机器学习纳米学位

I. 定义

1.1 项目概述

项目对猫狗图像进行分类, 是 kaggle 在 2013 年发布的一个比赛项目[1].

项目应用了卷积神经网络，对预处理网络进行了应用，预训练时用到的网络结构有：VGGNet[2]，Xception[3]，InceptionResNetV2[4]。

项目选择的数据集是 kaggle 猫狗大战数据集（Dogs vs. Cats Redux: Kernels Edition），训练集为 25000 张图片，猫狗各一半，标注在文件名上。测试集 12500 张，没标定是猫还是狗，由数字 id 命名文件。部分如下：

图 1．部分图像示例

这一项目使用的深度学习可用于进一步的图像分析模型学习，后期在已用的网络基础上可以深入进行 RNN, GAN 等理论与应用的学习.

1.2 问题陈述

对项目的猫狗图片进行二分类，图片的大小有不同，背景下各异，这都为辨识增加了难度.

所以首先要预处理图片．然后预训练并构建模型，在已有模型基础上进行模型融合，最后进行预测和可视化.

问题的处理（二分类）分为几个部分，在下一部分文中会进行细化：

１）图片预处理；２）进行模型预训练；３）构建模型，载入预训练结果；４）训练与模型预测

最终输出的结果：对每张图片预测是狗的概率，1=狗，0=猫.

1.3 评价指标

模型和结果的指标是 LogLoss, 计算方法:

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

LogLoss 越小越优。上式中：n 为测试集中图片数量

$\hat{y}_i$ 为预测是狗的概率

y_i 为 1 若图片是狗，为 0 若图片是猫

$\log()$ 是 e 为底的自然对数

说明：Log loss 也被称为 logistic 回归 loss，或者交叉熵 loss(cross-entropy loss)，用于概率估计。它通常用在神经网络上，以最大期望（EM：expectation-maximization）的变种的方式，用于评估一个分类器的概率输出，而非进行离散预测。

LogLoss 函数的标准形式：

$$L(Y, P(Y|X)) = -\log P(Y|X) \quad (2)$$

推导：损失函数 L(Y, P(Y|X))表达的是样本 X 在分类 Y 的情况下，使概率 P(Y|X)达到最大值（换言之，就是利用已知的样本分布，找到最有可能导致这种分布的参数值；或者说什么样的参数才能使我们观测到目前这组数据的概率最大）．最大化 P(Y|X)就等同于最小化 L。

逻辑回归（**Logistic Regression**）的 **P(Y=y|x)**表达式如下

$$P(Y=y|x)=\begin{cases} h_\theta(x)=g(f(x))=\frac{1}{1+exp\{-f(x)\}} & ,y=1 \\ 1-h_\theta(x)=1-g(f(x))=\frac{1}{1+exp\{f(x)\}} & ,y=0 \end{cases} \quad (3)$$

将其代入上式，可得：

$$L(y,P(Y=y|x))=\begin{cases} \log(1+exp\{-f(x)\}) & ,y=1 \\ \log(1+exp\{f(x)\}) & ,y=0 \end{cases} \quad (4)$$

因为各个观测样本之间相互独立，那么它们的联合分布为各边缘分布的乘积。对得到似然函数两边取对数，得（m 为样本数）：

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] \quad (5)$$

即（1）式

II. 分析

2.1 数据的探索

数据集中训练集和测试集图片数量统计如表 1 所示：

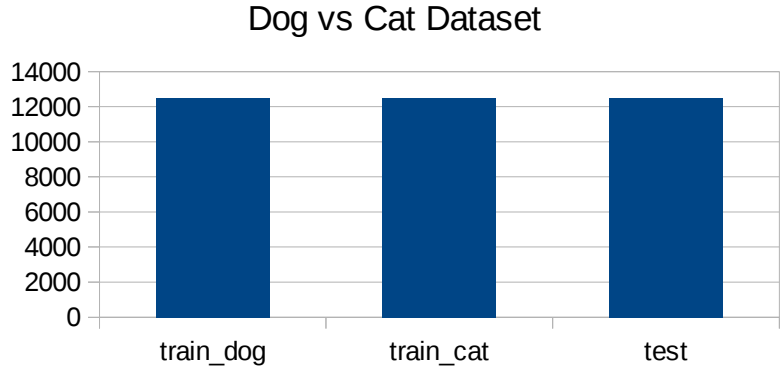


表 1 .数据集统计

数据集中尺寸并不一致，尺寸分布的散点图如图 2 （坐标值为像素）。除个别样本，测试集与训练集的尺寸分布基本一致。尺寸分布在对角线两侧，说明图片既有“宽扁”也有“窄高”的。

