

机器学习纳米学位

毕业项目

马王涛

2018 年 4 月 17 日

## I. 问题的定义

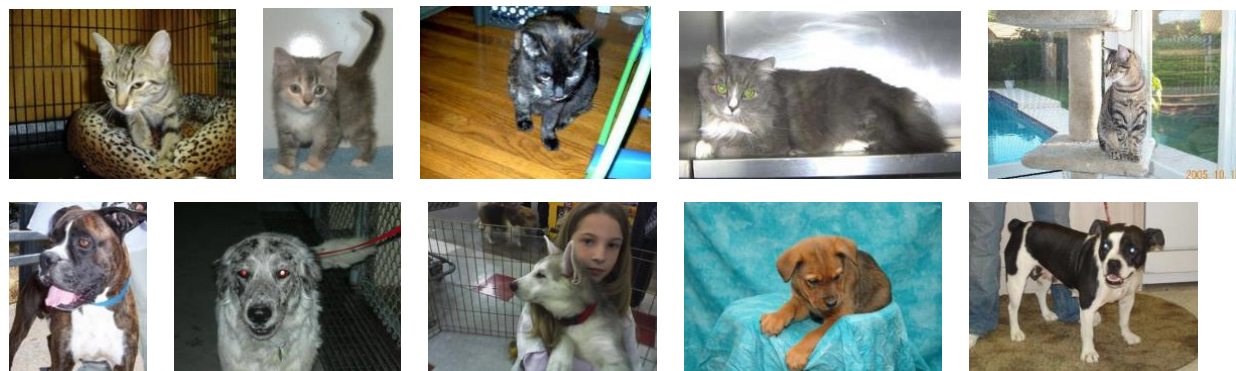
### 项目概述

计算机视觉是一个跨学科研究领域，研究的问题是如何使计算机具备对数码图片或视频的高阶理解能力[1]。计算机视觉可以进一步分为多个细分领域，如物体识别、身份验证等。近年来，计算机硬件性能大幅提升，标签数据集的数据量越来越大（如 ImageNet Large Scale Visual Recognition Challenge 比赛中由 ImageNet 提供的包含超过 20,000 个物体分类的大型数据集），以及深度学习的发展，使得计算机识别物体的准确率越来越高。与此同时，在多年 ILSVRC 中积累的模型也为公众提供了预训练模型，使得公众可以通过迁移学习来构建自己所需的模型。本次项目为“猫狗大战”，是一个物体识别项目。

### 问题陈述

本次项目中要解决的问题是使计算机能够根据输入的图片自动准确的识别图片中的动物为猫还是狗，输出的是判别输入图片为狗的概率。如上所述，该问题可以通过迁移学习来解决，即选取历年 ILSVRC 中具有不错表现的模型作为预训练模型，在此基础上构建最后两层（即池化层和损失函数层）并针对这两层进行训练以使得该模型能够更适应于识别猫狗。本质上，这次项目要解决的是一个二分类问题，因此可以使用 Log Loss 来作为衡量模型性能的标准；与此同时，也可以通过识别的准确率来衡量模型性能。

#### 原始数据集图片示例



### 评价指标

本次项目用于衡量和对比模型性能的指标为 Log Loss。Log Loss 具体计算方式如图 1，其中  $y$  代表被识别图片为猫还是狗，若是狗，则  $y=1$ ； $p$  代表  $y=1$  的概率，取值范围为 0

至 1。相对于准确率，该指标可以更精准衡量模型的性能，即“距离正确结果有多远”，而准确率只能表明预测是“对或错”。举例来说，现有两个模型 A 和 B，模型 A 预测图片中的动物为狗的概率为 0.75，模型 B 预测的概率为 0.55，作为一个二分类问题，概率超过 0.5 的预测类别即为最终的预测结果，以准确率来衡量，这两个模型是同样好；然而，若以 Log Loss 来衡量，模型 A 预测结果对应的 Log Loss 为 0.28768（即  $p=0.75, y=1$ ），模型 B 为 0.59784；显然，模型 A 的预测结果更接近真实情况。模型的 Log Loss 为所有样本 Log Loss 的算术平均值，该指标的数值越低，代表模型的预测性能越好。

图 1 LogLoss 计算公式

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

其中：

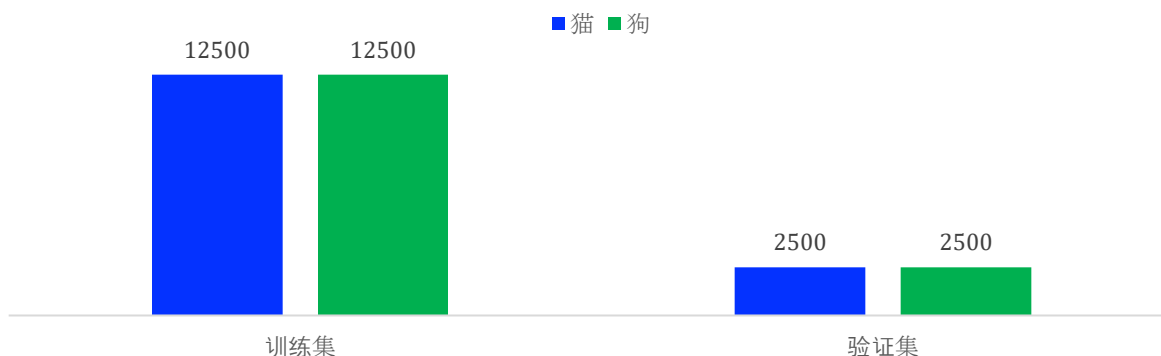
- $n$ ：代表样本总量或一个 batch 中的样本量；
- $y_i$ ：为实际标签，本次项目中数值为 1，代表该图片实际为狗；
- $\hat{y}_i$ ：代表预测图片为狗的概率。

## II. 分析

### 数据的探索

本次项目的数据集来自 Kaggle 上的竞赛“Dogs vs. Cats Redux: Kernels Edition”[2]。训练数据中包含 25,000 张图片，猫与狗分别为 12,500 张图片，每张图片的文件名中包含了标签（即该图片中的动物为猫还是狗）；测试数据包含 12,500 张图片，每张图片用数字序号命名，我们会预测每张图片中的动物为狗或猫的概率（狗=1，猫=0）。项目中，图片经过数列化处理以后将作为预测模型的输入，每张原始图片都可以转换为一个（224，224，3）的 numpy array，模型最终的输出为一个代表概率的数值（即在 0 到 1 之间的数值）。

