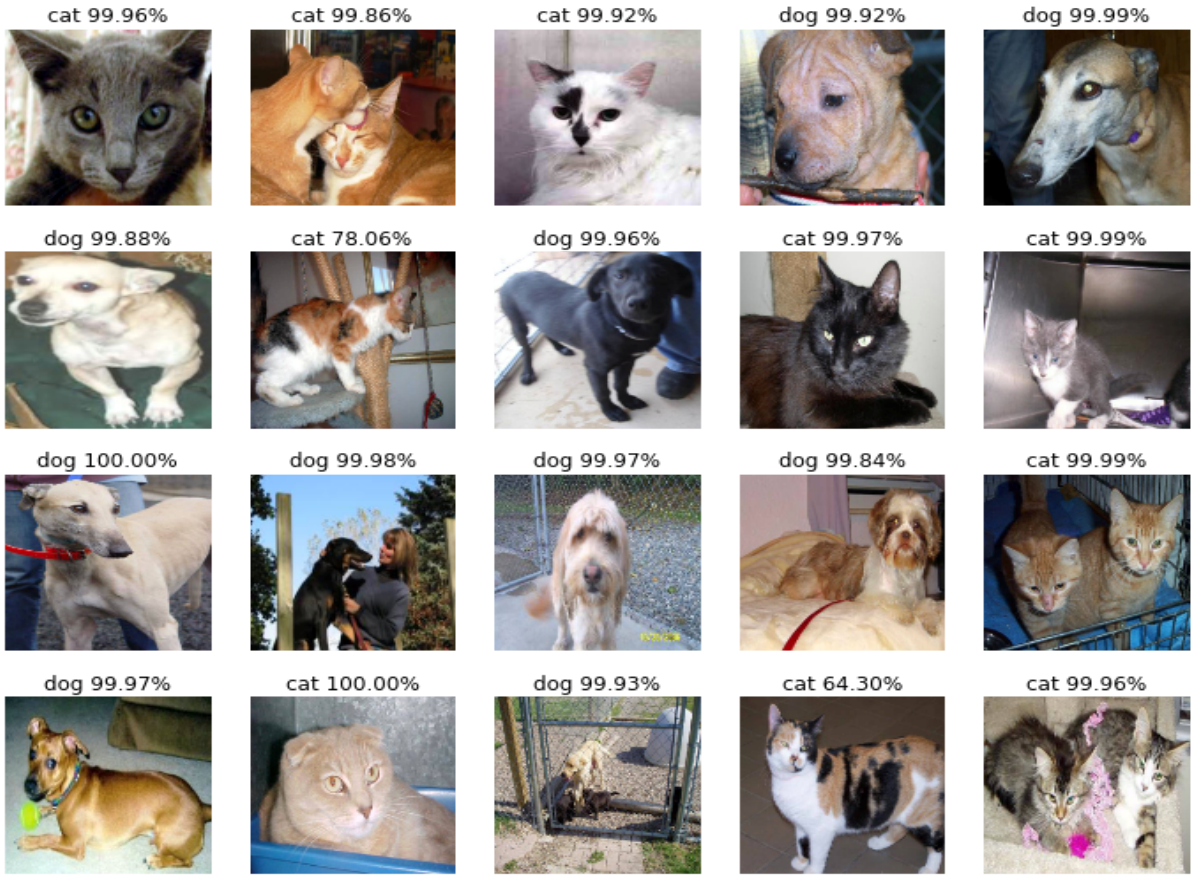


机器学习纳米学位

《猫狗大战》项目实战

2018 年 10 月 10 日

Junjie Huang
SERAPHIC Information Technology (Shanghai) Co., Ltd.
Songhu Road No. 433, Yangpu District, Shanghai

一、定义

1.1 项目背景

该项目来自于 Kaggle 上的一个竞赛：Dogs vs. Cats。参赛者需要训练一个模型去识别给定的图片是猫或者狗，这是计算机视觉领域一个典型的图像分类问题。