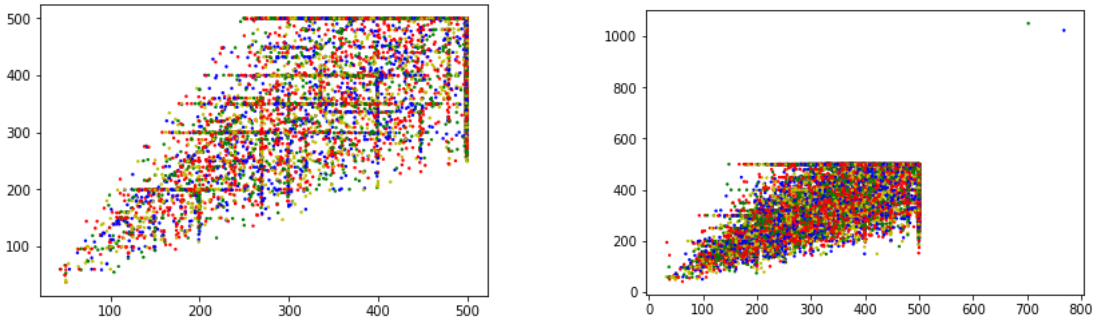
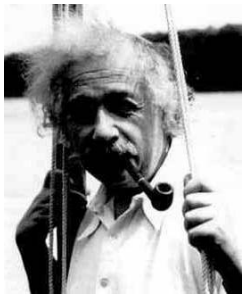


图 2．图片尺寸散点图（左为测试集，右为训练集）

另外 Kaggle 提供的数据中存在异常数据，它们混杂在 train 文件夹的 25000 张图片中．例如，

图片背景较复杂：

i)dog.6725.jpg：显然，人比狗多；



ii)异常数据: dog.1773.jpg,

cat.6345.jpg
cat.5351.jpg
cat.5418.jpg
cat.4308.jpg
cat.12424.jpg
cat.7564.jpg
cat.3216.jpg
cat.11184.jpg
cat.10636.jpg
cat.7682.jpg
cat.9983.jpg
cat.12476.jpg
cat.9171.jpg
cat.4688.jpg
cat.2520.jpg
cat.8456.jpg
cat.10029.jpg
cat.10712.jpg
cat.7372.jpg
cat.4338.jpg
cat.3672.jpg
cat.7377.jpg