训练集及验证集数据分布



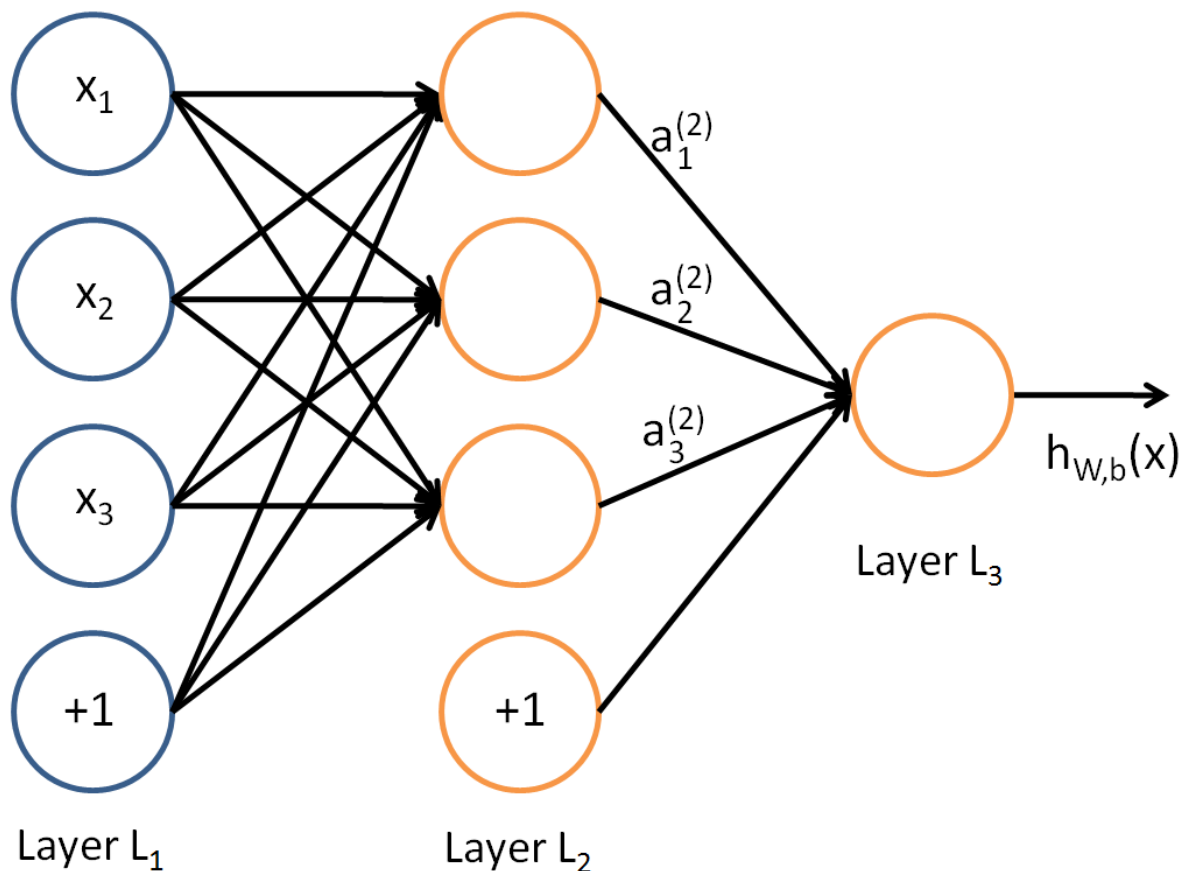
## 算法和技术

### 神经网络

神经网络是一种模仿生物神经网络结构和功能的模型，由大量的人工神经元联结进行计算。大多数情况下，人工神经网络能在外界信息的基础上改变内部结构，是一种自适应系统，也是一种非线性模型。神经网络中最基本的单元是单个神经元，每一个神经元都从其他单元获取输入值（即输入向量），求得输入向量与权向量的内积后，经一个非线性传递函数得到一个标量结果（即一个实数）。实际上，单个神经元的作用就是把一个  $n$  维向量空间用一个超平面分区成两部分（称之为判断边界），给定一个输入向量，神经元可以判断出这个向量位于超平面的哪一边。

多个（或有限个）神经元就可以构成单层神经元网络，多个单层神经元网络就可以构成多层神经元网络。一种常见的多层结构的前馈网络（**multiplayer feedforward network**）由输入层（**input layer**，如图 2 最左侧一层）、输出层（**output layer**，如图 2 最右侧一层）及隐藏层（**hidden layer**，如图 2 中间一层）组成。隐藏层可以有 multiple 层，每个隐藏层的节点数量可以任意设置，通常节点数越多，隐藏层数越多，模型的非线性越显著，但也需要注意，当神经网络隐藏层数过多或节点数量过多时，很容易导致模型过拟合或泛化能力较差，即仅在训练集上有良好的预测性能，而在新数据集上的表现大打折扣。当然，除了多层结构的前馈网络，神经网络中还有递归神经网络（即 **recurrent neural network**，简称 **RNN**）和卷积神经网络（即 **Convolution Neural Network**，简称 **CNN**）等其它类型，它们的传递方式较前馈网络更为复杂。例如，递归神经网络中的每一层不仅使用上一层的输出作为自己的输入，同时也会将状态在自身网络中循环传递，使得模型可以接受更广泛的时间序列结构输入。

图 2 多层结构的前馈网络示例



图片来自: <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>

## 卷积神经网络

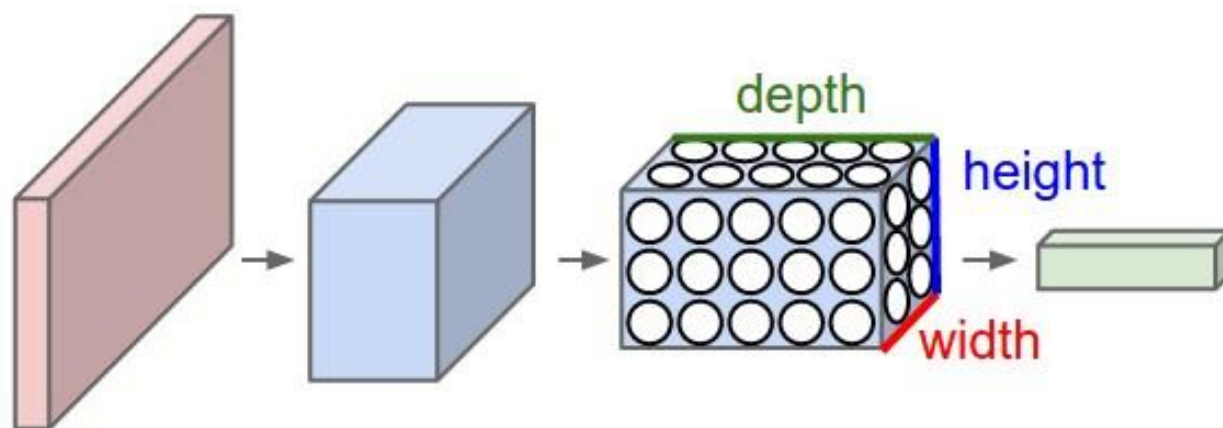
卷积神经网络与之前提到的前馈网络比较相似，也是由可学习调整权重值的神经元组成；每个神经元接受输入，计算出与权重值的内积，然后可根据需要添加非线性层（比如较为常用的 **rectified linear unit**，简称 **ReLU**）；卷积神经网络是一个可微函数，输入为数列化的原始图像，输出为各分类的得分（即概率）；与一般的神经网络一样，卷积神经网络在最后一个全连接层也有损失函数，例如 **Softmax** 或 **SVM**。

