在本项目的预测任务中，它能够轻松达到85%以上的测试集合准确率。

提交指明了一种模型架构

Good job! 你按照要求实现了模型！

经过预训练的模型本身已经完成了大部分繁重的工作，所以这样一个相对简单的模型结构是个不错的选择。

这里我想补充一些迁移学习相关的知识。
迁移学习的思路就是将一个预训练的模型，通过新的训练集进行二次训练。分为三种形式：

- **Transfer Learning：冻结（将层设置为不可训练）预训练模型的全部卷积层，只训练自己定制的全连接层。**
    - 比如说我们这个项目中，使用的4个迁移训练的模型都是在IMAGENET训练集上已经训练好的，然后我们把卷积层以后的部分去掉，加上全新的未训练的层（相当于把卷积的部分保留并冻结，重新训练分类的部分），然后用我们提供的新的训练集进行二次训练。相比直接在我们的训练集上训练一个全

新的模型，迁移学习节省了大量的计算成本，同时因为IMAGENET数据集足够强大，可以导致更好的效果。

- **Extract Feature Vector：先计算出预训练模型的卷积层对所有训练和测试数据的特征向量，然后抛开预训练模型，只训练自己定制的简配版全连接网络。**
    - 在迁移学习中，迁移的模型（如ResNet）本身一般具有大量的参数，即使是进行迁移学习，也需要很强大的计算资源和计算时间（即使是使用GPU也需要数小时、数天甚至数周的时间）才能训练的动这么深的模型。即使把所有的层都冻结，那么在训练过程中每次更新梯度时，训练集都要和模型中的所有参数进行计算（大量的矩阵相乘），而即使这些参数是定死的、不需要进行更新，这个过程也是非常费时间的。而如果你是使用CPU在进行这一步，无疑会需要更多的时间。所以，Extract Feature Vector的做法是，把训练集经过预训练模型生成出bottleneck features，然后让你们直接通过bottleneck features进行训练。这种方法相当于是将整个模型拆分成了两个部分。第一步是将所有图片通过ResNet的卷积结构（所有层冻结），然后将数据"编码"成bottleneck features；第二步则是用这些bottleneck features训练我们后加的新的结构（分类器），即在本项目中你需要实现的部分。
    - 我们这个项目实际上采用的就是这种方法，因为参数冻结的部分本身就不需要更新，也不需要回传的参数，这种方法实际上和直接训练一个大型的迁移学习网络是差不多的。而这种做法可以省去每次更新过程中和原先模型中参数进行的大量矩阵运算，从而训练的速度就非常快了。
- **Fine-tune：冻结预训练模型的部分卷积层（通常是靠近输入的多数卷积层），训练剩下的卷积层（通常是靠近输出的部分卷积层）和全连接层。**

    - Fine-tune的形式下分不同程度的解冻原有层参数，甚至可以解冻所有层。实际上，预训练模型的每一层都可以自定义解冻，进行二次训练。相比冻结所有预训练模型卷积层，Fine-tune可以学到更多的特征知识，可以带来更好的效果，这种方法也是现在迁移学习中最为常用的做法。但是Fine-tune的代价就是需要大量的计算成本，包括计算时间和计算性能。
    - 如果你对Fine-tune感兴趣的话，具体如何解冻一些卷积层、使其可以二次训练，可以参考如下代码：

```
from keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(weights='imagenet', include_top=False)
for layer in base_model.layers[:NB_IV3_LAYERS_TO_FREEZE]:
    layer.trainable = False
for layer in base_model.layers[NB_IV3_LAYERS_TO_FREEZE:]:
    layer.trainable = True
```

- 更多内容，建议你参考keras官方文档中的示例：https://keras.io/applications/。

提交通过说明代价函数与优化方式编译结构。

Great work! 很好的定义了损失函数和优化器。

你选择了`categorical_crossentropy`作为损失函数、`rmsprop`作为优化器，干的不错！

但是，我更推荐你使用Adam [Ref] 或者 Adagrad[Ref]作为优化器，这也是目前最常使用的优化器算法。想要了解更多的话， An overview of gradient descent optimization algorithms这篇文章介绍了当前流行的一些优化器算法的优劣比较，Usage of optimizers in Keras这篇文章介绍了Keras中各类优化器的使用方法。

提交应指出为什么这一架构会在分类任务中成功，以及为什么早期的尝试不成功。

不错的解释！

更多阅读材料：

- ImageNet: VGGNet, ResNet, Inception, and Xception with Keras
- ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks
- (上一篇的中文翻译版)ResNet, AlexNet, VGG, Inception: 理解各种各样的CNN架构
- Systematic evaluation of CNN advances on the ImageNet