表 2．部分筛选出的异常图片

原始图片 → 训练数据文件夹分类 → 尺寸统一生成

2.2 算法和技术

2.2.1 神经网络

神经网络算法领域最初是被对生物神经系统建模这一目标启发，但随后与其分道扬镳，成为一个工程问题，并在机器学习领域取得良好效果。

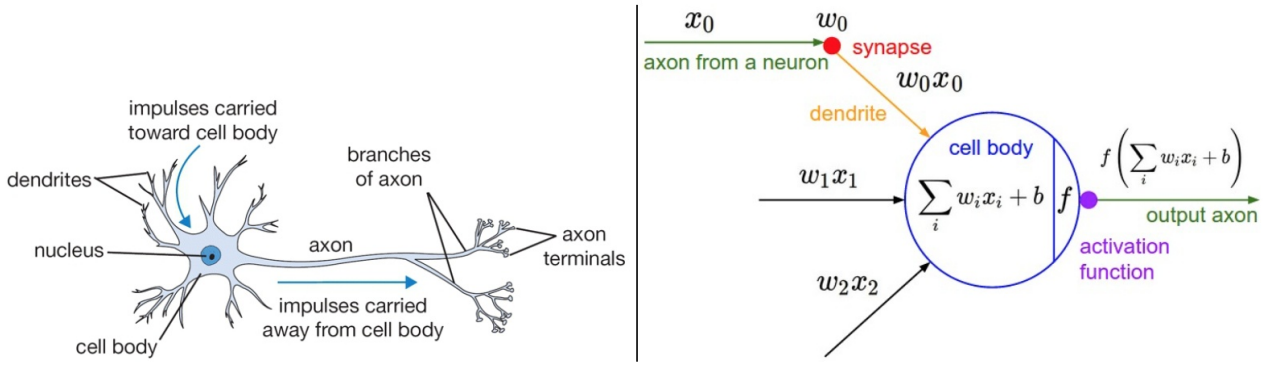


图 3．神经网络的类比示意

大脑的基本计算单位是神经元（neuron）。人类的神经系统中大约有 860 亿个神经元，它们被大约 10^{14} 至 10^{15} 个突触（synapses）连接起来。下面图表的左边展示了一个生物学的神经元，右边展示了一个常用的数学模型。每个神经元都从它的树突获得输入信号，然后沿着它唯一的轴突（axon）产生输出信号。轴突在末端会逐渐分枝，通过突触和其他神经元的树突相连。

在神经元的计算模型中，沿着轴突传播的信号（比如 x_0 ）将基于突触的突触强度（比如 w_0 ），与其他神经元的树突进行乘法交互（比如 w_0x_0 ）。其观点是突触的强度（也就是权重 w ），是可学习的且可以控制一个神经元对于另一个神经元的影响强度（还可以控制影响方向：使其兴奋（正权重）或使其抑制（负权重））。在基本模型中，树突将信号传递到细胞体，信号在细胞体中相加。如果最终之和高于某个阈值，那么神经元将会激活，向其轴突输出一个峰值信号。在计算模型中，我们假设峰值信号的准确时间点不重要，是激活信号的频率在交流信息。基于这个速率编码的观点，将神经元的激活率建模为激活函数（activation function），它表达了轴突上激活信号的频率。由于历史原因，激活函数常常选择使用 sigmoid 函数 ，该函数输入实数值（求和后的信号强度），然后将输入值压缩到 0-1 之间。

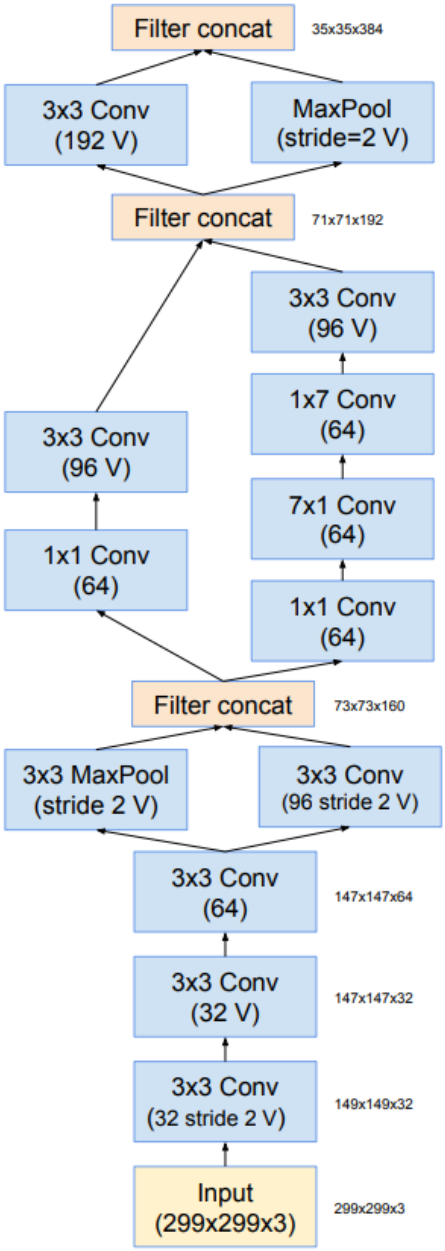


图 4. Inception-ResNet-v2 网络结构

2.2.2 卷积神经网络与预训练模型

卷积神经网络和常规神经网络非常相似：它们都是由神经元组成，神经元中有具有学习能力的权重和偏差。每个神经元都得到一些输入数据，进行内积运算后再进行激活函数运算。整个网络依旧是一个可导的评分函数：该函数的输入是原始的图像像素，输出是不同类别的评分。在最后一层（往往是全连接层），网络依旧有一个损失函数（比如 SVM 或 Softmax），并且在神经网络中我们实现的各种技巧和要点依旧适用于卷积神经网络。