一般的神经网络并不适合用来做图像分类，因为图片中的像素点较多，就会导致神经网络中的参数过多。举例来说，对于一张像素为（200，200，3）的图片来说，一般的神经网络中一个神经元需要用到的权重值数量为 120,000 个；如此之多的权重值，不仅需要耗费大量的时间训练，还可能导致过拟合问题。

相较于一般的神经网络，卷积神经网络的优势在于合理利用了图像这种输入的特点，将权重值分布在三个维度上，即宽（**width**）、高（**height**）和深（**depth**）。举例来说，同样对于一张像素为（200，200，3）的图片来说，每一层的神经元只会与上一层中一部分神经元有连接，而不是上一层的所有神经元，这样这个神经元的权重值数量就可以大幅减少；假设最终的分类数为 10，则输出层会输出一个（1，1，10）的向量，其中每一个数值代表属于该分类的概率。

本次项目使用的就是卷积神经网络，与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。尽管与全连接网络相比，卷积神经网络中包含的参数量少了很多，但是总量依然巨大，例如 Xception 包含的参数量超过 2,000 万个[3]；考虑到本次项目可以用于训练的数据量较为有限，从零开始构造并训练未必能够训练出性能较好的模型。相反，采用 ILSVRC 中创建的表现较好的模型作为预训练模型，在此基础上增加全局平均池化层和损失函数层并针对这两层进行训练，使得该模型能够更适应于识别猫狗，倒是更好的解决方案。

图 3 卷积神经网络示例



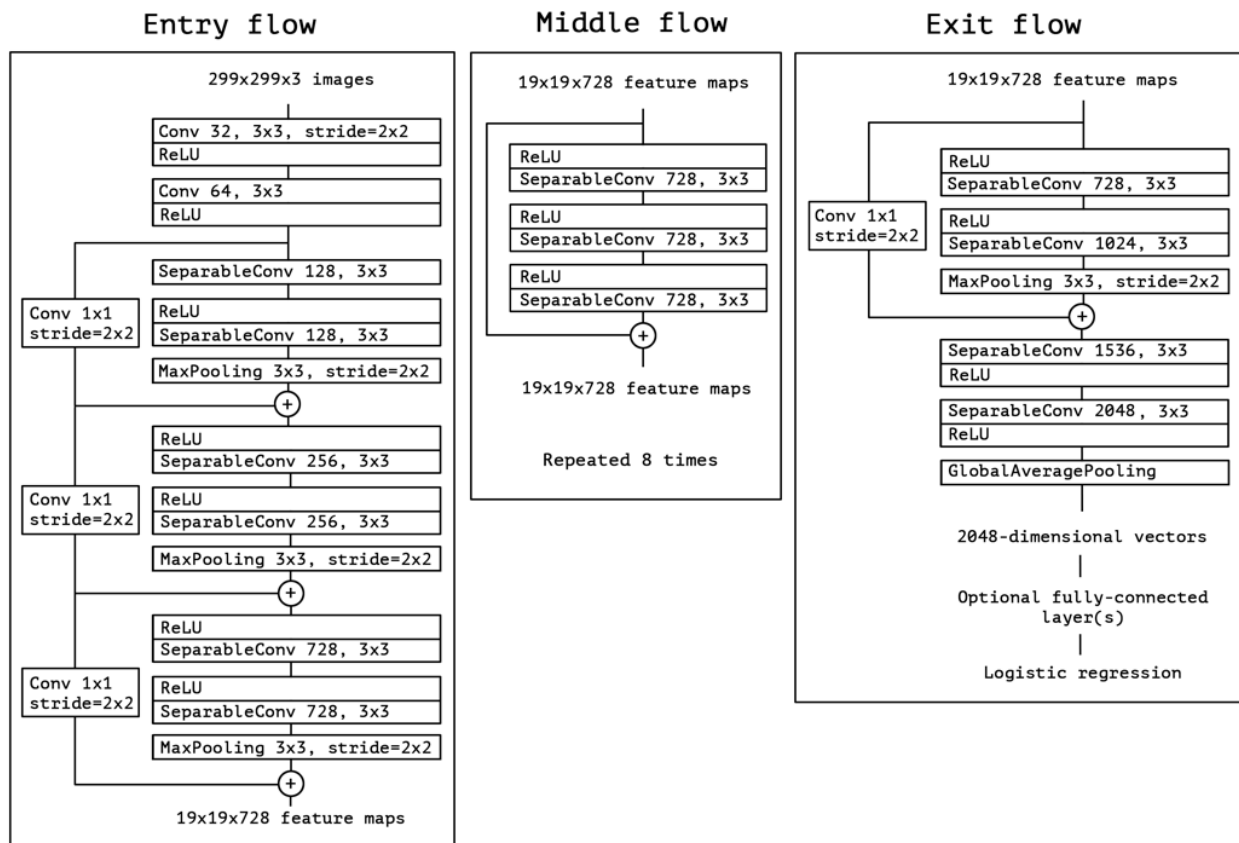
图片来自: <http://cs231n.github.io/convolutional-networks/>

### Xception 结构简介

本次项目拟采用 Xception 作为预训练模型。Xception 模型是一种类 Inception（Inception-type）模型，因此也具有对计算资源要求较低的优点。举例来说，Xception 仅使用了 2,000 万个参数[3]，而 VGG 使用的参数量为 1.8 亿个[5]，是其 9 倍之多。采用 Xception 架构的重要假设是图层间的相关性与各图层上的空间相关性在卷积神经网络的特征构造中是可以完全分离的。Xception 架构的另一个优点是容易修改和调整，因为它只是可分离卷积层（separable convolution layer）的线性叠加，其原始架构如下图所示。[3]

图 4 Xception 架构图





数据由 **entry flow** 输入模型，然后重复 8 次通过 **middle flow**，最终进入 **exit flow**。数据通过所有的卷积层和可分离卷积层之后都会经过归一化处理。所有可分离卷积层均不做延展。

## 基准模型

卷积神经网络的发展过程中创建了许多模型，从早期的 AlexNet 到 VGG 再到 Xception；总体而言，这些模型的性能（以准确率和损失来衡量）逐步提高。本次项目拟采用 Xception 作为预训练模型来创建一个更适合识别猫狗的模型，采用 Kaggle 比赛排行榜中 10% 百分位的 Log Loss 为基准（0.06127）[4]；换言之，本次创建的模型的预测结果的 Log Loss 应低于该基准。

