

机器学习纳米学位

毕业项目：《猫狗大战》 叶硕杨 优达学城

2018年07月13日

I. 问题的定义

项目概述

“猫狗大战”的毕业项目是 kaggle 的一个比赛项目（详情见：<https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition>）。此比赛项目的数据集分为两个：训练集和测试集。训练集共有25000张已经标记“dog”、“cat”名词的图片，各占一半；测试集是没有标签的12500张图片。

完成这个比赛项目需要应用机器学习中的深度学习、神经网络（主要是卷积神经网络 CNN）的相关知识。

选择这个项目是运用课程学到的深度学习中神经网络章节中学到的知识，来跟深入搭建、理解神经网络的作用和方法。

问题陈述

这个比赛的要求是参赛人员运用机器学习知识，搭建一个神经网络，通过对25000张有“dog”、“cat”标记的训练集图片进行学习，从而根据学习到的结果，判定给定的测试集里面的图片是“cat”还是“dog”，因此属于机器学习中的“二元分类”问题。

解决方法的难度在于，除了搭建合适的卷积神经网络模型，还需要对很多的参数进行调节，使得得到的模型具有更准确的识别率。

对于这个问题，大致的实现步骤为：

1. 数据的读取与预处理。将图片信息进行读取，并且预处理成为之后模型可以使用的数据结构。
2. 搭建卷积神经网络（CNN）。
3. 调参。调节CNN网络的参数。
4. 训练学习。使用预处理完成后具有标签的训练集（train）对模型进行训练。
5. 预测评估。使用测试集对学习完成的模型进行评估。

6. 如果符合项目要求，结束；如果不符合，回到步骤3，或者步骤3。

最后的要求是：达到或超过 kaggle 上 Public Leaderboard 的前10%的成绩。

评价指标

kaggle网页上说明，本问题采用的 log loss 进行训练结果的评估^[1]：

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

其中：

- n 是test测试集的图片数量
- \hat{y}_i 是可能是dog狗的预测可能性
- y_i 如果图片是狗则为1，如果是猫则为0
- $\log ()$ 是自然底数对数函数

II. 分析

数据的探索

本项目使用的全部的数据集包括：训练集 (train) 和测试集 (test)。

训练集：是模型学习的全部数据集，一共25000张 jpg 图片，通过文件名的方式做了类别的标记，“dog”和“cat”各12500张，以数字为区分，例如“cat.17.jpg”、“dog.57.jpg”等。

