《猫狗大战》项目报告

一、问题的定义

1.1 项目概述

近年来，深度学习技术的提出与发展，让人工智能领域再度成为了人们的焦点，各个方向百花齐放，其中“计算机视觉”也取得了不错的进展。如何利用好深度学习技术，自动学习和识别图像的特征，让机器具有人类的“视觉”也成了亿万学者追求的目标。

“猫狗大战“项目来自于 Kaggle 于 2013 年起举办的比赛，属于图像分类领域的热门话题，同时，深度学习的兴起也带动了该领域的长足发展。在深度学习之前，图像分类技术主要利用传统机器学习模型（如 SVM[1])来学习手工特征（如灰度直方图特征[2]、纹理特征[3]等），近年来，随着深度学习的兴起，图像分类领域也出现了各种各样的比赛，其中比较出名的如 ImageNet，从当初火热的 AlexNet[4]到后来的 VGGNet[5]、InceptionNet[6]到如今 SENet[7]等，在图像分类领域做出了卓越贡献。

1.2 问题陈述

“猫狗大战”项目意在能够根据深度学习方法自动对猫狗图片进行分类，要求根据给定的数据集（包含很多张不重复的猫和狗的图片）和对应的标签（猫或狗）来训练一个端到端的模型，使得模型能够自动识别出未知图片（测试集）是猫还是狗，即二分类问题。

解决此问题需要通过深度学习相关算法，根据很多猫狗的数据（训练集）作为训练，让模型的参数根据不同的图片进行调整，而调整后的好坏可以用数据中自带的类别标签作为参考，根据预测出来的类别和真实类别间的误差关系（损失函数）评价模型学习的效果，从而反馈给模型，进一步调整（反向传播）。

而模型方面，可以选择多种常用的深度学习模型（如 VGGNet、ResNet 等）。

1.3 评价指标

由于模型是一个二分类模型，因此常用的评价指标是 logloss 函数（二分类交叉熵），该函数能够反应预测的类别和真是类别之间的误差，并且 logloss 值越大，反应的误差越大，对于反向传播的过程而言调整参数力度就越大，非常契合二分类问题的模型训练。其中 logloss 计算方法如下：

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

其中，n 表示训练样本总数，y_i 代表真实标签，$\hat{y}_i$ 表示模型预测出来的标签（对于猫狗大战，表示预测为狗的概率）

二、分析

2.1 数据的探索

项目给定的数据集来自于 Kaggle 猫狗大战官方给定的数据集，其中训练集共 25000 张图片，测试集 12500 张图片；其中，猫狗比例为 1：1，各占一半。由于官方没有划分验证集，因此项目中将从训练集中随机划分 20%作为验证集，剩下的 80%构造为新的训练集。另外，训练集中图片的名称带有类别标签和索

引，而测试集中只带有图片索引。

另外，图片尺寸不一，这对于模型而言，绝大多数模型是固定输入尺寸的，因此在后面实验环节，需要将图片调整到同一尺寸。

最重要的，是检查数据是否异常。猫狗大战中，异常数据点显然就是既不是猫也不是狗的图片，对于此，很容易想到既然可以用 ImageNet 预训练的模型进行迁移学习到猫狗分类任务，那么自然地，我们也可以使用这些预训练模型对训练集进行测试，对预测的 top-n 个标签中，如果既没有 ImageNet 1000 类中的若干类猫，也没有其若干类的狗，那么就能判断该图片是否为异常。

2.2 探索性可视化

1. 样本数量分布：根据图 1 可以得出，训练集中猫狗数量是一样的，均为 12500 张；而由于测试集未给出标签，但从官方给出的信息可以得到，测试集猫狗数量也是均衡分布的。

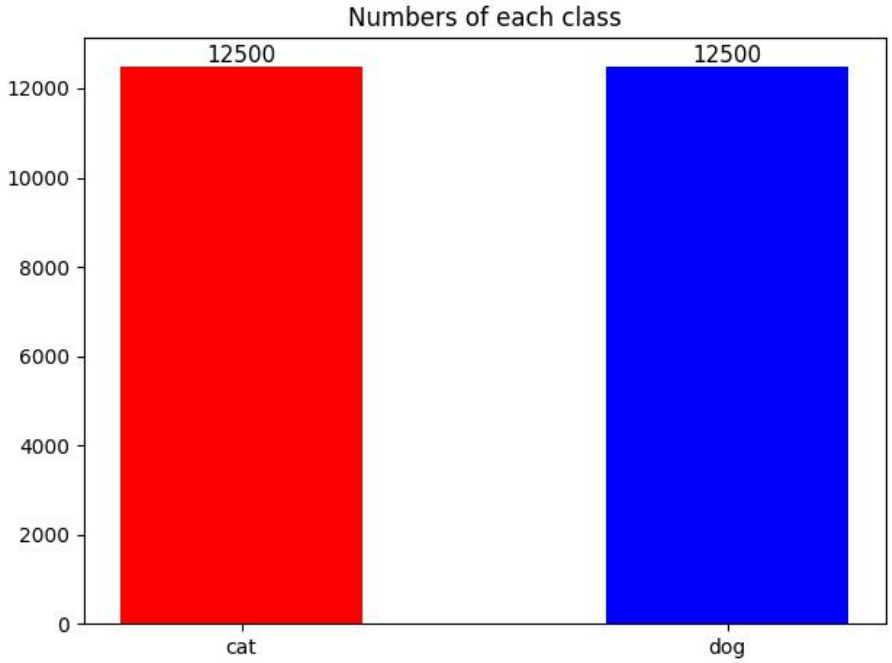


图 1.训练集中猫狗图片数量

2. 样本尺寸分布：根据图 2 可用看出，训练集中各种图片大小不一，大多数样尺寸不超过 500*500，但有两张图片尤其大：cat.835.jpg，dog.2317.jpg。因此若在模型输入之前不将它们尺寸调整到同一水平，系统将会非常复杂，且不一定能有很好的学习效果。图 3 为测试集图像大小的分布，同样，分布不均匀。