## III. 方法

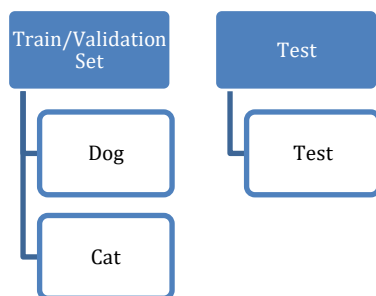
### 数据预处理

数据预处理。本次项目的数据集实为图片，无论在训练还是预测时，若将所有数据同时加载到内存中，会导致内存不足的问题。因此，需要使用生成器（即 **Generator**）来确保更有效的利用计算机的内存资源，生成器的优点在于它会记住上一次返回时在函数体中的位置，对生成器函数的第二次（或第  $n$  次）调用跳转至该函数中间并保持上次调用的所有局部变量都不变。具体来说，我们会用到 Keras 中 `ImageDataGenerator` 的方法

`flow_from_directory`，此种方法的另一个优点是可以直接读取文件夹中的图片，将数据分批及做其他预处理（如将图片的尺寸调整为预训练模型输入所需要的尺寸）。在使用 `ImageDataGenerator` 构建生成器时，我们设置的参数包括以下几个：

- `batch_size`：每一批数据的数量（即图片数量）为 32；
- `target_size`：Xception 的输入尺寸为（299，299），故预处理后图片的尺寸也需要为（299，299）；
- `class_mode`：识别的物体仅为猫和狗这两类，故此项参数应为“binary”；
- `shuffle`：此项参数在为训练集及验证集构建生成器时应为“True”，将训练所用的数据打乱，以获取更好的训练效果。此外，`ImageDataGenerator` 的方法 `flow_from_directory` 会根据所读取的文件夹的结构自动标示数据，因此我们需要将原始训练数据集先分为训练集（相当于原始数据集 80% 的数据量）及验证集并分别放到两个文件夹中，然后在两个数据集的文件夹中进一步创建子目录分别用于放置猫和狗的图片。对于测试集，虽然无需将图片分类存放，但是根据 `flow_from_directory` 的要求，需要在测试集数据所在的文件夹中再创建一个子目录。三个数据集的最终结构如图 5。

图 5 数据集文件夹结构



数据预处理中的另一个重要的环节是剔除异常值；换句话说，就是要把非猫狗的图片从训练集中剔除掉，以减少噪音，使得模型能够更好的学习。由于本次项目的输入数据为图片，很难使用传统的统计方法来剔除异常值，比如剔除处于四分位距 1.5 倍以上位置的数据点。一种比较高效的方法是在 ImageNet 数据集上训练出来的预训练模型来筛选异常值，因为 ImageNet 数据集包含了猫狗和其它物品种类。具体来说，预训练模型最后的输出是输入图片为 ImageNet 数据集中各分类的概率，并且最终可以根据一个叫 Top-n 的数值来判断模型是否准确预测了图片中物体的种类。业界通常使用的是 Top-1 和 Top-5 准确率；Top-1 准确率的意思是，我们只看模型预测出的概率最高的种类，这个种类是否与图片的正确分类是否一致，若一致则表示预测准确；同样的，Top-5 准确率的意思是，我们只看模型预测的概率最高的五个种类，如果这五个种类中包含了图片的正确分类，则表示预测准确。

我们计划选用 ResNet50、InceptionResNetV2 及 DenseNet 作为筛选异常值的模型，分为以下几个步骤：

1. 导入模型；

2. 分别对三个模型的输入做数据预处理；
3. 对数据进行预测；
4. 根据预测数据，设置 Top-n 参数，筛选出异常值（例如将 Top-n 设置为 45，通过 ResNet50 筛选出的异常值总共有 35 个）；
5. 将三个预训练模型筛选出的异常值合并，然后全部从训练集中删除。

经过筛选以后，总共得到 47 个不同的异常值，部分示例如下。可以看到，有些图片被至少两个模型一致认为非猫非狗的图片，有些图片则是只被一个模型认定为异常值。总体而言，筛选的效果不错，绝大部分都是异常值；极少图片中包含了猫狗，有些事猫狗在图片中所占的比例过低，有些是猫狗的颜色或形态比较“奇怪”，较难辨别。