测试集：一个12500张 jpg 图片，没有标签，名字只是使用1~125000进行命名，用于最后的测试评分。

数据集的一些特征：

1. 两个类别“dog”、“cat”的数据数量是相等的，各为125000个训练样本，图2-1展示了部分数据样本。

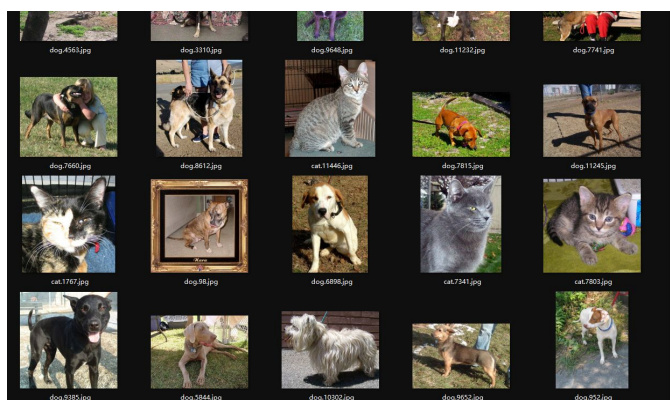


图2-1 数据集部分样本

- 所有的图片尺寸不是统一的。最小的尺寸为60*39px，最大的为1023*768px，见下图2-2与图2-3。

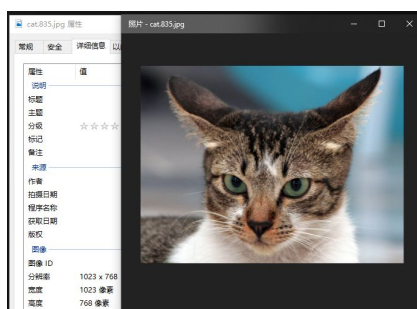


图2-2 cat.835.jpg

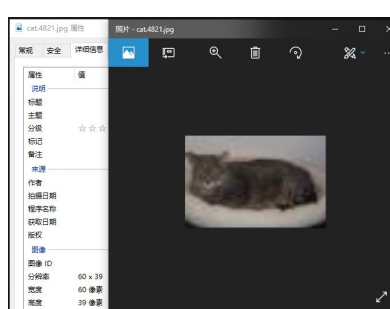


图2-3 cat.4821.jpg

- 异常值。训练集中存在一些内容既不是“dog”也不是“cat”的图片，如下列展示出来的一部分，因此训练模型的时候要注意过拟合的产生使得模型变形。



图2-4 cat.7564.jpg

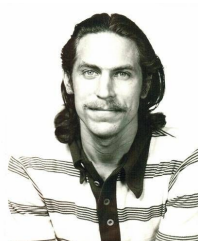


图2-5 cat.7377.jpg



图2-6 cat.9171.jpg



图2-7 dog.9517.jpg

- 判别的“猫”、“狗”场景不唯一，有单个猫狗出现、有多个猫狗出现、有猫狗不是图片主体的情况、猫狗同时出现等等，如下图：



图2-8 cat.16.jpg



图2-9 cat.2236.jpg



图2-10 dog.1486.jpg



图2-11 cat.2159.jpg

探索性可视化

本项目主要的数据特征就是数据集里面图片的尺寸，观察图片尺寸是否差异较大，为后续建立模型后的调整建立认识基础，可视化^[2]图片见图2-12、图2-13、图2-14。

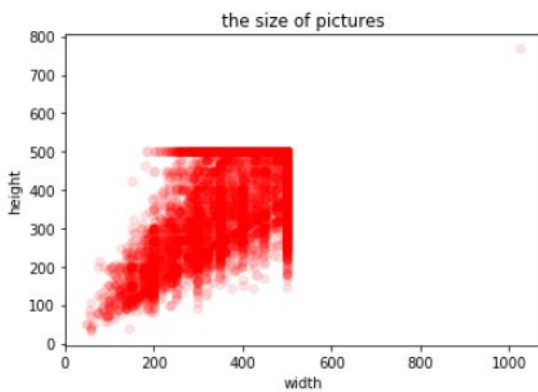


图2-12. cat width-height 分布

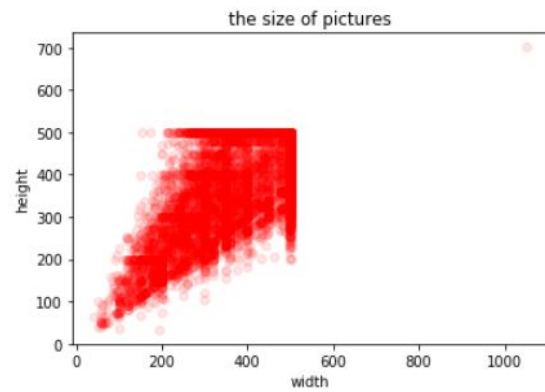


图.2-13 dog width-height 分布

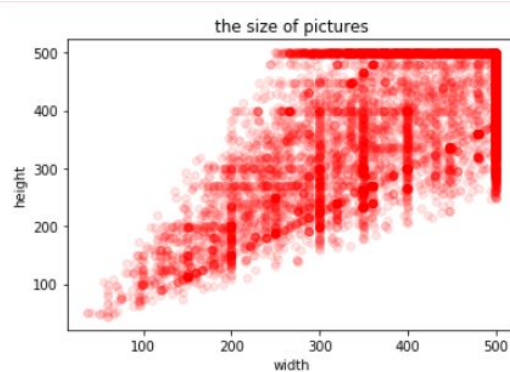


图2-14. test width-height 分布

上面图片中，图*-8是训练集的 cat 的图片尺寸分布图，图*-8是训练集中 dog 的图片尺寸分布图，图*-8是测试集的图片尺寸分布图。

可以看出，训练集中大部分图片的尺寸，width 在200~500px，height 在200~500px之间。test集中的尺寸分布类似。

也有部分图片尺寸特别小（100*100px以下），分辨率不高，这部分作为训练集会对模型有一定不好的作用，但是占比相对较小，可以先保留处理，观察搭建的模型受到的影响有多少。

算法和技术

本项目是图片领域的二元分类问题。对于这个问题，深度学习中的卷积神经网络（Convolutional Neural Network，CNN）。卷积神经网络的是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，^[3]，对于大型图像、语音识别的问题处理上有出色表现。