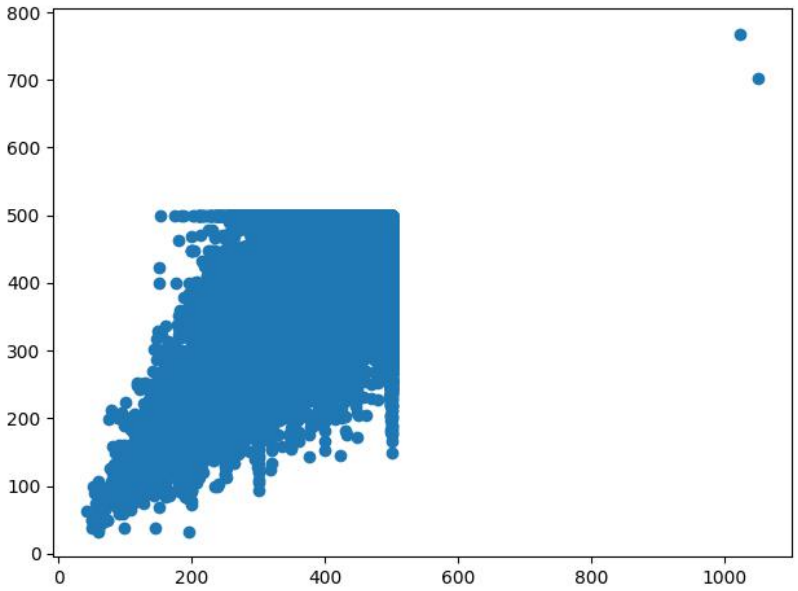


图 2.训练集中样本尺寸分布

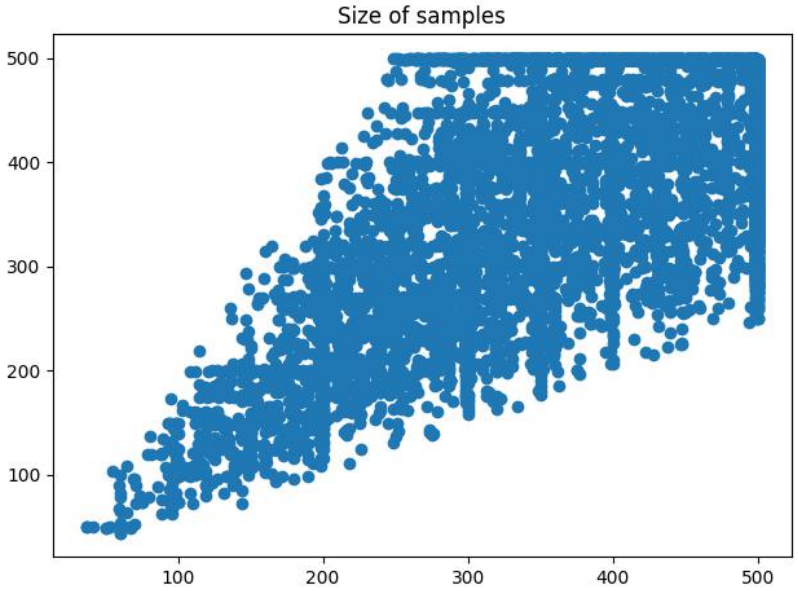
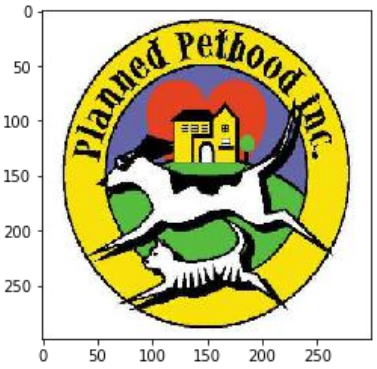
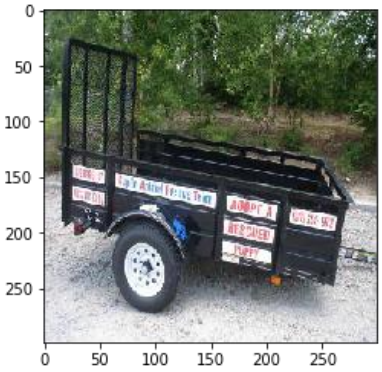


图 3.训练集中样本尺寸分布

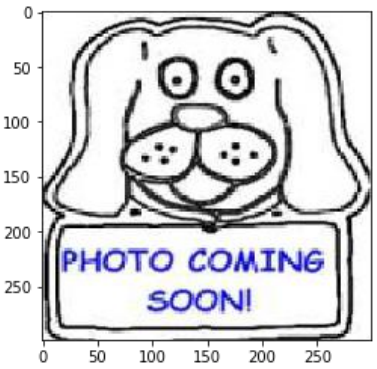
3. 异常值的删选：上节提到异常数据对于系统而言是非常致命的，就像人类儿时如果学到的东西是不正确的，那么对新事物的预测也会出现偏差。根据 ImageNet 上预训练模型 Xception[9]进行预测，最终删选出 52 张图片。其中 6 张为：



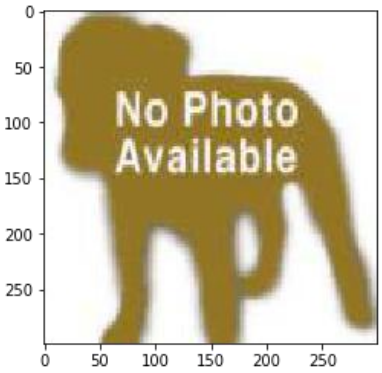
cat.4688.png



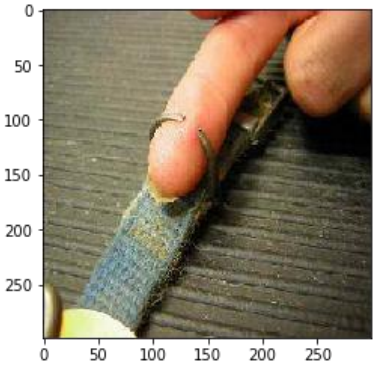
dog1194.png



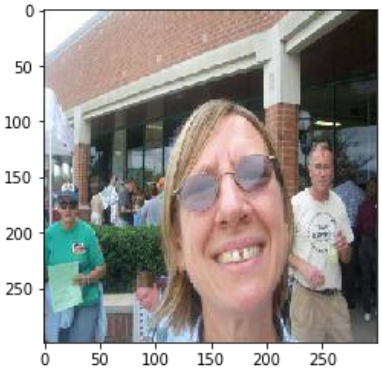
dog.1895.png



dog.8898.png



dog.10801.png



dog.12376.png

很明显，以上几张删选出来的图片并不是说图片不够明确是否为猫狗，而是

完全不是猫和狗的照片，即使出现手绘风格的狗的照片，但由于它们与正常狗的照片相比，完全不包含狗的特征，因此即使人可以认为手绘风格的狗也算狗，但模型学不到狗的潜在特征，因此也算为异常。

2.3 算法和技术

对整个项目的算法和技术从几个部分进行讨论：

1. 读取数据环节：采用 Python3.5 的 PIL 库下的 Image 类进行读取图片，并对图片做初步的处理（如 resize 等）；

2. 预处理环节：数据读取出来以后，是初始状态的像素（0-255），不利于模型的学习，因此预处理环节就显得尤为重要。项目预处理函数有 keras 框架提供，对应于不同模型，方便使用对应模型的预处理函数。同时，异常值检测也是该部分的核心内容。

3. 模型相关：搭建一个适合学习猫狗数据库的模型，首选 Keras 框架，其一，该框架能在调用 GPU 同时，极大缩短了代码量，相比于 Tensorflow 而言，搭建的模型更具有可读懂性，且较为容易学习。此外，Keras 最大的好处是封装了 ImageNet 优秀的预训练模型，可以很方便的调用它们进行迁移学习或者 fine-tuing。模型训练和测试环节也较为简单，只需要编译模型时指定一些重要参数，如损失函数，评价指标，优化器等。下面详细介绍下模型的选择及其重要算法。