卷积层的参数是有一些可学习的滤波器集合构成的。每个滤波器在空间上（宽度和高度）都比较小，但是深度和输入数据一致。举例来说，卷积神经网络第一层的一个典型的滤波器的尺寸可以是 5x5x3（宽高都是 5 像素，深度是 3 是因为图像应为颜色通道，所以有 3 的深度）。在前向传播的时候，让每个滤波器都在输入数据的宽度和高度上滑动（更精确地说是卷积），然后计算整个滤波器和输入数据任一处的内积。当滤波器沿着输入数据的宽度和高度滑过后，会生成一个 2 维的激活图（activation map），激活图给出了在每个空间位置处滤波器的反应。

一个简单的卷积神经网络是由各种层按照顺序排列组成，网络中的每个层使用一个可以微分的函数将激活数据从一个层传递到另一个层。卷积神经网络主要由三种类型的层构成：卷积层，汇聚（Pooling）层和全连接层（全连接层和常规神经网络中的一样）。通过将这些层叠加起来，就可以构建一个完整的卷积神经网络。这里采用了 3 个较成熟的预训练网络 VGG19，Xception， Inception-ResNet-v2（图 4）。预训练之后的权重进行融合，再进行训练。

在 2014 年，VGG 模型架构由 Simonyan 和 Zisserman 提出，在“Very deep convolutional networks for large-scale image recognition”这篇论文中有介绍。VGG 模型结构简单有效，前几层仅使用 3×3 卷积核来增加网络深度，通过 max pooling（最大池化）依次减少每层的神经元数量，最后三层分别是 2 个有 4096 个神经元的全连接层和一个 softmax 层。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

图 5.Very deep convolutional networks for large-scale image recognition 的图表

“16”和“19”表示网络中的需要更新需要 weight（要学习的参数）的网络层数（图 5 中的列 D 和 E）,包括卷积层，全连接层，softmax 层.

Xception 是由 François Chollet 本人（Keras 维护者）提出的。Xception 是 Inception 架构的扩展，它用深度可分离的卷积代替了标准的 Inception 卷积。

2.2.3 技术

项目中会使用基于 TensorFlow 的 Keras。

是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。

Keras 是一个高层神经网络 API，Keras 由纯 Python 编写而成并基 Tensorflow、Theano 以及 CNTK 后端。通过 Keras 可以更方便地构建和使用神经网络。

2.3 基准模型

kaggle 排行榜前 10% 的 LogLoss 作为基准阈值进行对照，也就是在 Public Leaderboard 上的 LogLoss 要低于 0.06127。LogLoss 的理论原理推导见 1.3。

III. 方法

3.1 数据预处理

3.1.1

训练数据文件夹分类：首先通过 filter()将文件名分类，将 cat 与 dog 的训练数据分别软链接至不同的文件夹。

3.1.2

通过使用 ImageDataGenerator.flow_from_directory()进行预处理，它以文件夹路径为参数,生成经过数据提升/归一化后的数据。在此次应用中，通过设置 target_size 参数，不同尺寸的图像会变换到统一的尺寸，例如，使用 VGG19 模型预处理时，target_size 设置为（224,224），而 InceptionResNetV2 和 Xception 的 target_size 则为（299,299）。

此外，预训练模型中还有 preprocess_input 进行预处理。对于 VGG：函数会在把图片缩放到 244x244 后，RGB 分别减去其平均值。

对于 Inception 的训练预处理：

1. 对图片进行随机 crop, 使其与 bbox 的重叠部分大于 0.1，长宽比在 (0.75, 1.33) 之间，croped 之后的图片大小为原图的(0.05, 1.0)。

2.将 crop 之后的图片大小 resize 为 crop_height(224), crop_width(224)

3 将 crop 图片左右翻转（50% 的概率会翻转）

4 调整 crop 图片的亮度(32. / 255.)和饱和度(0.5, 1.5)