### ResNet50 筛选出的部分异常值（总共有 35 个）

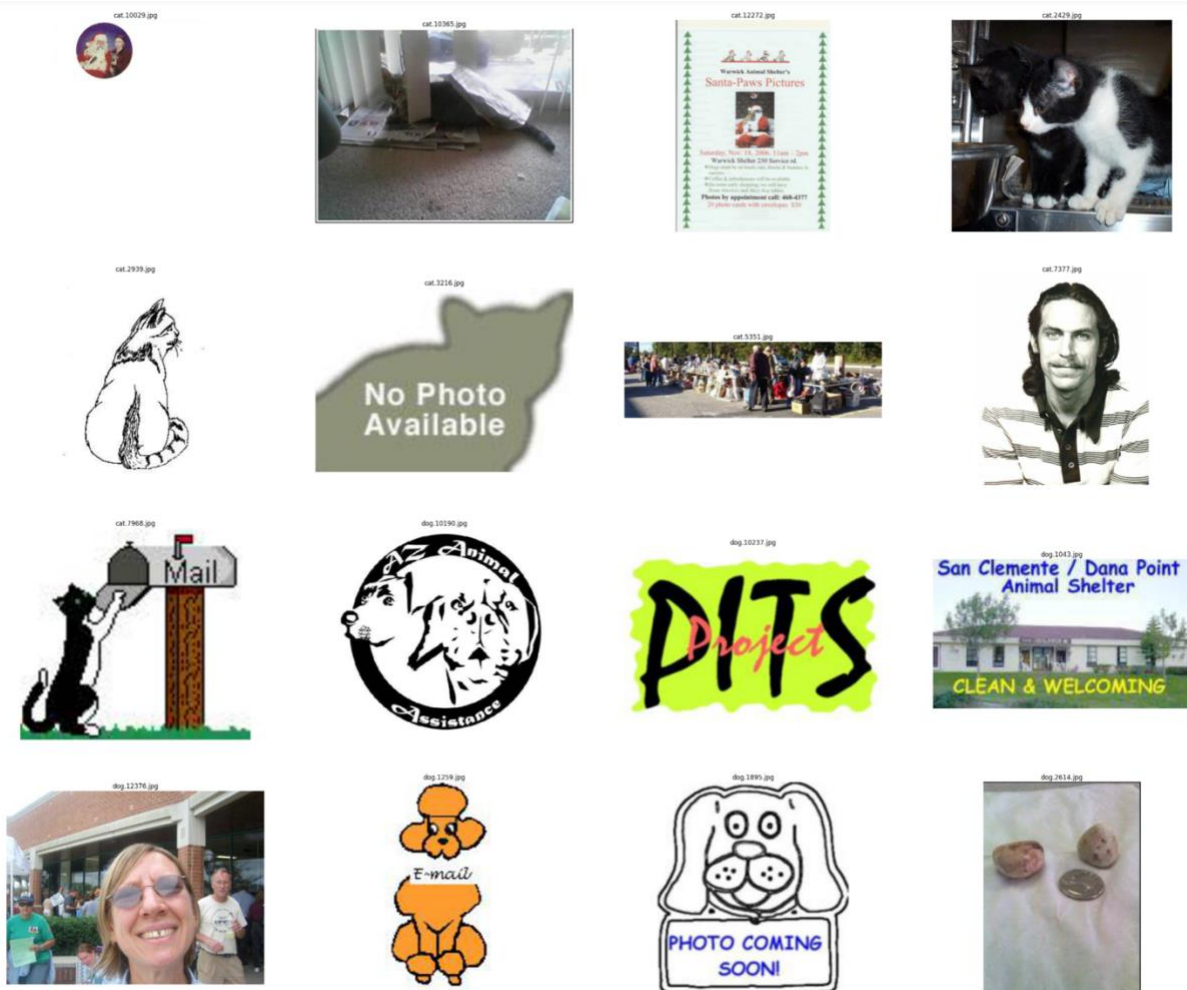


### InceptionResNetV2 筛选出的部分异常值（总共有 23 个）





DenseNet 筛选出的部分异常值（总共有 25 个）



## 执行过程

构建模型。本次项目使用的预训练模型，选取的模型为 Xception。采用预训练模型实为该模型使用 ImageNet 数据训练所得的模型及参数，然后在此基础上增加池化层和损失函数层，以适应本次的预测需求。池化层使用的是全局平均池化层（即 Global Average Pooling），池化层尺寸为（2，2），步长为（2，2）；损失函数层使用的激活函数为“Sigmoid”；为了防止模型过拟合，池化层与损失函数层中间添加 Dropout，dropout rate 初始设置为 0.5。模型构建完成以后，即可使用训练集和验证集来训练模型。

图 6 以预训练模型为基础构建新模型代码

```

#定义一个以预训练模型为基础构建新模型的函数
def build_model(model, img_size, layer_to_train=None, dropout, preprocess=None):

    """
    model: 选择的预训练模型
    img_size: tuple, 预训练模型要求的图像输入尺寸,如(299,299)
    layer_to_train: integer, 设置预训练模型中可训练的层数 (即倒数几层)
    dropout: float, 随机舍弃率, 通常为0.5~0.7
    preprocess: 设置预训练模型相应的预处理方式
    """

    #设置输入数据格式为 (width, height, channel)
    input_tensor = Input((img_size[0], img_size[1], channels))
    x = input_tensor

    #若需要, 对数据进行预处理
    if preprocess:
        x = Lambda(preprocess)(x)

    #加载预训练模型
    conv_base = model(weights='imagenet', include_top=False, input_tensor=x)

    #设置预训练模型中可训练的层数 (即倒数几层)
    if layer_to_train:
        for layer in conv_base.layers[::-layer_to_train]:
            layer.trainable = False
        for layer in conv_base.layers[-layer_to_train:]:
            layer.trainable = True
    else:
        for layer in conv_base.layers:
            layer.trainable = False

    #增加顶层构建模型
    x = GlobalAveragePooling2D()(conv_base.output) #在与训练模型上增加池化层
    x = Dropout(dropout)(x) #增加dropout层以防止过拟合
    x = Dense(1, activation='sigmoid')(x) #增加激活层, 激活函数使用sigmoid
    model = Model(conv_base.input, x)

    #函数返回一个模型
    return model

```

预测及后期数据处理。将测试集数据输入模型，但同样需要分批预测，即使用 `predict_generator` 方法，预测结果代表每张图片中的动物被认为是狗的概率。由于这个概率数值可能是 0 至 1 之间的任何数值，有些为“极小数值”（即非常接近 0，如 0.005），有些可能为“极大数值”（即非常接近 1，如 0.99998），这些“极小数值”和“极大数值”在预测样本时与 0 和 1 几乎没有差别，但是由于模型性能是以 Log Loss 来衡量，当样本预测错误时，这些“极小数值”会大幅增高 Log Loss，例如当一张狗的图片被预测为只有 0.0049 的概率为狗时，这个样本的 Log Loss 为 5.3185，但若这一概率数值为 0.005，相应的 Log Loss 为 5.2983，虽然两者在预测结果上（即都应认定该图片为猫）没有任何差别，但前者的 Log Loss 更高。为此，我们需要对最终的预测数据进行一些处理，使得预测结果中小于 0.005 的数据都变换为 0.005，高于 0.995 的数据也都变换为 0.995，这样可以减少一些“不必要的”Log Loss。

图 7 预测并处理预测结果代码

```

#模型预测数据并做后续clip处理
def predict(model, generator):
    pred = model.predict_generator(generator, verbose=1)
    pred = pred.clip(min=0.005, max=0.995)
    return pred

```

完善

本次项目的训练集数据量较小，且采用了预训练模型做迁移学习，只适合对部分层进行训练。因此，可训练层的数量及随机舍弃率成为了决定模型表现的关键因素。本次项目中，我们最初选择放开预训练模型的最后三层，使用的随机舍弃率为 0.5；在调试模型的过程中，我们逐步增加可训练层的数量至最后 6 层、16 层、26 层，同时使用不同的随机舍弃率（ $p=0.5$ 、 $p=0.6$ 、 $p=0.7$ ）；根据模型结构，预训练模型中的最后 3 层为第 14 个模块中第 2 个可分离卷积层，最后 6 层为整个第 14 个模块，最后 16 层包括了第 13 个和第 14 个模块，最后 26 层包括了第 12、13 和 14 个模块。在研究过程中，我们完整记录了这 12 个模型在训练集、验证集及测试集上的表现并进行对比（如表 1），以观察可训练层数量对于模型性能的影响。从表中可以看到，无论是放开多少可训练层和采用何种随机舍弃率，最佳权值模型在测试集上的表现几乎总是优于最终权值模型。

表 1 不同数量可训练层及不同随机舍弃率模型的表现对比

		3 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最终权值	99.65%	0.01182	99.42%	0.02074	0.04467
p=0.5	最佳权值	98.45%	0.04648	99.52%	0.01677	0.04374
p=0.6	最终权值	99.65%	0.01091	99.38%	0.02473	0.05009
p=0.6	最佳权值	99.27%	0.02337	99.38%	0.02084	0.04679
p=0.7	最终权值	99.43%	0.01761	99.30%	0.02613	0.04899
p=0.7	最佳权值	98.18%	0.05400	99.32%	0.02125	0.04844

		6 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最终权值	99.96%	0.00281	99.32%	0.02671	0.04923
p=0.5	最佳权值	99.67%	0.01114	99.30%	0.01870	0.045
p=0.6	最终权值	99.91%	0.00269	99.24%	0.02661	0.05379
p=0.6	最佳权值	98.62%	0.03825	99.40%	0.01483	0.04296
p=0.7	最终权值	99.96%	0.00123	99.34%	0.03321	0.05162
p=0.7	最佳权值	99.85%	0.00423	99.58%	0.01455	0.04205