提交使用模型的检查点训练模型，并将拥有最佳交叉验证损失的模型权重保存下来。

Awesome！

训练过程中，注意到第2次epoch之后验证误差就几乎没有提升了，同时因为你保存了最优模型，意味着你后面的训练都是在浪费计算资源；同时也观察到，第20次epoch时，验证误差远大于训练误差，这说明模型出现了过拟合。思考并尝试尽量减轻这种过拟合现象吧~

提示：

- 添加dropout层可以很有效的避免模型过拟合；
- 添加batch normalization层可以降低Covariate Shift并加速运算过程，也能带来一些降低过拟合的效果；
- 数据增强（data augmentation）也可以增加模型的鲁棒性和泛化能力；
- 使用Early Stopping技术。

你可以用可视化的形式将训练过程中的loss曲线输出到notebook中，具体参考Display Deep Learning Model Training History in Keras这篇文章，这样可以让训练过程更为直观，你可以更方便地判断模型是否出现了欠拟合或过拟合。

提交读取了模型获得最小交叉验证损失的模型权重。

干得不错！你保存并读取了验证误差最小的模型！

测试数据集上的准确度达到了 60% 或更多。

很棒！测试集准确度达到了79.4258%。

你可以继续优化你的网络架构和参数选择或者尝试不同的bottleneck features，根据我的经验，ResNet最好可以优化到85%以上的准确率！而Xception可以优化到88%以上准确率！

提示：

- 添加dropout层可以很有效的避免模型过拟合；

- 添加batch normalization层可以降低Covariate Shift并加速运算过程，也能带来一些降低过拟合的效果；
- 数据增强（data augmentation）也可以增加模型的鲁棒性和泛化能力；
- 尝试使用不同的优化器，如Adam和Adagrad；
- 尝试其他bottleneck features，如Xception。

提交包含一个满足如下要求的函数，它以文件路径作为输入，并返回由 CNN 预测的狗品种。

很棒地完成了任务！

第六步：写出你自己的算法

提交使用第五步建立的 CNN 模型检测狗的种类。提交应对不同种类的输入图片有着不同的输出结果，并且提供实际（或最接近的）狗的种类。

出色地实现了算法！

将dog_detector放在face_detector之前进行判断是一个明智的做法，因为前者的准确率更高。

第七步：测试你的算法

提交应至少检测了 6 张图片，其中至少包含了 2 张人脸图片与 2 张狗狗图片。

尝试了许多不同的图片，很有趣~

对提升模型性能提出了很好的建议。干得不错！

以下是我对改进模型提出的建议，希望对你有帮助：

1. **交叉验证（Cross Validation）**
   在本次训练中，我们只进行了一次训练集/测试集切分，而在实际模型训练过程中，我们往往是使用交叉验证（Cross Validation）来进行模型选择（Model Selection）和调参（Parameter Tunning）的。交叉验证的通常做法是，按照某种方式多次进行训练集/测试集切分，最终取平均值（加权平均值），具体可以参考维基百科的介绍。
2. **模型融合/集成学习（Model Ensembling）**
   通过利用一些机器学习中模型融合的技术，如voting、bagging、blending以及staking等，可以显著提高模型的准确率与鲁棒性，且几乎没有风险。你可以参考我整理的机器学习笔记中的Ensemble部分。
3. **更多的数据**
   对于深度学习（机器学习）任务来说，更多的数据意味着更为丰富的输入空间，可以带来更好的训练效果。我们可以通过数据增强（Data Augmentation）、对抗生成网络（Generative Adversarial Networks）等方式来对数据集进行扩充，同时这种方式也能提升模型的鲁棒性。
4. **更换人脸检测算法**
   尽管OpenCV工具包非常方便并且高效，Haar级联检测也是一个可以直接使用的强力算法，但是这些算法仍

然不能获得很高的准确率，并且需要用户提供正面照片，这带来的一定的不便。所以如果想要获得更好的用户体验和准确率，我们可以尝试一些新的人脸识别算法，如基于深度学习的一些算法。

5. **多目标监测**
   更进一步，我们可以通过一些先进的目标识别算法，如RCNN、Fast-RCNN、Faster-RCNN或Masked-RCNN等，来完成一张照片中同时出现多个目标的检测任务。

下载项目

返回 PATH

给这次审阅打分