计算机视觉是深度学习技术最早实现突破性成就的领域。自 2012 年度深度学习算法 AlexNet 赢得图像分类比赛 ILSVRC 冠军，深度学习开始受到学术界广泛的关注。短短数年间，深度学习算法迅猛发展，图像分类的错误率不断递减。深度学习完全打破了传统机器学习算法在图像分类上的瓶颈，实现了计算机视觉研究领域的重大突破。

在技术革新同时，工业界也将图像分类、物体识别应用于各种产品中了。在 Google，图像分类、物体识别技术已经被广泛应用于无人驾驶汽车、YouTube、谷歌地图、谷歌图像搜索等产品中。

深度学习虽然最早兴起于图像识别，但是在短短几年内迅速推广到了机器学习的各个领域，并且均有出色的表现，在图像识别、语音识别、自然语言处理、机器人、电脑游戏、搜索引擎和金融等各大领域均有应用。

1.2 问题描述

本项目的目标是根据给定数据集训练模型，识别给定的图片是猫或狗，并且识别准确率能到达一个较高的水平。

这是一个典型的二分类问题，可以通过传统的监督学习算法如支持向量机、朴素贝叶斯分类器等算法来完成，也使用深度学习方法如卷积神经网络来完成。

1.3 输入数据

Kaggle 猫狗大战项目提供了测试集文件 train.zip 和训练集文件 test.zip，前者用于训练模型，后者用于模型预测。

训练集包含 25k 张照片，其中猫和狗各占一半，每张图片都带有标签；测试集则包含 12.5K 张照片，没有标记为猫或者狗。

1.4 评估指标

本项目拟采用对数损失函数来评估模型表现，定义如下：

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

其中：

- n 为测试集中的图片数量
- $\hat{y}_i$ 为预测图片内容为狗的概率

- y_i 是类别标签，1 对应狗，0 对应猫

对数损失越小，代表模型的表现越好，预测能力越强。

二、分析

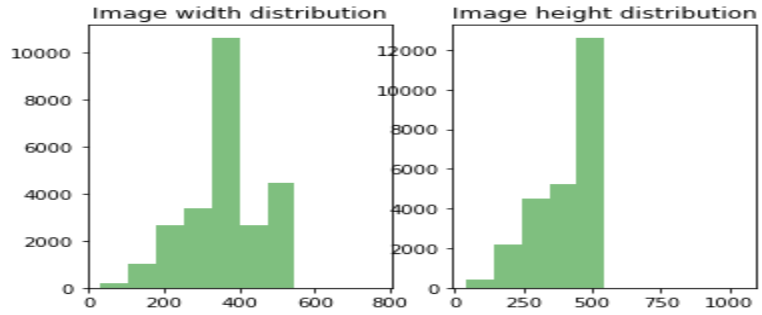
2.1 数据探索

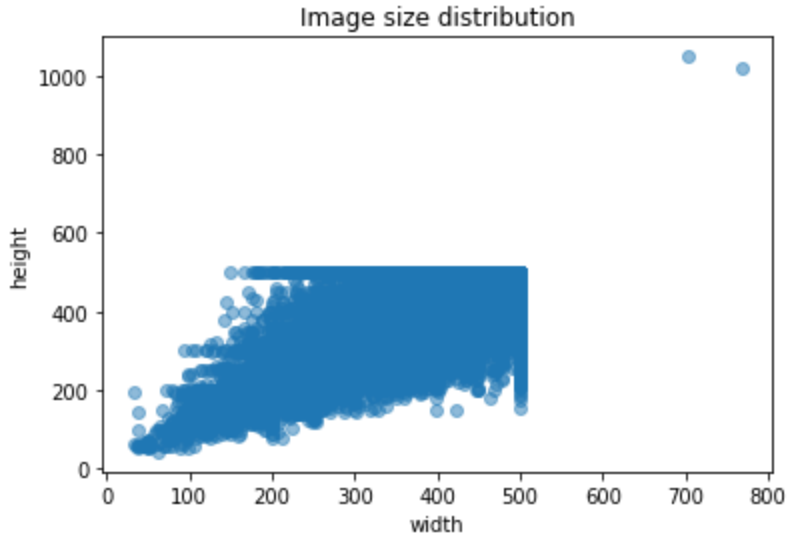
通过研究数据集，发现图片尺寸相差很大，比如测试集中，图片宽度范围为 42px～1050px，高度范围为32px～500px，像素乘积范围为 1900～785664。