5 每个元素减去 0.5，再乘以 2.0

3.1.3

训练集中存在少量误标和异常图片，可通过使用预训练模型 ResNet50 和 Xception，筛选出预测与标记不一致的图片，然后再人工选择删除，以确保训练集的可靠性。通过试验，发现 Xception 的误检相对低一些，采用 top-5 或 top-10 的误检都较高；训练集中，猫检出的异常值图片，相对狗异常值图片更多。最后对猫采用 top-50，找出 99 张图片；对狗采用 top-20 进行训练，找出 49 张图片。部分图片如图 6，图 7。

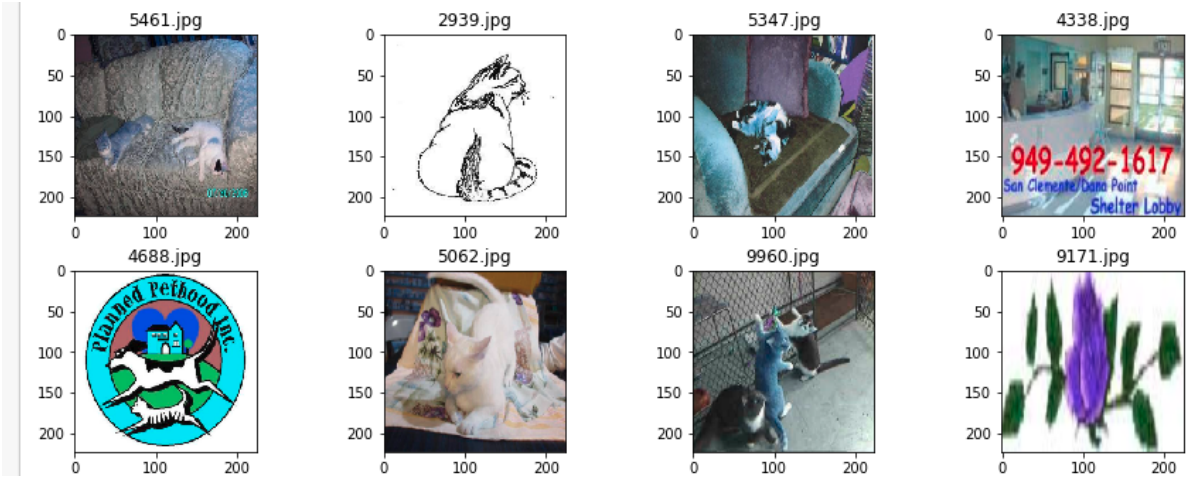


图 6. 关于猫的部分异常图

图 7. 关于狗的部分异常图

从可视化结果看，异常图大多是无关的文字和图片。最终人工选择去除４５张图片。

3.2 执行过程

3.2.1　总的解决思路如下，

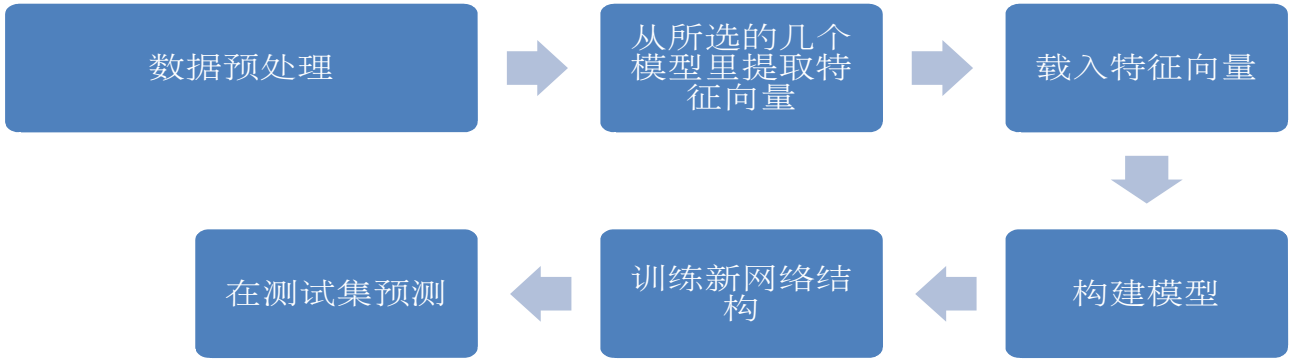


图 8. 流程示意

- 从几个模型提取特征向量：通过预训练的网络得到特征向量，并保存，以便在下面的模型构建中使用。导出的每个 h5 文件含 3 个数组。预训练的时间会使用 GPU 计算，大约１５分钟左右。不知为何每次挂起后再运行程序，GPU 就无法启动，需要重启电脑。