并且，随着数据的增加，使得卷积神经网络的实现有了数据基础，而不再是理论上的概念；随着硬件的计算能力按照“摩尔定律”指数级提升，特别是 GPU 的计算能力的大幅提高，使得卷积神经网络的快速运行成为可能。两个因素使得卷积神经网络的成长和普遍运用成为可能。

卷积神经网络由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构。

卷积神经网络主要的结构及作用如下：

- 卷积层。网络中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法最佳化得到的。卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网路能从低级特征中迭代提取更复杂的特征。
- 激活层。使用修正线性单元（Rectified linear unit, ReLU）作为神经元的激活函数，相比较其他的函数，具有梯度不饱和性，且能够加快计算速度。
- 池化层。池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。通常来说，CNN的卷积层之间都会周期性地插入池化层。
- 全连接层。全连接层（fully connected layers，FC）的作用主要就是实现分类（Classification）^[2]，<https://zhuanlan.zhihu.com/p/33841176>，起到将学到的“分布式特征表示”映射到样本标记空间的作用。

一个典型的卷积神经网络的结构如下面所示^[4]：

Input-> Conv-> ReLU-> Conv-> ReLU-> Pool-> ReLU-> Conv-> ReLU-> Pool->Fully Connected

输入→卷积→ReLU→卷积→ReLU→池化→ReLU→卷积→ReLU→池化→全连接

基准模型

这个毕业设计确定的基准模型，选择一个现有的模型得分作为基准（Benchmark），即达到 kaggle Public Leaderboard 前10%，次项目的 Leaderboard 一个1314个成绩，排名第131名的成绩是 Reziproke 的 logloss 得分0.06127，见图2-15。因此，最后模型的结果 logloss 得分要低于 0.06127。

130	▼35	RaviKiranK		0.06114	35	1y
131	▼35	Reziproke		0.06127	4	1y
132	▼35	mathieuzaradzki		0.06149	32	1y

图2-15

III. 方法

数据预处理

1. 分类。将 train 文件夹内的25000张按照“dog”、“cat”的类别分为两类，并且对数据打上标签。根据数据集的特点，train 里面的图片名字为“cat/dog+数字.jpg”的命名方式，所以根据名字前三个字母分为两类 numpy 数据，并且创建一个顺序对应的标签 numpy 数据。
2. 统一尺寸大小。后续选取的模型需要输入的图片尺寸为（299，299，3），而原始的图片尺寸大小不一，因此使用 cv2.resize（）方法将图片尺寸统一化成（299，299）的（高，宽）尺寸。
3. 无序化，产生验证集（validation）。使用 sklearn 的 train_test_split 函数将上一步得到的 train 数据进行打乱处理，并且按照8：2的比例产生出验证集（validation）
4. 对于尺寸特别大、特别小的图片数据先不进行剔除；同样，对于明显不是“dog”/“cat”的图片也不进行筛选剔除。因此这些图片占比很少，对于模型的学习的影响不会太大；并且模型未进行学习，无法自动识别异常图片，人工解决显然不是正确的方法。

执行过程

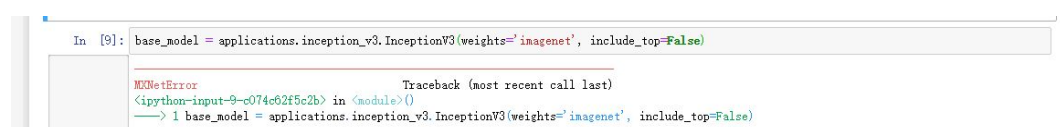
1. 数据预处理过程。首先采用的方法：通过 os.symlink 方法，将 train 里面的图片数据对应到新建的两个“catRun”、“dogRun”文件夹内，避免复制图片的较

大的工作量。然后使用 keras 里面 image 函数中的 ImageDataGenerator^[5]，采用 flow_from_directory 的方法将 train 数据进行尺寸转化。但是因为项目里第一次使用 keras，对于部分函数的使用没有全面了解，遇到了当时无法解决的困扰，然后选择的另一个方法：将图片导入成 numpy 数据，然后转成统一的 (299, 299) 尺寸，这样处理的 code 表达的也相对简洁清晰。