由于项目中选择的几个模型均与卷积神经网络有关，因此模型的介绍将从卷积神经网络开始。首先，就深度学习模型而言，卷积神经网络的诞生和发展对于计算机视觉来说是一个质的飞跃，其中最核心的思想是卷积、池化层。从最初的

人工神经网络堆叠的全连接网络说起，全连接网络前后两层所有的神经元都必须与另一层每一个神经元进行连接，某种意义上能够很好地映射出原始输入的高纬特征，但是当输入数据本身维度就很高时，就会导致整个模型参数非常臃肿，计算量大大增强，且容易过拟合。以比较小的图片（28,28）的彩色图片为例，第一层全连接层节点数就达到了 28*28*3 个神经元，当层数稍微多一两层时，参数数量很容易达到千万级。而卷积神经网络则能够极大降低参数的数量，以(28,28) 的灰度图片为例，介绍下卷积神经网络（图片来自于博客[10]）：

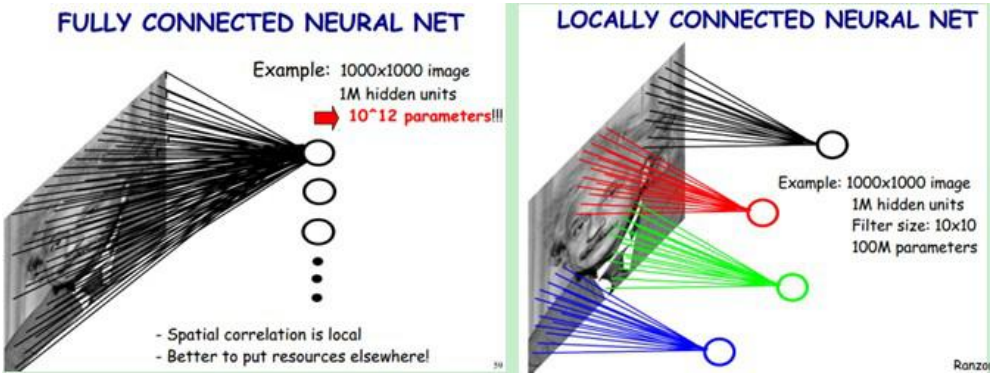


图 4.全连接网络和局部连接网络

卷积神经网络与普通神经网络的区别在于，卷积神经网络包含了一个由卷积层和池化层构成的特征抽取器，如上图（图)4）右侧所示，卷积层中，一个神经元只与上一层部分神经元连接。在一层卷积层当中，通常会有若干个参数矩阵，称之为卷积核，这些卷积核会被作为滑动窗口按 patch 提取图像特征，而每个 patch 又被称为特征图（feature map）。在一次卷积操作中，每个 feature map 是共享当前卷积核的权重的。一个卷积的卷积操作可以用下面的图 5，图 6 表示，

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	0	1
0	1	0
1	0	1

图 5.5*5 图片（左）及一个卷积核（右）

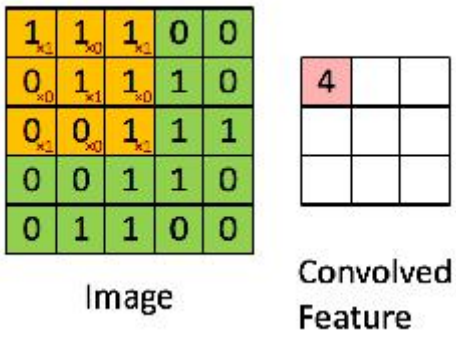
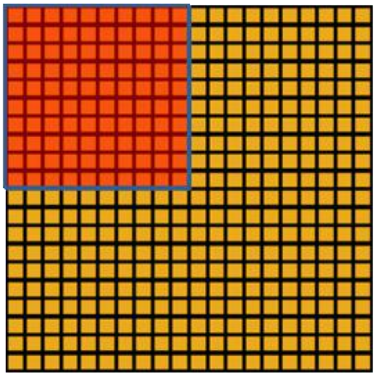


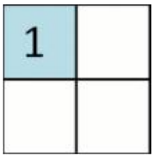
图 6 一次卷积操作

其中图 5 绿色图代表一个 5*5 的图片，数字代表像素，黄色图代表 3*3 的卷积核，利用这个卷积核做滑动窗口，在图像上计算每个特征图的值（像素与卷积核参数的乘积之和），该值即为当前特征图的卷积特征。而卷积神经网络还有一个重要操作是池化（pooling），池化分为最大池化和平均池化。池化操作如图 7 所示，对于卷积后的特征，为了进一步降低过拟合并减少参数，我们可以将卷积后的特征再分块，对每块取最大值即为最大池化，取平均值即为平均池化。通常情况下，卷积后的特征维度已经不高了并且能够保留的信息已经被大量减少，因此卷积后的一般选用最大池化已保留特征最明显的特征图提取的特征。基于以上两种操作，卷积神经网络能够极大减少模型的参数，降低过拟合。

鉴于此，项目选用一个带有很多层卷积池化层的神经网络模型 VGGNet19 作为第一个模型进行实验。VGGNet 提出时带有多种结构，如图 8 所示，我们选用的 VGG19 一共带有 16 个卷积层和 5 个最大池化层。其中，所有的激活函数均为 ReLU，该激活函数计算公式如下所示，相比于其它激活函数，它不存在梯度饱和问题，且计算简单快捷。

$$f(x) = max(0, x)$$





Convolved feature

Pooled feature

图 7.池化操作

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 8.VGGNet 模型结构

此外，项目选用 ResNet50 作为第二个网络，该网络是一个 50 层的网络，其最大的特点是特征的“跳跃传播”，如图 9 所示，当一个深层网络向下传播时，

原始输入的数据图像信息或多或少在传播过程中丢失了部分，因此为了弥补这种损失，可以将前面某层隐藏层特征跳过几步融合到另一层网络当中，这样有效降低了深层网络特征衰减的问题。

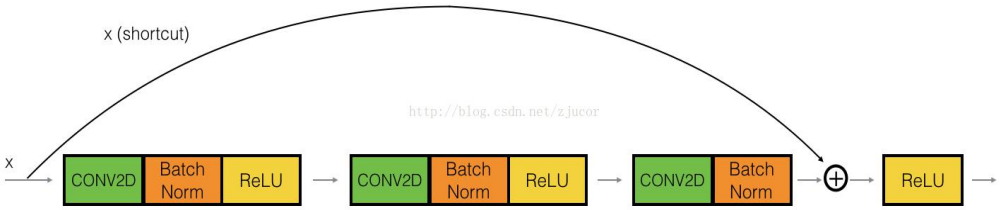


图 9.ResNet 部分结构

接着我们还选用了 Inception V3 网络作为第三个网络。Inception V3 网络基本思路是用多个卷积核对同一层进行卷积操作，并且再卷积后的特征上再做卷积，如图 10 所示，最底层是一个 3*3 卷积特征，在此之上再进行 1*3 卷积，在