- 载入特征向量：把预训练导出的特征向量文件进行汇集，合成一条特征向量。进行 shuffle(X_train, y_train)处理，打乱 X, y，避免 validation_split 出问题。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        (None, 4096)              0
_________________________________________________________________
dropout_13 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 4097
=================================================================
```

图 9. 预训练模型融合后的数据处理

- 构建模型：如上图，将拼接的数据按以上网络层处理（Dropout 然后全连接层）。使用 model.fit()进行训练，这里设置验证集占２０%.

- 预测测试集

model.predict 进行预测，最后生成预测结果。在预测中，为避免 LogLoss 产生无穷大问题，所以使用 clip 函数，将预测值限制在[0.005, 0.995]．然后导出 csv 数据文件。

测试中的调整：

模型构建时要注意相关接口的维度问题，运行程序时多次因维度和数据属性问题卡住。同时 batch 太小可能会导致不收敛，适当加大 batch_size 会缩短时间。相比较而言，预训练耗时比模型融合后训练耗时要长。为尝试在预训练时，曾使用 batch_size=64 进行对比，预训练时间半小时；而 batch_size=16 时预训练在 25 分左右，观察 CPU 的占用相对低 30%左右。最终还是选择 batch_size=16。在融合模型后 fit 时，batch_size 使用 128 相比使用 64 的 validation_loss 小 0.001，准确率相差 0.0001；batch_size=128 时，本机运行速度会快一点，由每个 epoch 1s 缩短至 1s 内。

改进：

从起初的单模型预测，变为融合多个预训练模型权值，然后 Dropout.

为使用 callbacks 记录训练和验证时的 loss 与 accuracy，先建立一个 LossHistory 类，包含 on_train_begin, on_batch_end, on_epoch_end, loss_plot 等函数，用于训练后显示 loss 与 accuracy 曲线图像。同时也用 tensorboard 做图。

IV. 结果

模型的评价与验证，合理性分析

图 10.kaggle 评分

得到的 *kaggle* 评分结果为 0.03696，比删除训练集异常图片的结果略优，满足低于 0.06127 的要求值。验证集的准确率最高达到 99.6%．通过对比 kaggle 评分，模型较好地完成了图像分类，能准确区分猫、狗。

V. 项目结论

结果可视化

图 11 中 train acc/loss 指训练所得准确率/损失值，val acc/loss 指验证准确率/损失值，与 3.1 构建模型的训练结果一致。

下图是用 tensorboard 得到的训练 loss 图像。

图 12. tensorboard 显示

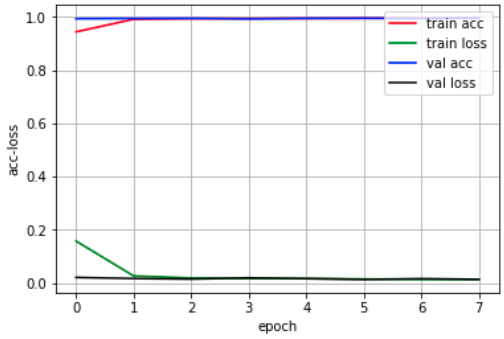
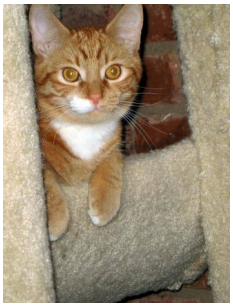


图 11. 训练与验证时的准确率、损失值

以测试集中的 15.jpg 和图片 12.jpg 为例，对图片中间层进行可视化，如图 13：



15.jpg

12.jpg