		16 个可训练层				
--	--	----------	--	--	--	--

随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最终权值	99.97%	0.00091	99.50%	0.01745	0.04461
p=0.5	最佳权值	99.45%	0.01613	99.56%	0.01242	0.03993
p=0.6	最终权值	99.98%	0.00174	99.30%	0.03150	0.04794
p=0.6	最佳权值	99.79%	0.00623	99.50%	0.01238	0.03896
p=0.7	最终权值	99.96%	0.00121	99.46%	0.01715	0.0412
p=0.7	最佳权值	99.72%	0.00861	99.54%	0.01181	0.04065

		26 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最终权值	99.96%	0.00162	99.40%	0.02043	0.04418
p=0.5	最佳权值	98.65%	0.03780	99.38%	0.01456	0.04489
p=0.6	最终权值	99.96%	0.00169	99.38%	0.03371	0.05169
p=0.6	最佳权值	99.66%	0.01271	99.50%	0.01058	0.03868
p=0.7	最终权值	99.94%	0.00207	99.56%	0.02108	0.04498
p=0.7	最佳权值	98.64%	0.03820	99.60%	0.01098	0.03958

除了使用随机舍弃率来防止过拟合，我们同时也采用了早期停止（early stopping）来防止这个问题。当模型经历了指定代数（epoch）的训练以后在表现上没有任何提高，早期停止规则就会强制模型停止训练，以防止过拟合问题出现。本次研究中，我们设置的停止代数（patience）为 5，即当模型在连续 5 代训练后在表现上（以验证集损失为准）没有任何提高，模型就会停止训练。从表 2 中可以看到，本次研究中的 Xception 模型比较容易过拟合，各参数条件下，很少有训练代数达到 10 代的（每个模型至多训练 10 代）。

表 2 各模型的训练代数

随机舍弃率	3 个可训练层	6 个可训练层	16 个可训练层	26 个可训练层
p=0.5	6	8	7	6
p=0.6	8	6	9	7
p=0.7	6	10	8	6



IV. 结果<sup>1</sup>

模型的评价与验证

本次研究中，最佳模型使用了 26 层可训练层和 0.6 的随机舍弃率，该模型表现符合预期。无论是以准确率还是损失作为衡量标准，最佳模型在训练集、验证集及测试集上的表现基本优于其它模型，具体数据如表 3；同时，最佳模型在训练集、验证集和测试集上的表现较为一致也表明模型的性能受到输入变化的影响非常小。

从训练的过程来看，无论是最佳模型还是其它模型，大部分都在较早的代数就已经达到了最优状态（如表 4）；最优状态之后的代数中，虽然模型在训练集上的表现在不断提高，但是在验证集上的准确率（或损失）并没有累积提高（或降低），这意味着模型已经过拟合了，此处仅以最佳模型的训练过程为例做展示，如图 8。

表 3 最佳模型与其它模型在训练集、验证集及测试集上的表现对比

		3 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最佳权值	98.45%	0.04648	99.52%	0.01677	0.04374
p=0.6	最佳权值	99.27%	0.02337	99.38%	0.02084	0.04679
p=0.7	最佳权值	98.18%	0.05400	99.32%	0.02125	0.04844

		6 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最佳权值	99.67%	0.01114	99.30%	0.01870	0.045
p=0.6	最佳权值	98.62%	0.03825	99.40%	0.01483	0.04296
p=0.7	最佳权值	99.85%	0.00423	99.58%	0.01455	0.04205

		16 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失

<sup>1</sup>本次研究中分别使用了未剔除异常值和剔除异常值以后的训练集分别训练了 12 对模型，每对模型的参数相同，仅是使用的训练集不同，训练过程没有任何差别。根据训练结果，以测试集损失来衡量，最佳模型为使用未剔除异常值的训练集训练所得，故此处模型的结果评价仅使用未剔除异常值训练集的训练结果。至于剔除异常值以后的训练效果反而有所下降的原因，会在本报告“需要作出的改进”做进一步分析。

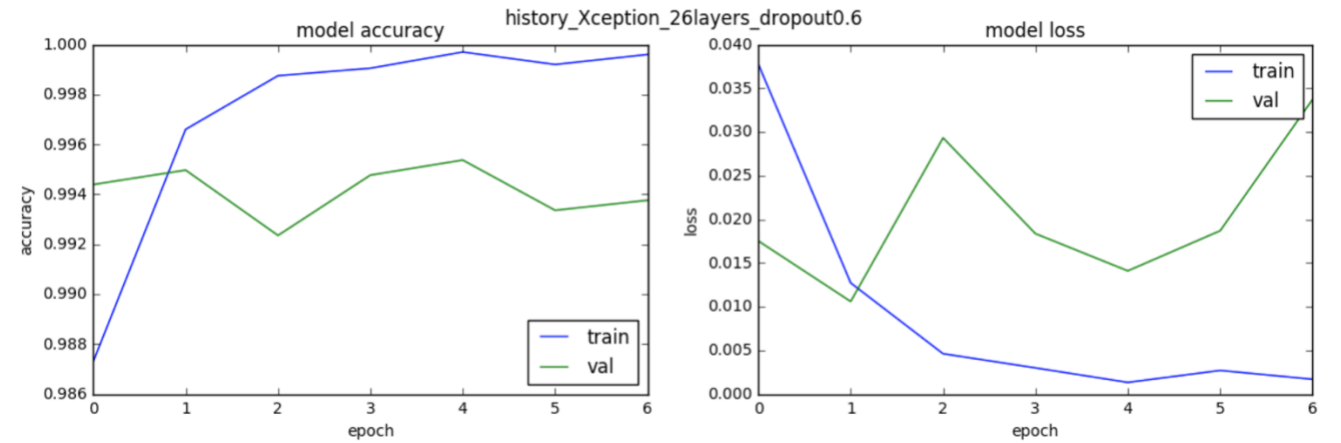
p=0.5	最佳权值	99.45%	0.01613	99.56%	0.01242	0.03993
p=0.6	最佳权值	99.79%	0.00623	99.50%	0.01238	0.03896
p=0.7	最佳权值	99.72%	0.00861	99.54%	0.01181	0.04065

		26 个可训练层				
随机舍弃率	权值	训练集准确率	训练集损失	验证集准确率	验证集损失	测试集损失
p=0.5	最佳权值	98.65%	0.03780	99.38%	0.01456	0.04489
p=0.6	最佳权值	99.66%	0.01271	99.50%	0.01058	0.03868
p=0.7	最佳权值	98.64%	0.03820	99.60%	0.01098	0.03958

表 4 各模型达到最佳权值的代数

随机舍弃率	3 个可训练层	6 个可训练层	16 个可训练层	26 个可训练层
p=0.5	1	3	2	1
p=0.6	3	1	4	2
p=0.7	1	5	3	1