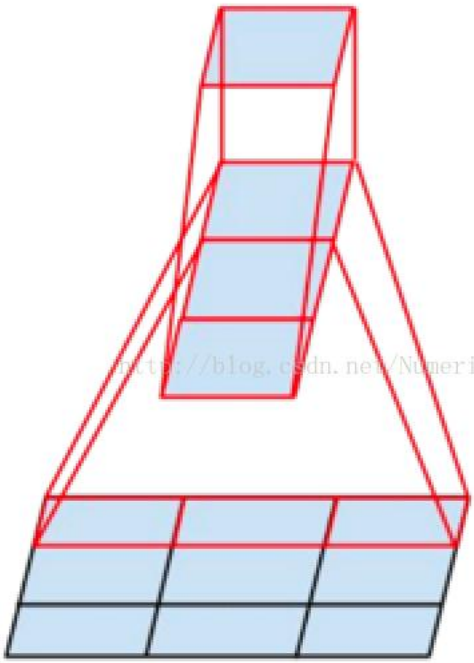


图 10.InceptionV3 卷积示意图

顶层做 1*1 的卷积。这样的操作旨在让特征图的大小缓慢下降，避免在靠前的层数中丢失大量信息，而同时又能避免大量参数传递到后层，并且能够让模型自主

学习每个 feature map 哪种卷积操作最好，一个完整的 InceptionV3 块如图 11 所示，而项目导入的 InceptionV3 带有 9 个这样的块。

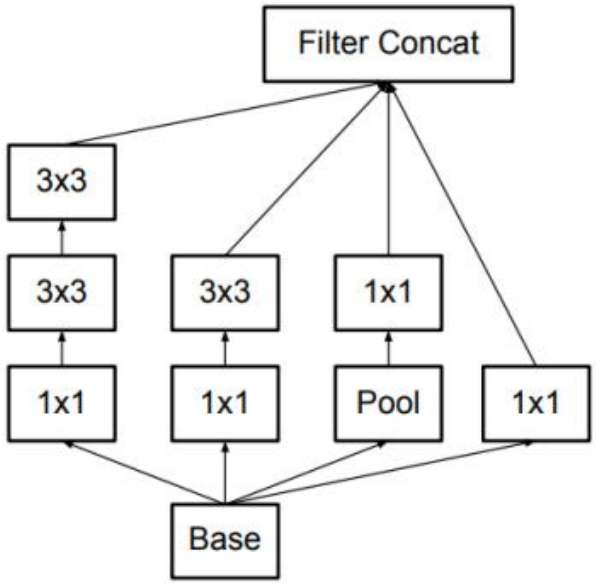


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

图 11 InceptionV3 部分结构

此外，项目还使用了 Xception 网络，Xception 网络是谷歌继 InceptionV3 后又提出的一种模型。在传统的卷积网络当中，卷基层会同时寻找跨空间和跨深度的相关性，而实际上这种操作是冗余的，每个通道空间的信息是能够完全独立开来的。例如图 12 所示，每个卷积核会对同一 feature map 下所有通道空间信息进行卷积汇总，额外增加了计算量的同时还混合了跨通道的信息。因此 Xception 的提出旨在将这种跨通道的的卷积方式改变，完全将每个通道空间独立开单独计算卷积，最后再将卷积后的信息汇总。这种卷积操作称之为可分卷积 (separable conv2d)，其主要结构如图 13 所示。