2. 采用 numpy.load 的方法载入数据会比较缓慢，后续改为了预分配取代 append 命令；将 dtype 设置成 uint8。最后数据预处理得到目标结果。
3. 交流中获得的更加节省内存的预处理方法为：先列出所有文件名、对应的 label，然后对文件名和 label 进行 train_test_split，然后使用 uint8 的格式读取图片，最后在模型里面进行预处理。
4. 搭建平台。由于自己使用的笔记本的 OS 是 windows 10 家庭版，因此进行第一次预处理尝试的时候，发现是无法建立 Symlink 的；并且，根据上一个作业项目的经验教训，自己电脑的计算能力不足以完成本项目的计算任务（主要是缺少良好的 GPU），因此，后续的大部分实现的过程都是在 AWS 的计算机上进行的。
5. 训练速度改进。做机器学习之前的作业项目，由于不需要进行大量的数据计算训练，因此都是在笔记本上进行的，但是最后一次的作业项目使用 CPU 进行训练的时候，花费了比较多的时间，因此前期就决定所有的训练任务均在 AWS 上进行^[6]。
6. 花费了一定的时间学习使用 AWS。刚开始接触 AWS 的时候，不理解其中的使用机制，遇到很多低级的错误。

第一、工单的申请。对于想要使用更加节省的竞价实例，失败了2次后，在 mentor 的指导下，填写了正确的工单申请；

第二、环境的选择。有选错过程序的环境，TensorFlow 的环境选成了 MXNet 环境，导致正确的代码无法运行，见图3-1；



```
In [9]: base_model = applications.inception_v3.InceptionV3(weights='imagenet', include_top=False)

RuntimeError                                Traceback (most recent call last)
<ipython-input-9-c074c62f5c2b> in <module>()
----> 1 base_model = applications.inception_v3.InceptionV3(weights='imagenet', include_top=False)
```

图3-1 环境错误情况

第三、PuTTY 的模式设置。后来摸索出来可以进行保存整体设置内容，节省的很多操作时间；

第四、AWS 的 AMI 的保存。前几次每次运行 AWS 都重新配置环境，浪费了好多时间，后来在学习群里交流后，把基础环境保存成自定义的 AMI，后面节省的很多操作时间。

7. 之前查询资料的时候，得知模型进行融合操作，可以提升模型性能，因此设定的大致方法是：先尝试单个模型的情况，然后进行参数的调节，如果能够提升

成绩，则使用单模型；如果无法达到要求，则再进行模型的融合^[7]。最后单模型调参的过程能够达到最后的项目要求，最后确定为单模型。

8. 第一次使用的模型是 keras 官方文档给出的例子中的 ResNet50，但是第一次得到的模型的结果并不理想，因此改为了模型更深的 InceptionResNetV2。另一个考虑是，ResNet50 的图片尺寸要求和其他模型，比如 Inception、Xception 都不一样，为了保证之后融合有比较顺利的处理，因此不再考虑 ResNet50。

完善

1. 第一的图形预处理，使用的是 ImageDataGenerator 函数进行，将输入的数据进行了归一化处理，见图3-2，如下图。但是经 mentor 提醒、查询资料，得知 InceptionResNetV2 的模型需要的输入的数据在 -1~1 之间，并不是 0~1 之间。结合第二点，将数据集的预处理得到了改进。

```
1 train_data_dir = 'trainRun'
2 train_datagen = image.ImageDataGenerator(rescale=1./255,
3                                           shear_range=0.2,
4                                           zoom_range=0.2,
5                                           horizontal_flip=True)
```

图3-2 错误的预处理方法

2. 第一次做出的初版，模型的输入预处理错误了，只是进行了图形数据的预处理，没有添加模型自己的预处理，见图3-3，导致训练结果完全不在预期的数值上，很离谱，见图3-4。

```
8 base_model = InceptionResNetV2(weights='imagenet', include_top=False)
9 model = Model(base_model.input, GlobalAveragePooling2D(base_model.output))
10

/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/h5py/_init_.py:36: FutureWarning: Conversion of the second argument of issubdtype from float to np.float64 is deprecated. In future, it will be treated as np.float64 == np.dtype(float).type.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.

Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.7/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [=====] - 3s 0us/step
```