图 8 最佳模型训练过程

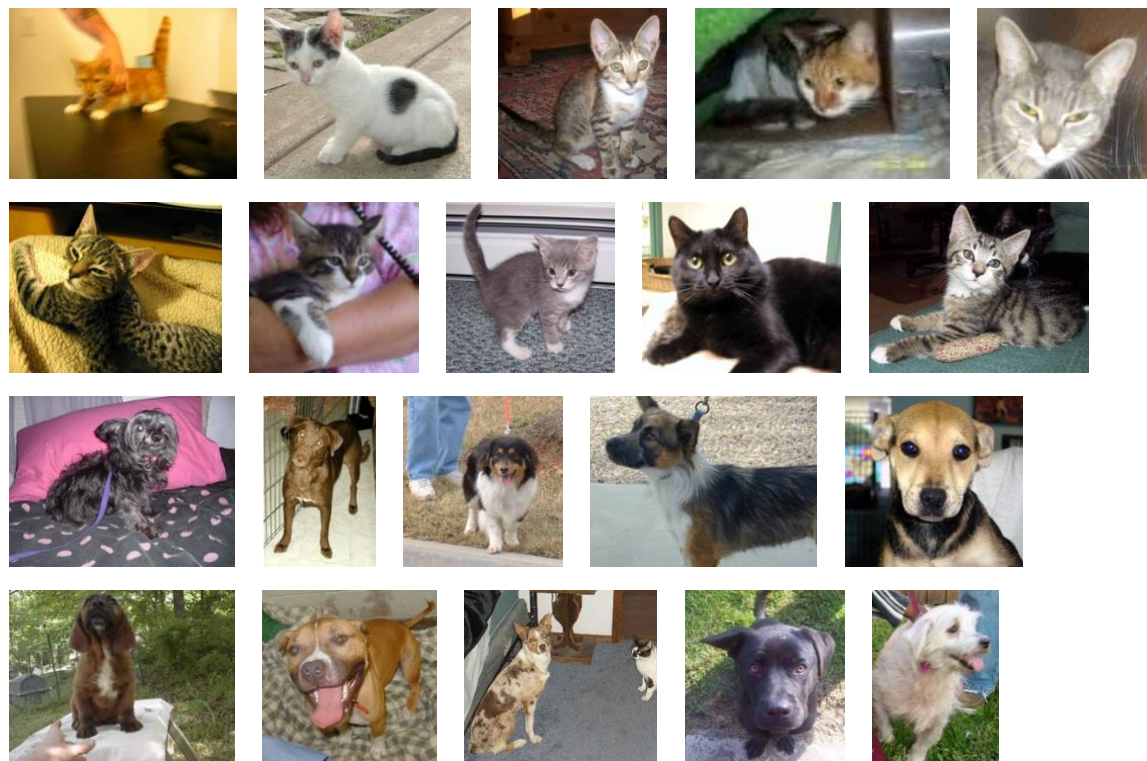


合理性分析

本次研究中，最佳模型使用了 26 层可训练层和 0.6 的随机舍弃率，最终在测试集上的损失为 0.03868，相当于比赛排行榜中 1%百分位的排名，远优于基准模型的 0.06127。由

于缺少测试集的标签，我们就以最佳模型在验证集上的表现来做分析，下面分别列出了最佳模型在验证集上准确和错误预测的图片。






验证集中准确预测的图片示例

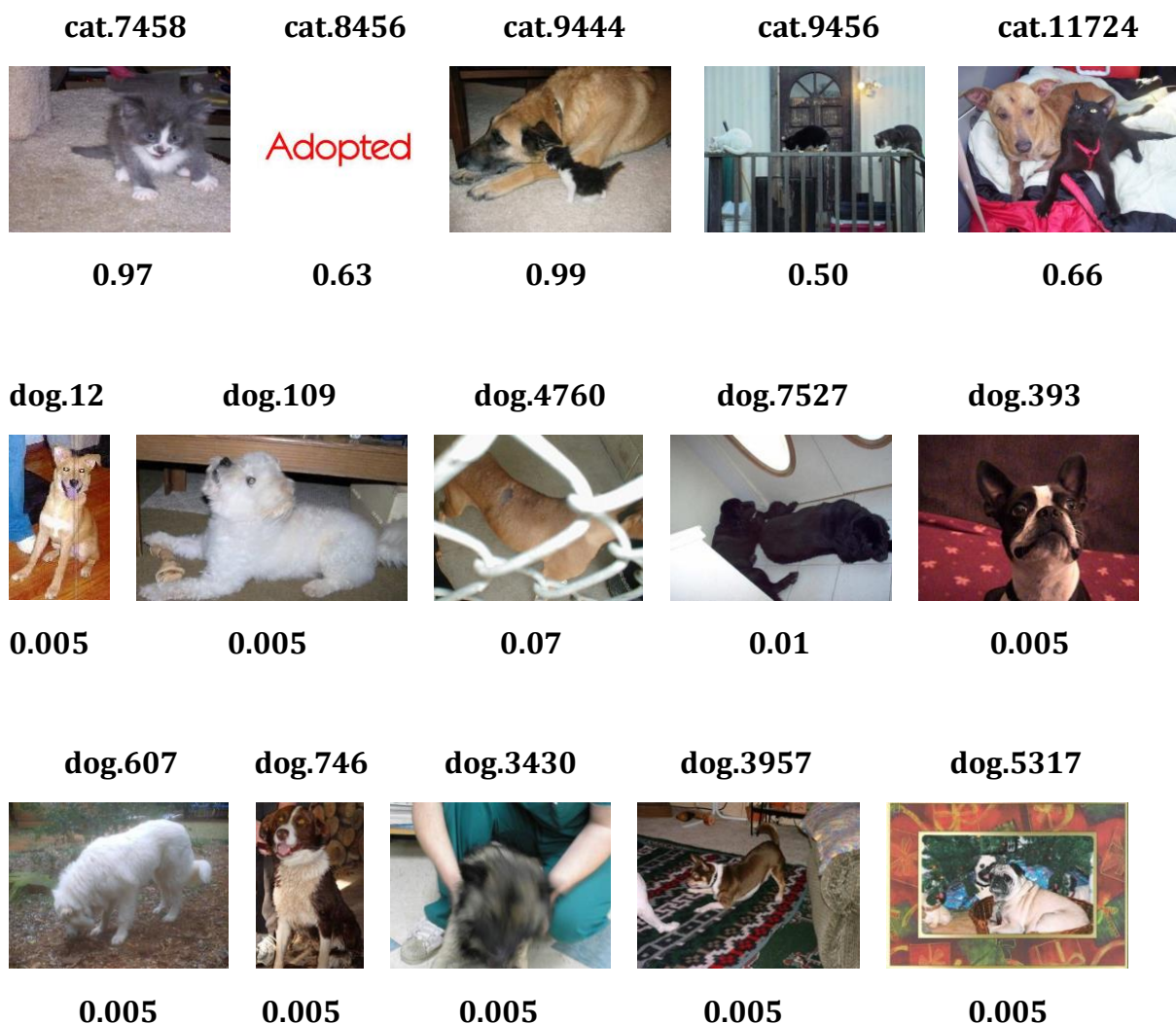


从准确预测的图片来看，模型能够识别不同颜色和处于不同背景下的猫和狗，比如图片第二行从左至右的第二张图片和第三行从左至右的第三张图片中，模型分别可以识别被抱在怀里的猫和在草地上牵着的狗。即使有些较为模糊的图片，比如第一行从左至右的第一张图片中的猫，模型依然能够准确识别出图片中的动物。