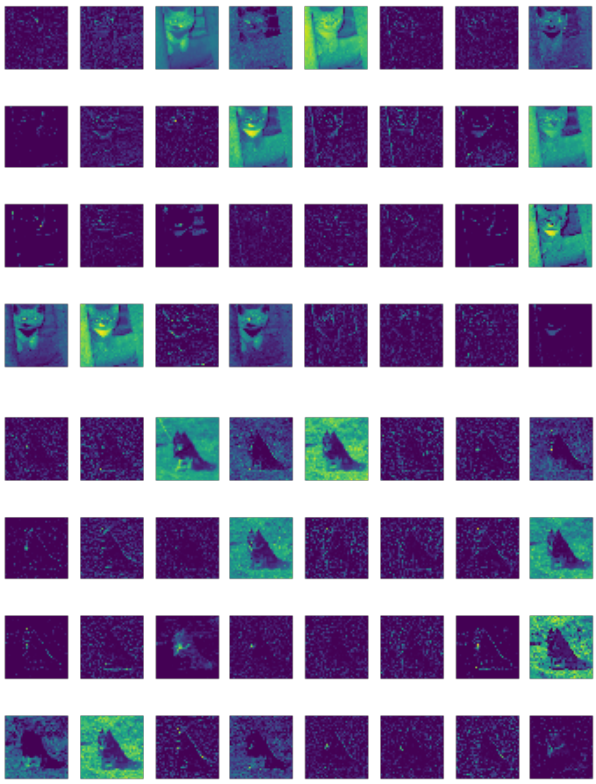


图 13. 中间层可视化

	id	label
0	1	0.995000
1	2	0.995000
2	3	0.995000
3	4	0.995000
4	5	0.005000
5	6	0.005000
6	7	0.005000
7	8	0.005000
8	9	0.005000
9	10	0.005000
10	11	0.005000
11	12	0.995000
12	13	0.005000
13	14	0.005000
14	15	0.005000
15	16	0.005000
16	17	0.993256

图 14. 部分预测结果

模型对这两张图片的分类结果，可以在输出的预测概率中查看（图 14）。概率值是预测为狗的概率，所以数值>0.5 视为狗。可见图片 12（id 值）的预测值 0.995，分类为狗；图片 15 的预测值 0.005，分类为猫。分类正确。

对于随机抽取的图片，模型分类效果也挺棒，如下图

图 15. 随机图片分类效果

对项目的思考

回顾整个项目，主要的思路是运用了预训练模型，既避免了重复造轮子，又提高了分类的效果。项目串联了数据预处理，keras 搭建网络，数据可视化等方法。

项目里软链接的应用，logloss 预测值的限制等小技巧是处理细节的一些收获。

可作出的改进

*1.*利用 val_loss 和 loss 值调整 drop-out rate，进行试验。Drop-out rate 是否可以调整得更合适，可以通过 val_loss 和 loss 加以确认。选取 0.2, 0.5, 0.8 三个数值作为 drop-out rate 进行比较，如下表：

Drop-out rate	loss	accuracy	val_loss	val_accuracy	loss-val_loss
0.2	0.0099	0.9974	0.0153	0.9954	-0.0054
0.5	0.0147	0.9959	0.0139	0.9964	0.0008
0.8	0.023	0.9937	0.0146	0.996	0.0084

表 3. Drop-out 效果对照

可见，准确率都达到 99%，随着 drop-out rate 减小，loss 下降且 accuracy 有所增加。val_loss 呈 U 形变化，在 drop-out rate 取 0.2 时会有所增大。并且取 0.2 时，val_loss 和 loss 的差值的绝对值变大，有过拟合趋势。相对而言，取 drop-out rate=0.5 较合理。

*2.*关于 fine-tune：通过对 optimizer 的选择和比较，发现使用 SGD 时学习率 0.0001 的表现不及 0.001，最终 val_loss 相差 0.01 左右；而 SGD 的表现不及 Adadelta，后者的学习率不必设为固定值（固定值效果反而不好）。

*3.*在提高代码运行效率方面可以改进：例如在图片异常值查找过程中，一些判断分类的循环语句代码可以做成函数形式加以调用。程序可以加以分块细化，提高可读性和效率。另外也试验了 ResNet，效果一般。可选择更好的预训练模型进行融合。

引用

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

[2]K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[3]François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In ICLR, 2017.

[4]Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In ICLR, 2016.