训练集中最小的图片为 dog.10747.jpg，尺寸为 50x38：

训练集中最大的图片为 cat.835.jpg，尺寸为 1023x768：

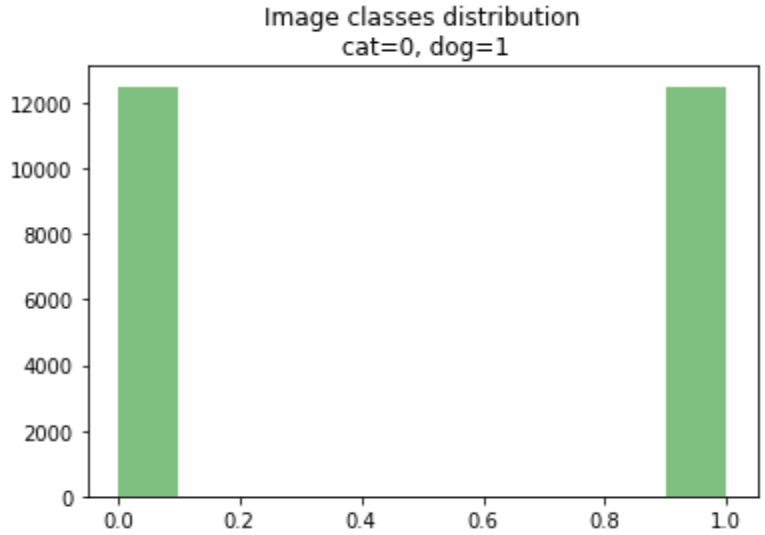
对训练集中的图片按像素值，使用 matplotlib 绘制图片大小的分布图如下：