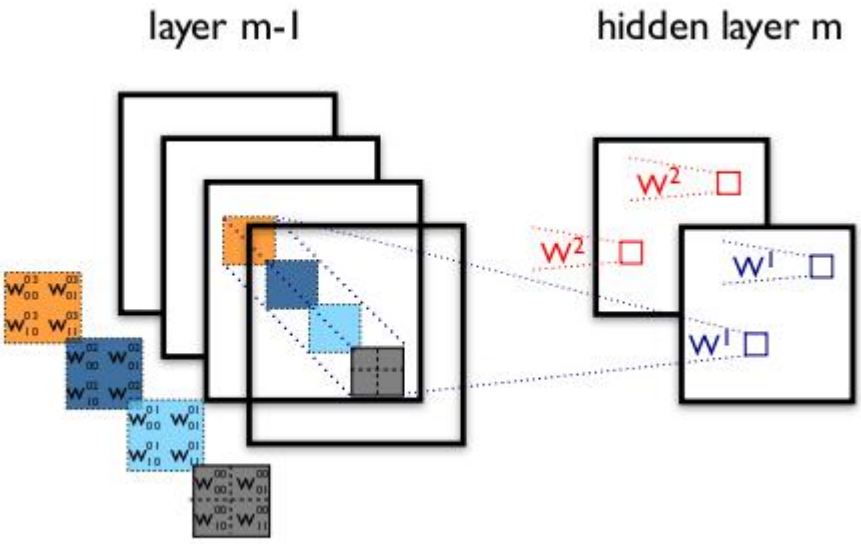


图 12.传统的卷积操作

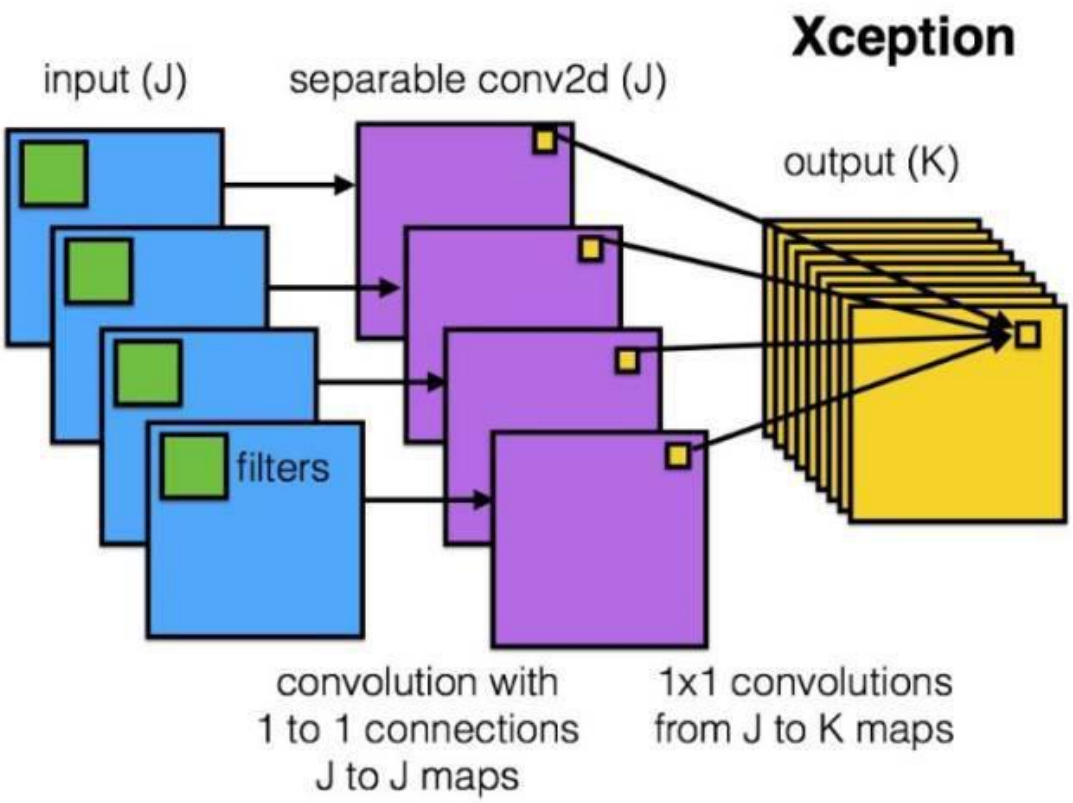


图 13.Xception 部分结构

最终，我们使用以上四种模型做迁移学习。由于这些模型是再 ImageNet 上

表现优异的，且训练的样本中本身就带有多种猫狗图片，这为将其迁移到猫狗分类任务上提供了理论依据。此外，使用预训练的模型，其参数已经在很大程度上进行了调整，梯度也已经减少到了一个等级，因此能够极大加快训练速度，更加容易收敛且鲁棒性更强。由于猫狗大战是二分类任务，而预训练的模型都是 1000 类，因此加载时，应当去掉 top 层（即后面几层全连接层），将最后一层的高纬特征进行全局平均池化（GlobalAveragePool）操作，再接入分类器。GlobalAveragePool 是将每个维度的特征值取平均，得到每个维度上的全局特征，极大减少了参数的数量从而降低了过拟合的效果。此外，对于对于全局特征的总结还有一种全局最大池化，即对于每个维度的特征矩阵取最大值，而对与我们使用的这几个模型，特征层的输出维度都非常大，如果直接取最大值将丢失很多重要信息。最后，为了防止过拟合，需要引入 Dropout 让每个神经元有一定的概率不激活，此概率的控制会被作为超参数进行多组实验以选择最佳。

4. 可视化：可视化环节需要绘制不同的图像，而通常需要绘制的信息并不能直接获取（比如存在对应表格数据），因此项目需要通过代码来获取数据的结构、信息。而 Python 中最流行的库就数 Matplotlib 了，该库能够非常方便且灵活的绘制我们所需的图片，如上一章提到的数据分布图和样本尺寸分布图等等，均是该库绘制的。

2.4 基准模型

实验选了历届 ImageNet 比赛中夺冠的几个模型作为迁移学习的基准模型，如 VGG19，ResNet50，Inception V3，Xception。这些模型都曾在 ImageNet 上大放异彩，对于 1000 类的图像分类都能取得很好的成绩，据此可知，对于猫

狗二分类的问题，也能胜任。

此外，Kaggle 中猫狗大战比赛上已经录用的前一百（10%）的成绩均可作为项目中模型学习好坏的参照。

三、方法

3.1 数据处理

实验中由于数据集要复用，将整个数据集操作相关封装成 Dataset 类。该类封装了以下方法：异常值删选、划分验证集、训练数据生成器、测试数据生成器。1）异常值删选：在加载数据过程中，首先创建 Dataset 对象，初始化时调用划分验证集的方法。该过程调用方法为：dt = Dataset()。异常值处理算法大致思路如下：

A) 定义 ImageNet 中各种猫狗的 ID，比如'n02085620'代表的是某种狗的类别。这些 ID 来自于博客[11].

B) 载入 keras 中 Xception 模型及其权重；

C) 读取图片至（299,299）尺寸，并用 Xception 对应的预训练函数进行预处理，并将其作为输入，利用 Xception 进行预测，并将预测的概率解码成类别 ID。

取预测结果的 TOP-x 为标准，如果预测出的 x 个标签中均没有猫和狗，那么可以视这张图片为异常图片。其中 x 取 30，实验中尝试过 10、30、50，发现取 10 时误分类的图片较多，而取 50 时，有些图片是异常图，但被分到狗里面，比如(dog1194.png)原本是辆三轮车，当 top 从 30 改为 50 时，它被认为不是异常。

D) 将异常图片名称从文件名列表中剔除，剩余的作为纯净的数据。

2）划分验证集：实验中验证集与训练集的比例称为 val_rate,val_rate 在模型间对比时，选择的是 0.2，在最终融合的模型中多了两组对比实验以确定当前数据集上最优比例。当 val rate 过大时，就会导致训练数据太少，模型训练不充分；当 val rate 过小时，val loss 就缺少了可靠性。

划分验证集算法大致思路如下：

A) 初始化验证集样本列表；

B) 打乱剔除异常值后的 filelist（它被作为 Dataset 对象的属性保存起来，便于调用），设定 random.seed 使得验证集在一次完整实验中固定，便于模型间对比和结果复现。

C) 取’train/’目录下 filelist 前 1-val_rate 作为训练集，后 val_rate 作为验证集。

D) 将验证集的 filelist 复制给 Dataset 对象的属性 self.val_data_file.

3）加载训练数据生成器：由于内存限制，若模型训练时采用 Model.fit(x,y)方法（该方法来自 keras 的模型，下同），x 变量无法存取足够多的训练数据，因此只能通过 Model.fit_generator(generator)来实现增量式的训练。其中，训练和验证集的生成器来自于 Dataset 类的同一个方法 data_generator。

```python
# 加载数据:mode=='trian'加载训练数据, =='val'加载验证数据
def data_generator(self,batch_size, mode, shape=(224,224), preprocess_input=None):
    filelist = []
    if mode == 'train':
        filelist = self.train_data_file
    elif mode == 'val':
        filelist = self.val_data_file
    else:
        print('Wrong Mode!')
        return
    while True:
        count = 0
        datas, labels = [],[]
        for file in filelist:
            im = Image.open(self.train_path + file)
            im = im.resize(shape)
            img = np.array(im)
            img = preprocess_input(img)

            if file.split('.')[0] == 'cat':
                label = np.array([0],dtype=np.int8)
            else:
                label = np.array([1],dtype=np.int8)
            datas.append(img)
            labels.append(label)
            count += 1
            if count >= batch_size:
                datas = np.array(datas).reshape([-1,shape[0],shape[1],3])
                labels = np.array(labels)
                yield (datas, labels)
                count = 0
                datas = []
                labels = []
```

该过程调用方法：

train_generator=dt.data_generator(batch_size=parms['batch_size'], mode='train',shape=(299,299), preprocess_input=resnet50.preprocess_input)