图3-3 未添加预处理函数

```
9 model.compile(optimizer='adadelta',
10              loss='binary_crossentropy',
11              metrics=['accuracy'])

In [15]: 1 model.fit(X_train, y_train, batch_size=16, epochs=5, validation_data=(X_valid, y_valid))

Train on 20000 samples, validate on 5000 samples
Epoch 1/5
20000/20000 [=====] - 216s 11ms/step - loss: 0.1036 - acc: 0.9377 - val_loss: 5.9191 - val_acc: 0.3622
Epoch 2/5
20000/20000 [=====] - 147s 7ms/step - loss: 0.1115 - acc: 0.9567 - val_loss: 8.0719 - val_acc: 0.4992
Epoch 3/5
20000/20000 [=====] - 187s 9ms/step - loss: 0.1109 - acc: 0.9570 - val_loss: 8.0719 - val_acc: 0.4992
Epoch 4/5
20000/20000 [=====] - 196s 10ms/step - loss: 0.1005 - acc: 0.9591 - val_loss: 8.0719 - val_acc: 0.4992
Epoch 5/5
20000/20000 [=====] - 212s 11ms/step - loss: 0.1034 - acc: 0.9613 - val_loss: 8.0719 - val_acc: 0.4992

Out[15]: keras.callbacks.History at 0x7f9417dfc5d0

In [ ]: 1
```

图3-4 最后的结果

3. 激活函数的调节。第一次搭建模型的使用的激活函数是 relu，见图3-5，这个激活函数的作用对与搭建的模型有偏差，导致最后模型无法很好收敛，见图3-6。如下图。

后来更改模型的 layer，并且将激活函数改为 sigmoid，使得模型该激活层的输出对学习能够产生比较好的收敛。

```
1 # 添加全局平均池化层
2 x = base_model.output
3 x = GlobalAveragePooling2D()(x)
4 # 添加全连接层
5 x = Dense(1024, activation='relu')(x)
```

图3-5 激活函数代码

```
- 1s 71us/step - loss: 0.6924 - acc: 0.5657 - val_loss: 0.6918 - val_acc: 0.8520
- 1s 27us/step - loss: 0.6912 - acc: 0.6383 - val_loss: 0.6905 - val_acc: 0.8116
- 1s 26us/step - loss: 0.6899 - acc: 0.7096 - val_loss: 0.6892 - val_acc: 0.7804
- 1s 27us/step - loss: 0.6887 - acc: 0.7436 - val_loss: 0.6880 - val_acc: 0.9014
- 1s 27us/step - loss: 0.6876 - acc: 0.7577 - val_loss: 0.6869 - val_acc: 0.9474
- 1s 26us/step - loss: 0.6863 - acc: 0.7775 - val_loss: 0.6856 - val_acc: 0.9528
- 1s 27us/step - loss: 0.6851 - acc: 0.7760 - val_loss: 0.6844 - val_acc: 0.9658
- 1s 26us/step - loss: 0.6839 - acc: 0.7855 - val_loss: 0.6832 - val_acc: 0.9596
```

图3-6 激活函数为 ReLU 的训练结果

- Dropout 比例。因为选择的单模型 InceptionResNetV2 的深度有572，模型的原始输出很多分类，加入 Dropout 层是为了避免过拟合。加入Dropout之前课程的学习和作业内容，Dropout 的比例一直设置成默认的0.5，但是第一次的训练的结果并不是特别高，于是调低了数值，发现得到的结果更好一些，测试了几个，最终将 Dropout 的比例设置成0.25。
- 优化器 optimizer 选择调试。之前对于 optimizer 没有特别深入的了解，根据单模型的参考资料选择的是“adadelat”，但是会遇到瓶颈期。后来参考了官方文档的例子，将放开一定 layers 后第二次训练的 optimizer 改为了 SGD，设置的参数是（lr=1e-4, momentum=0.9），最后模型的成绩得到提升，最后达到了项目要求。
- fine tune 调节。对于放开一些神经网络 layer 进行微调，首先放开了23层，最后结果没有达到要求，如图3-7；后来减少改为放开13层，fine tune 的结果达到了目标要求。

```
[17]: 1 for layer in model.layers[760]:
      2     layer.trainable = True

[20]: 1 model.fit(x_train, y_train, batch_size=16, epochs=5, validation_data=(x_valid, y_valid))

Train on 20000 samples, validate on 5000 samples
Epoch 1/5
32/20000 [.....] - ETA: 1:57 - loss: 0.0377 - acc: 1.0000
/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/keras/engine/training.py:498: UserWarning: Discrepancy between trainable weights and collected trainable weights, did you set `model.trainable` without calling `model.compile` after?
'Discrepancy between trainable weights and collected trainable'
20000/20000 [=====] - 142s 7ms/step - loss: 0.0908 - acc: 0.9669 - val_loss: 0.0523 - val_acc: 0.9850
Epoch 2/5
20000/20000 [=====] - 190s 9ms/step - loss: 0.0879 - acc: 0.9669 - val_loss: 0.0693 - val_acc: 0.9806
Epoch 3/5
20000/20000 [=====] - 214s 11ms/step - loss: 0.0853 - acc: 0.9671 - val_loss: 0.0453 - val_acc: 0.9868
Epoch 4/5
20000/20000 [=====] - 218s 11ms/step - loss: 0.0838 - acc: 0.9697 - val_loss: 0.0653 - val_acc: 0.9824
Epoch 5/5
20000/20000 [=====] - 227s 11ms/step - loss: 0.0816 - acc: 0.9700 - val_loss: 0.0774 - val_acc: 0.9786

t[20]: <keras.callbacks.History at 0x7f1062cdd610>
```