验证集中错误预测的图片示例

(图片下方数值为图片被预测为狗的概率)

<b>cat.1575</b>	<b>cat.2663</b>	<b>cat.3300</b>	<b>cat.6655</b>	<b>cat.11184</b>
				
<b>0.29</b>	<b>0.84</b>	<b>0.74</b>	<b>0.70</b>	<b>0.57</b>



从错误预测的图片来看，模型犯错的原因不尽相同。图片 cat.9444 和图片 cat.11724 中，有猫和狗两种动物，可能是因为狗在画面中占得面积较大，导致模型将该图片标识为狗，而非猫；图片 cat.11184 其实为异常值（即既不是猫也不是狗），模型确认为它有 56% 的可能性为猫，这可能与图片中的颜色为灰色有关系。有一个有趣的现象是，模型似乎更容易在识别狗的时候犯错，在验证集中，有 69 张被标识为狗的图片（包括 3 个异常值）都被认为猫，而只有约 12 张被标识为猫的图片（包括 2 个异常值）被标识为狗。

## V. 项目结论

### 对项目的思考

整个项目实施的过程主要为这几步：选定预训练模型、数据预处理、在预训练模型基础上构建新模型、训练模型、预测数据处理、调整参数以获取最优模型。整个过程中，最为困难但也是最有意思的部分为调整参数这个环节。在这个环节中，我们可以发现模型的表现并不是受某个单一因素（比如可训练层数或随机舍弃率）的影响，也不是与某一因素呈简



单的线性关系，比如放开的可训练层越多或随机舍弃率越高，预测性能越好；相反，它是需要调整多个参数才能达到最优的状态。造成这种现象的一种可能的原因是从“深度”这个维度上（即放开更多的可训练层或模块）让模型更多的学习有助于模型变得更为适合识别猫狗，但由于放开更多可训练层更容易导致过拟合，就需要提高随机舍弃率。举个简单的例子，假设有一个 10 层的神经网络，每一层上有 100 个神经元；当我们选择放开可训练层数分别为最后 5 层和最后 4 层，两种训练方法分别对应的随机舍弃率为 0.6 和 0.5；虽然两种方法（单次）实际训练的神经元总数均为 200 个，但是第一种训练方法得到的模型在“深度”上有更好的适应性。当然，这只是基于“神经网络越深性能通常越好”这种假设的推测，严谨的论证还需要更多的研究工作来支持。

本次研究得出的最佳模型在验证集上的识别准确率达到了 99.50%，也就是说在 1000 张图片中只有 5 张会被错误识别；若考虑到模型的预测速度，这应该已经可以至少达到甚至超过人类的预测准确率。本次研究所采用的迁移学习方法，同样也可以应用到 ImageNet 数据集中包含的其它种类物体的识别任务中。

需要作出的改进

本次模型虽然已有不错的表现，但是依然有一些可以改进提高的地方。在使用单个模型的情况下，首先，可以考虑放开更多模块或可训练层来验证这种方法是否有助于提高模型的表现（即降低测试集损失），因为根据目前的研究结果，放开更多的模块或可训练层（并辅以相应的其它参数，比如随机舍弃率）可以使得模型拥有更好的表现（见表 5）；其次，可以考虑调整其它的参数，比如正则化参数、学习率。另外一种较为有效的提高模型表现的技术是集合多个预训练模型输出的特征向量，然后输入到激活层和输出层，这种方法背后的思想有些类似于 Bagging（即取多个模型预测值的平均值作为输出[6]），合并后的特征向量能够更好的概括图像的特征，从而拥有更好的泛化能力。

表 5 各模型最佳权值对比

随机舍弃率	3 个可训练层		
	验证集准确率	验证集损失	测试集损失
p=0.5	99.52%	0.01677	0.04374
p=0.6	99.38%	0.02084	0.04679
p=0.7	99.32%	0.02125	0.04844

随机舍弃率	6 个可训练层		
	验证集准确率	验证集损失	测试集损失
p=0.5	99.30%	0.01870	0.045
p=0.6	99.40%	0.01483	0.04296
p=0.7	99.58%	0.01455	0.04205



16 个可训练层			
随机舍弃率	验证集准确率	验证集损失	测试集损失
p=0.5	99.56%	0.01242	0.03993
p=0.6	99.50%	0.01238	0.03896
p=0.7	99.54%	0.01181	0.04065

26 个可训练层			
随机舍弃率	验证集准确率	验证集损失	测试集损失
p=0.5	99.38%	0.01456	0.04489
p=0.6	99.50%	0.01058	0.03868
p=0.7	99.60%	0.01098	0.03958

最后，本次研究中，我们同时使用了未剔除异常值和剔除异常值以后的训练集来训练模型，但发现剔除异常值以后的训练效果并没有明显优于未剔除异常值训练所得的模型（如表 6 所示）。可能的原因是在使用预训练模型筛选异常值时剔除了一些可能不该删去的样本，比如 InceptionResNetV2 删去了一些与背景颜色过于相近的猫，但动物颜色与背景色相近无论是在现实还是测试集中都是很有可能出现的情况，剔除这些样本可能使得模型的泛化能力有所下降。因此，就本次项目而言，另一项可以作出的改进是在剔除异常值时更为仔细谨慎，可以使用预训练模型先做一轮筛选，从筛选结果中人工选择哪些样本是应该删除，哪些是应该保留的。

表 6 剔除与未剔除异常值训练集所得模型对比

		最佳权值模型测试集损失	
可训练层数量	随机舍弃率	未剔除异常值	剔除异常值
3	0.5	0.04374	0.04901
3	0.6	0.04679	0.04909
3	0.7	0.04844	0.04742
6	0.5	0.04500	0.04696
6	0.6	0.04296	0.04577
6	0.7	0.04205	0.04499
16	0.5	0.03993	0.03925
16	0.6	0.03896	0.03911
16	0.7	0.04065	0.03989
26	0.5	0.04489	0.03992
26	0.6	0.03868	0.04387
26	0.7	0.03958	0.03921

## 参考文献

- [1] [https://en.wikipedia.org/wiki/Computer\\_vision#Definition](https://en.wikipedia.org/wiki/Computer_vision#Definition)
- [2] <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data>
- [3] François Chollet, Xception: Deep Learning with Depthwise Separable Convolutions
- [4] <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/leaderboard>
- [5] Zbigniew Wojna, Rethinking the Inception Architecture for Computer Vision
- [6] [https://en.wikipedia.org/wiki/Bootstrap\\_aggregating](https://en.wikipedia.org/wiki/Bootstrap_aggregating)