A）判断是否是加载训练数据

B）通过 Image 读取训练集图片，并 resize 至指定大小（根据模型的不同，输入的尺寸也不同，默认为（244,244）

C）将 Image 对象转成数组，输入到预处理函数 preprocess_input 进行预处理。预处理函数作为参数传递进来，根据模型的不同，该函数也不同。其中，预处理函数主要做了以下工作：将原本在[0,255]区间的像素值归一化到[-1,1]（或[0,1]），然后做零中心化（Zero_Center）操作，同时也将数据维度调整成 keras 后端对应的形式（Chanel first or Chanel last）。

D）根据文件名读取标签

E）根据指定的 batch_size 依次将该组数据标签对（datas,labels）缓存起来

F）重复 B～E 过程直至读取完训练数据

3.验证集同上，测试集同理，只是不缓存 labels

3.2 模型学习

本项目尝试了多种模型（VGGNet19，ResNet50，Inception V3，Xception），项目中每个模型被封装成了单独的类，这些类都包含了以下方法：初始化参数、搭建模型、训练、测试。由于单独的模型搭建思路完全一致，因此，下面以 ResNet50 为例，从初始化参数、搭建模型、训练、测试几个过程进行介绍。

A）初始化模型参数：该步骤主要是初始化的时候，将编译模型所需要的超参数赋值给模型的属性，如学习率、batch size、epoch 等。

B）搭建模型：

a) 首先从 keras.applictions 中导入加载了'imagenet'权重的预训练模型作为 base model，加载时不带有 top 分类层，因为 Imagenet 上分类为 1000 类：

```
base_model = ResNet50(input,weights='imagenet',include_top=False)
```

b) 将该 base model 的输出接入到全局平均池化层，替换原始的全连接层，有效降低过拟合效果，并且极大的降低了模型的参数，进一步的，为该层输出添加 Dropout，概率设置成 0.5

```
x = GlobalAveragePooling2D()(base_model.output)
x = Dropout(0.5)(x)
```

c) 将上层的输出接入到输出层，由于猫狗大战是二分类问题，因此输出层

只需要一个被 sigmoid 激活的神经元即可：

```
output = Dense(1,activation='sigmoid')(x)
```

d) 定义模型结构，生成模型对象：

```
resnet50=Model(base_model.input,output)
```

e) 定义优化器、损失函数及评价函数、编译模型。由于 Adam 不仅能够处理稀疏梯度和非平稳目标，还能为不同的参数计算不同的自适应学习率，而且对内存需求较少，能够适用于大多非凸优化问题，也适用于大数据集和高维空间等特点[8]，已经被广泛使用，成为主流，因此本项目也采用该优化器来训练模型。官方给出的损失函数为 logloss，也即 sigmoid 交叉熵函数，能够精确衡量预测的标签和真实值之间的误差，模型预测越不准确时，logloss 越大，反向传播时参数更新力度就越大，非常适合猫狗大战项目；由于项目给定的样本类间数据量均衡，因此可以直接使用二分类的分类准确率作为评价函数。最后，传入以上参数，编译模型：

```
adam = Adam(lr=self.lr)
resnet50.compile(optimizer=adam,loss='binary_crossentropy',
metrics=['binary_accuracy'])
```

B）训练模型：利用数据生成器训练模型，需要将训练数据生成器和验证数据生成器作为参数传递给 resnet.fit_generator(),同时还需定义模型训练的 batch size 和 epoch，前者决定一次训练时一个生成器产生多少数据，该值取决于显存的大小，该值越大总的训练时间相对也较短，但占用的显存也就越大；而 epoch 根据模型是否预训练、是否容易过拟合、是否层数很多等等都相关，epoch 很大时，模型会不断拟合训练集，导致过拟合。而不同的模型需要训练的 epoch 也不一样都相同，寻找一个训练的恰到好处的 epoch 并不容易，因此，项目使用了

ModelCheckPoint 的方法，该方法能够保存所有 epoch 轮训练中验证集损失最小的模型，该模型即为最优训练的模型，因为衡量的是验证集上的最优，证明了此时模型的泛化能力是最强的。进一步的，设置了 earlystopping 以便在验证集损失不再降低时提前停止训练。具体过程如下：

a) 定义 earlystopping，设置为 4 轮不降低则停止训练：

```
earlystopping = EarlyStopping('val_loss', patience=4)
```

b) 定义 ModelCheckpoint 保存最优模型：

```
cp = ModelCheckpoint(weights_path,save_best_only=True,
save_weights_only=True,verbose=1)
```

c) 将以上两个函数作为回调函数传入训练方法（部分参数略）：

```
calls = [earlystopping,cp]
hist = self.resnet50.fit_generator(train_generator, callbacks=calls)
```

D）测试模型：测试模型同训练，需要用测试数据生成器来实现按 batch 增量式测试，由于测试样本没有标签，因此我们需要将预测的概率和对应的图片存入文件中上传至官方进行打分。由于使用 os.listdir(path)读取图片名称时，得到的 filelist 是乱序的，因此该环节主要需要注意将测试集数据索引和预测的概率对其，以免出现上传后结果非常差的情况。主要思路为在测试数据生成器函数（Dataset.test_generator())中,将测试样本排序：

1.获取 filelist

2.对 filename 切割，保留数字型的索引

3.按索引对 filelist 排序

4.利用生成器对排序后的数据缓冲

另外还需要注意的是，当模型训练充分时，预测会有大量的 0 和 1 的概率

出现，这样会极大影响 logloss 的大小，当预测结果错误是，误差会被放大，因此还需要做的是将预测结果 clip 到[0.005,0.995]区内上。因此，整个测试过程如下：

1. 加载训练好的模型的权重；
2. 使用测试数据生成器测试网络，得到每张图片预测为狗的概率（因为训练时标签为 1 的是狗）；
3. 将得到的概率 clip

3.3 完善与提高

1. 融合模型

项目在尝试了多个模型以后，大胆地进行了模型间的组合以进一步提高模型的表现，由于 Inception V3 和 Xception 网络单个模型在本项目中表现的相对与 VGGNet19 和 ResNet50 要好，因此，实验中将两者融合达到了更好的性能，此外，Inception V3 和 Xception 网络思路是相似的，并且输入结构及预处理函数都是一样的，这就为该模型的融合提供了理论基础。具体融合思路如下：

A）模型初始化：该过程同单个模型一样，初始化超参数。

B）搭建模型：

a) 定义共同的输入层：input = Input(299,299,3)

b) 分别加载已训练好的 InceptionV3 模型和 Xception 模型

c) 分别取 InceptionV3 和 Xception 模型倒数第四层的输出，分别输入到 GlobalAveragePooling2D 层中进行降维，再降结果串联，扩充有效特征：

x1 = GlobalAveragePooling2D()(icv3.layers[-4].output)

x2 = GlobalAveragePooling2D()(xnet.layers[-4].output)

merged_x = Concatenate(axis=1)([x1,x2])

d) 然后将融合后的特征送入带有 512 个单元的全连接层，学习融合特征的显著信息部分，该层同时也避免了融合特征直接连接输出层维度骤降的问题：

merged_x = Concatenate(axis=1)([x1,x2])

x = Dense(512)(merged_x)

x = Dropout(0.5)(x)

output = Dense(1,activation='sigmoid')(x)

e) 编译模型同上

C）训练、测试模型同上

2. 参数优化

模型中重要的参数往往能够影响到最后的结果。因此实验为了确保对参数选择的严谨，设置了两组超参数的选择实验：val_rate(0.1,0.2)，dropout_rate (0.2,0.3,0.5,0.7,0.9)。对比时，确保其它参数一致，最终结果见下一章节。