图3-7 放开23层训练结果

IV. 结果

模型的评价与验证

解决本项目的方法是搭建一个卷积神经网络，但并不是从0开始搭建，而是使用 Keras 中一个对于图片的识别的通用模型：InceptionResNetV2^[8] 进行迁移学习^[9]，这个模型

是带有预训练权值的深度学习模型，因此只要站在巨人的肩膀上进行针对本项目的模型搭建，就可以得到一个效果良好的深度学习模型。

迁移学习（transfer learning）作为一个深度学习重要的手段，被证明在图像识别领域有良好效果的方法。本项目的模型是去掉了模型最后的多类的全连接层，加入了池化层和只进行二元判定的全连接层，通过调节、设定关键的参数，比如 batch_size、Dropout 比例等，是模型的成绩提升到目标要求。

搭建的模型和结果具有可靠性。文章之前对训练数据的分析得出，训练集的数据有异常值，图片内容不是“dog”/“cat”的图片，也被标记为这两个标签，但是占比不大。所以最后的模型是在具有一定误差的训练集上进行学习的，最后模型的测试结果为0.05600，如图4-1，达到目标要求。如果训练集上图片和对应的标签更加正确，最终模型可以学习到更加正确的分类模式。



图4-1 最终模型的测试结果

合理性分析

本项目搭建的模型，训练集上的 logloss 为 0.0623，验证集上为 0.0380，最后将学习到的模型对验证集进行预测得到的 csv 文件提交到 kaggle 官网进行评估，最后的 logloss 得分是 0.05600，比基准模型的 0.06127 要低。因此模型具有良好的学习能力，没有出现拟合和欠拟合的现象。

模型的搭建是合理的。

模型在训练集上的学习，在 layer_trainable=False 的情况下，训练得到的最后结果是：loss: 0.0750 - acc: 0.9719 - val_loss: 0.0617 - val_acc: 0.9848，如图；将770~783层的 trainable 设定为 True 后，训练的最后得分为：loss: 0.0623 - acc: 0.9774 - val_loss: 0.0380 - val_acc: 0.9908，如图。

V. 项目结论

结果可视化

本项目最重要的指标就是对图片的预测的准确性（Accuracy）和逻辑回归损失函数（logloss）数值。图4-2是训练集和验证集的 loss 值，图4-3是训练集和验证集的 Accuracy 值。

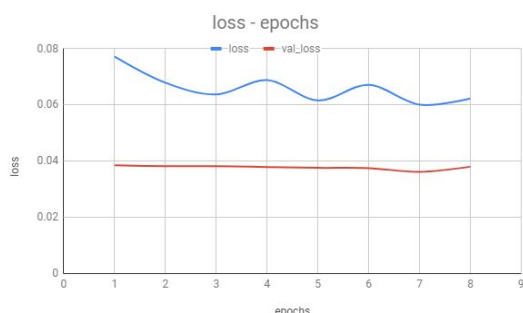


图4-2

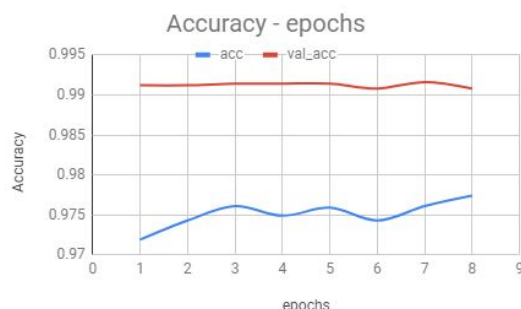


图4-3

从上图可以观察分析得出，随着训练次数的增加，训练集的 loss 值呈波动下降趋势，而验证集则基本平稳，有略微下降的趋势；训练集的准确率呈波动上升的趋势，而验证集的基本不变，维持在0.99高分之上。说明模型的结果良好，表现出良好的泛化能力。