四、结果展示和分析

实验通过设置多组对比实验来验证了以下几个观点及最优参数的确定，并分别展示和分析结果：

4.1 验证的观点

1. Xception 表现应该最优，VGG19 表现应该最差（参考 keras 官方文档提到的 ImageNet 上的表现）

2. 组合模型以提升总体性能

观点 1：Xception 表现应该最优，VGG19 表现应该最差。从 karas 给出的下表中可用看到，本次实验用到的四个模型里面 Xception 以往表现最佳。

模型信息

模型	大小	Top1准确率	Top5准确率	参数数目	深度
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.715	0.901	138,357,544	23
VGG19	549MB	0.727	0.910	143,667,240	26
ResNet50	99MB	0.759	0.929	25,636,712	168
InceptionV3	92MB	0.788	0.944	23,851,784	159
IncetionResNetV2	215MB	0.804	0.953	55,873,736	572
MobileNet	17MB	0.665	0.871	4,253,864	88

下面给出本次猫狗大战项目中的结果（Kaggle 测试集 score）：

Model	VGG19	ResNet50	InceptionV3	Xception
Logloss	0.08114	0.05515	0.04436	0.4010

从该表可以明显看出，Xception 网络以及 InceptionV3 明显优于前两者，这也符合我们的预期，也符合 Keras 给出的信息。此外，Xception 和 InceptionV3 网络性能相差无几，前者略胜一筹。

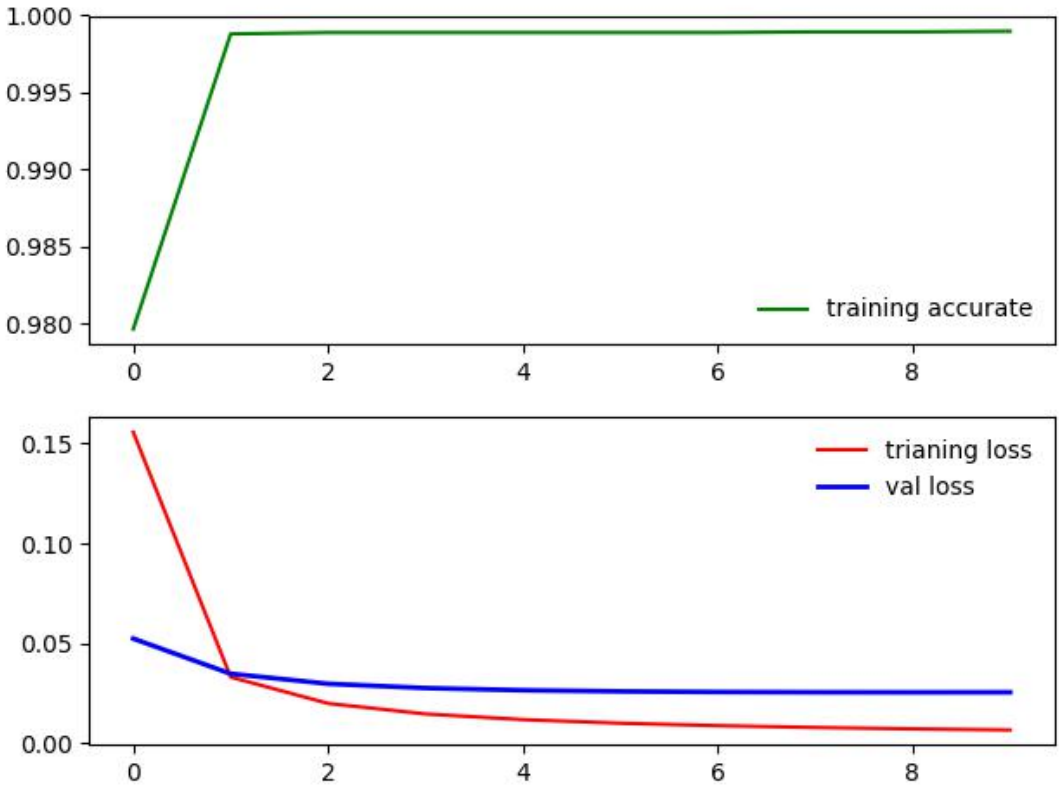
观点 2：融合模型以提高整体性能

Model	InceptionV3	Xception	Merged_Model
Logloss	0.04436	0.04010	0.03738

从该表可用得出，融合后的模型的确有一定的提升，证明了两个模型的泛化性能较单个模型而言相对好一些。

（另，该实验结果参数{‘val_rate’:0.1,’dropout_rate:0.5}）

从以上结果可用得出，本次“猫狗大战”以最终融合的模型结果，在 Kaggle 比赛提交过的分数上 30 名左右而结束，达到了预期目标。好在模型影响结果的超参数只有一个学习率，调整起来比较容易，最终学习率设置为 0.0001，batch_size 设置为 100，epoch 为 10。最终模型训练时各数据变化如下图所示：



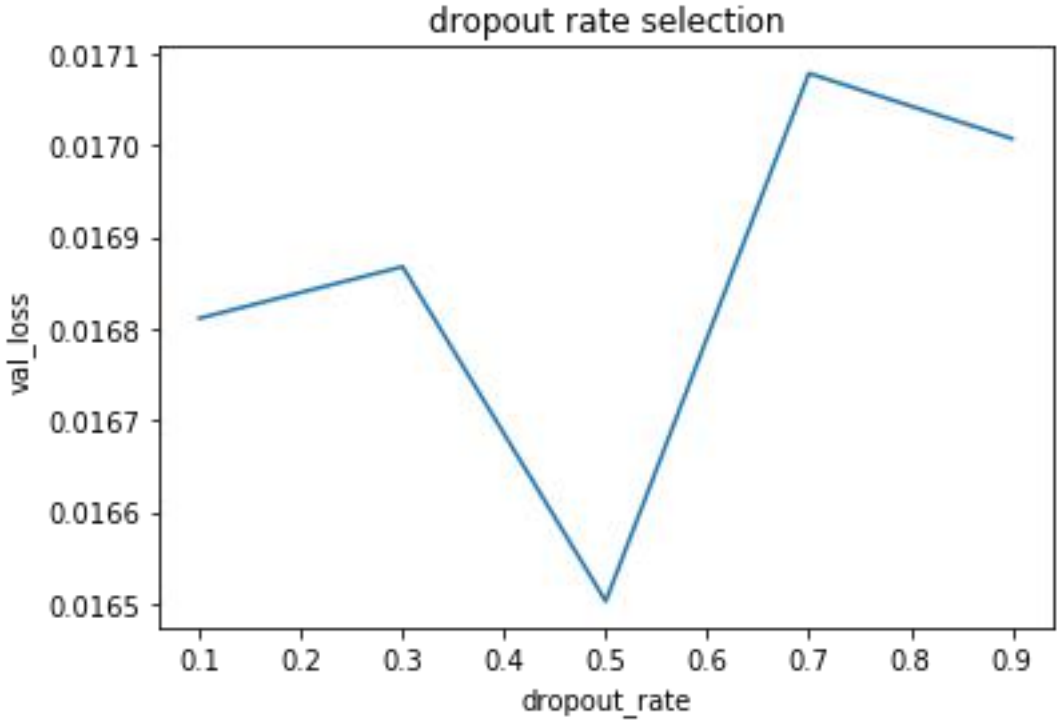
可以明显看到，绿色的线表示训练集上的准确率，以及非常接近 1 了说明训练很充分；红色的线代表训练集的 loss，无限趋向于 0；而蓝色表示验证集的 logloss，在前两个 epoch 时比训练集低是因为记录的是每个 epoch 结束时（即已经训练了一轮）的验证 loss，而训练集的 loss 是这个 epoch 所有个 batch 训练过程中的 loss 平均值。而后训练集 loss 继续下降，val loss 不再能低于训练集的 loss，这是非常合理的，因为验证集并没有参与训练，只是充当一个测试作用，并将最好的结果的模型当做泛化能力最强的模型保存起来。