对项目的思考

本项目做完之后，主体思路的搭建还是很直接的，但是刚开始第一次从零搭建整个神经网络，当时内心的压力还是很大。

整个项目的主要分为图形数据集的分类读取、预处理、带有预训练权值模型构建基准模型（base model）、将基准模型迁移学习为本项目的模型、对模型的关键参数进行设定微调模型。

项目中比较有意思的一点是，可以使用成型的模型进行迁移学习，然后自己通过Keras 亲手搭建了一个满足项目的模型，成就感满满。

比较困难的地方有两点。一点是 AWS 云服务的使用，对于程序领域之外的新人进入，一下子需要完全使用命令行对计算机进行操作和交互，十分的痛苦；另一点是，深度学习一个重要的是对数据进行预处理，但是函数的方法method 很多不是很清楚，有部分前期的处理自己理解错了，导致自己迟迟无法顺利看到模型的学习过程。

最终的模型有点超出自己的期望。之前打算顺利搭建完一个模型之后，就进行mentor 指点的模型融合方法，但是后来单模型的结果已经满足要求了，就先将毕业项目完成。

作为一个特别的项目，我认为它在通用的场景下并不能解决这类问题。因为识别的问题还没有包括“非猫非狗”的分类设定，对于通用场景下的多样的图片识别，还不具有相当的解决能力。但是这个项目的练习，对于初学者来说，对深度学习，尤其是卷积神经网络有了更加深刻的理解，对于后续的学习和成长，具有很好的基础作用。

需要作出的改进

1. 可以运用 Keras 具有的数据处理函数，将给定的数据进行数据增强，可以不使用全部的数据集进行学习，从而达到一样的效果。
2. 数据预处理。数据的异常值可以进行一定的剔除，比如先搭建一个初步筛选的模型，将表现为异常值的数据进行提取，将尽可能多的一场图片从训练集上排除，让模型学习到更加准确的特征模式。
3. 模型融合。对于多个模型的方法，可以结合多个模型的特点、优点，可以让模型的学习速率、准确性得到一定的提高。

参考文献

- [1] <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition#evaluation>, kaggle 中 dog_vs_cat 的评价说明
- [2] https://github.com/fountainhead-gq/dogs_vs_cats/blob/master/visualization.py, 可视化方法
- [3] <https://zh.wikipedia.org/wiki/卷积神经网络>, 维基百科
- [4] 机器之心, CNN(卷积神经网络)是什么? 有入门简介或文章吗?, <https://www.zhihu.com/question/52668301/answer/131573702>,
- [5] <https://keras.io/>, keras 官方文档
- [6] <https://zhuanlan.zhihu.com/p/33176260>, 利用AWS学习深度学习 For Udacity P5 (第三篇 : AWS Udacity P5 GPU环境构建)
- [7] 杨培文, 手把手教你如何在Kaggle猫狗大战冲到Top2%, <https://zhuanlan.zhihu.com/p/25978105>,
- [8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567v3 [cs.CV] 11 Dec 2015
- [9] Francois Chollet, Building powerful image classification models using very little data , <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html> ,
- [9] <https://cs231n.github.io>, cs231n官方课程、文档
- [10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, Rethinking the Inception Architecture for Computer Vision, arXiv:1512.00567 [cs.CV]
- [11] 程序园, keras 迁移学习 , 微调 , model的predict函数定义, <http://www.voidcn.com/article/p-xozdmoab-bew.html>
- [12] Greg Chu, How to use transfer learning and fine-tuning in Keras and Tensorflow to build an image recognition system and classify (almost) any object, <https://deeplearningsandbox.com/how-to-use-transfer-learning-and-fine-tuning-in-keras-and-tensorflow-to-build-an-image-recognition-94b0b02444f2>