4.2 最优参数的确定

实验中，我们对验证集比例（val rate）和融合模型的 dropout rate 均做了不同参数的对比实验。其中 val rate 从[0.1,0.2]选择，dropout rate 从[0.1,0.3,0.5,0.7,0.9]中选择。实验结果如下图表所示（具体实验运行记录见 notebook 导出的 html 中的第 9 章节)。

Val Rate	0.1	0.2
Val Loss	0.01873	0.01625

从该表可以得出，不同的验证集比例的确能够影响到实验的结果。原因是验证集大小也决定了训练的大小，当验证集较大时会导致训练不够充分，导致在验证集上的泛化能力差;验证集较小时，在验证集上的表现又不具备足够的代表性，缺乏可信度。因此第一次尝试项目时选择的是 0.1 的比率，导致最终结果的确不如第二次尝试使用 0.2 的结果。



上表是不同 dropout rate 对混合模型的影响，可以明显的看到，dropout 在

0.3 到 0.6 之间相对较好，并且对于此项目，0.5 达到最优效果。但图中 0.9 的 dropout rate 比 0.7 要好，这是因为项目中使用的混合模型均以完全训练到最优，然后才在特征层融合的，而 dropout 是作用于对融合特征的，因此即使融合特征层的参数只有少部分参与到训练，但经过 10 个 epoch 的训练，也会有一定的性能，如图中所示 0.017 左右。另外，也因为这 5 次对比实验均只跑了一次，偶然性也会导致这样的结果（由于一轮实验时间代价太大，目前只好跑一次）。

五、项目总结

本次项目虽然最终排名挺让我感觉满意，但整个过程遇到了太多的困难，主要耗费精力的问题为下面几个：

1. 内存不够支撑 Model.fit()。遇到的第一个难题，就是怎么能够将 25000 张图片读取到一个变量里面，这样才能使用 Model.fit 来训练。当时的思路是多次调用 fit 函数，结果发现每次调用 fit 的时候，实际上是重新训练，当时这让我非常头疼。再尝试了缩小图片大小到（128,128）时，能一个变量完全存储了，但另一个问题又出现了，尝试使用的第一个模型（vgg19）要想使用预训练的模型，必须要输入结构一致，因此最初方案是不使用预训练模型，直接训练刚“出生”的 VGG19，结果发现很难训练，几个小时下来 loss 都只会降低一点点。再查阅了 keras 文档才发现，keras 支持 fit_generator，如获至宝，最终得以解决。

2. 验证集 loss 能降到 0.02 左右，测试集却始终保持（0.07）左右的水平。当时非常不能理解，因为我设定的验证集是训练开始到训练结束都不会变的，也不会出现验证集泄露等问题，验证集 loss 能降下来一定代表着测试集 loss 不会太高。在查阅了很多资料以后，依旧无果，最终通过可爱的培神的指导，对测试

的概率结果进行 clip，才得以解决这个问题。

3. 融合的模型一度没有单个的模型性能好。这个问题也困扰了一段时间，试过很多种融合方式，比如直接粗暴的生成单个模型的深度特征（去掉 top 层），然后将它们保存起来，再串联起来作为新的输入，发现这种过程太过繁琐，保存中间变量也浪费内存，也不符合端到端的模型性质。后来再反复调整融合后的层（比如几层全连接等等），效果均不理想，后来得知全连接层特别容易过拟合，尽量少用。最终，项目在搭建融合模型时就把训练好的两个模型加载进来，把它们去掉 top 层后的输出分别全局平均池化后串联起来得到融合特征，且通过锁层的方式把融合特征之前所有的层锁住，不参与到训练，实现了不保存中间结果的端到端模型。

虽然以上问题最终都解决了，但是对项目的还有些想法无法及时实现，否则就错过了毕业末班车了。以后可以从以下几个方面进行改进：

1. 尝试更多的更先进的模型，如 InceptionResNetV2。
2. 尝试融合更多的模型
3. 尝试做一下数据增强，比如旋转图片生成更多训练样本等等。
4. 尝试做一下结果层的融合，本次实验只做了特征层的融合。

最后，这次猫狗大战我才是最大的赢家，学到了太多的东西，这让我对深度学习领域的知识更加充满了渴望，同时也对代码开发的能力有了进一步的理解和加强，往后争取把其他毕业项目也过一遍！

参考文献或博客

[1]Fortuna J. ICA feature extraction and support vector machine image classification[J]. https://www.lap-publishing.com/, 2005.

[2]Raghu P P, Yegnanarayana B. Multispectral Image Classification Using Gabor Filters and Stochastic Relaxation Neural Network[J]. Neural Networks, 1997, 10(3):561-572.

[3]Sung J, Bang S Y, Choi S. A Bayesian network classifier and hierarchical Gabor features for handwritten numeral recognition[J]. Pattern Recognition Letters, 2006, 27(1):66-75.

[4]Yann LeCun, Gradient-Based Learning Applied to Document Recognition, 1998

[5]Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

[6]Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the Inception Architecture for Computer Vision[C]// IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2016:2818-2826.

[7]Hu J, Shen L, Sun G. Squeeze-and-Excitation Networks[J]. 2017.

[8]https://blog.csdn.net/u012759136/article/details/52302426

[9]Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions[J]. 2016:1800-1807.

[10][https://blog.csdn.net/yunpiao123456/article/details/52437794]

[11]https://blog.csdn.net/zhangjunbob/article/details/53258524