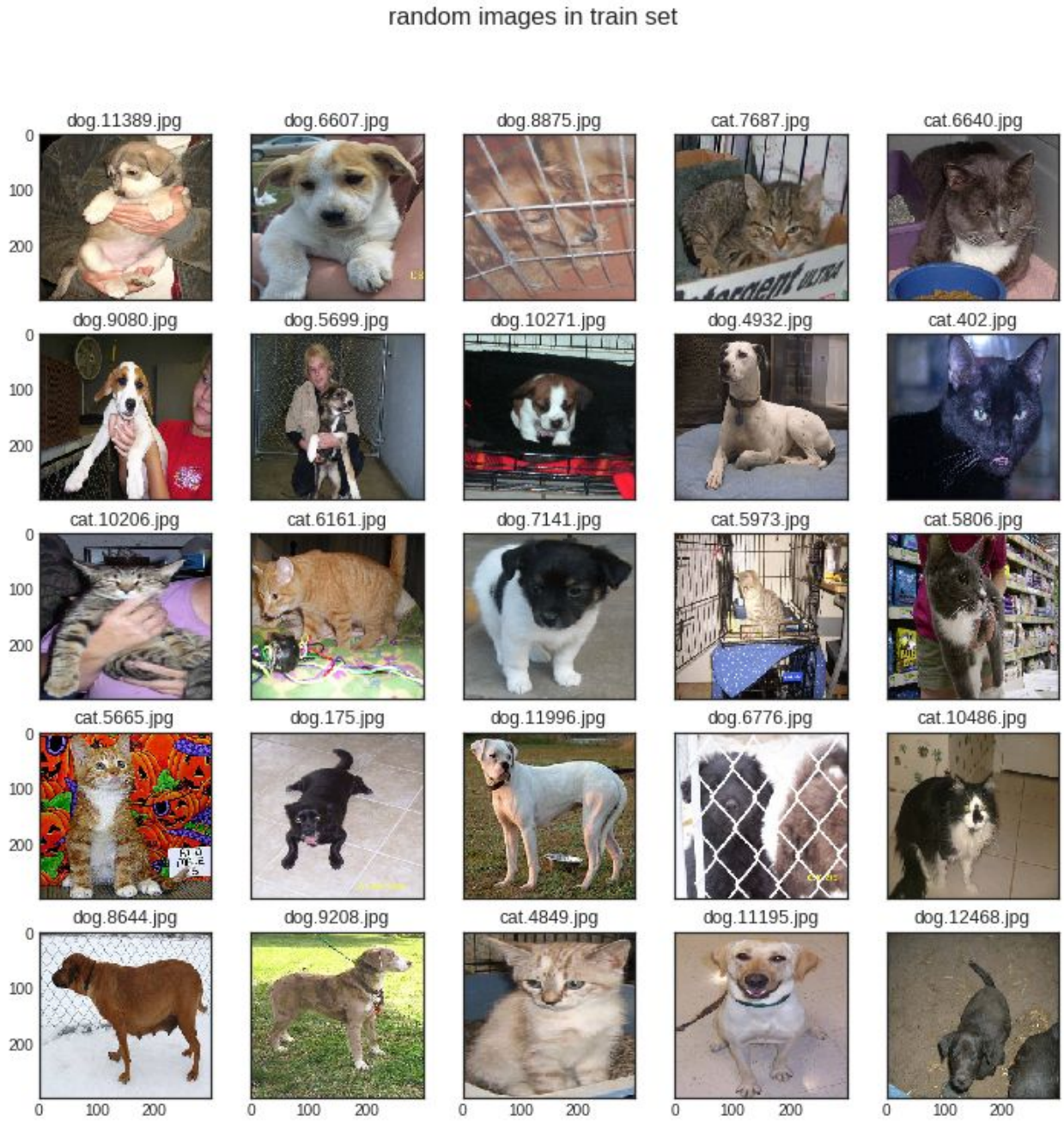


可以看出，图片宽度集中在200～500之间，图片高度集中在250～500之间的图片，同时图片的宽度和高度呈现出一定程度的关联性。在上面的散点图中可以明显看到有两个图片异常大，图片大小不一致无法应用于神经网络，因此需要对其进行缩放处理。

训练集中的类别分布直方图如下，可以看到猫和狗的图片数量均为 12500。



这里随机展示一下训练集中的图片：



另外，通过手工检测，发现训练集图片中有一些既非猫又非狗的异常值，这些脏数据会影响模型的精度，需要从训练集中剔除。

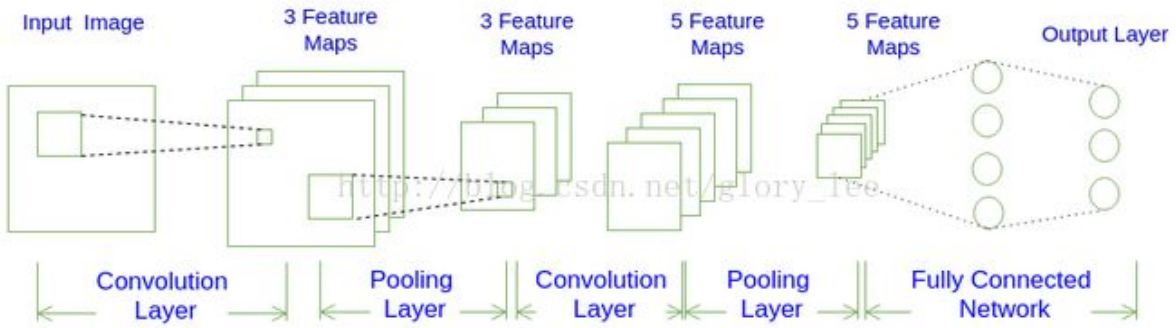
2.2 算法及技术

项目要求使用深度学习方法识别一张彩色图片是猫还是狗，计算各自的概率。这里拟采用迁移学习的方法，使用卷积神经网络（CNN）来解决问题。

2.2.1 卷积神经网络

卷积神经网络（Convolutional Neural Network, CNN）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，对于大型图像处理有出色表现。

卷积神经网络由一个或多个卷积层和顶端的全连通层组成，同时也包括关联权重和池化层。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。以下是一个卷积神经网络的结构示意图：

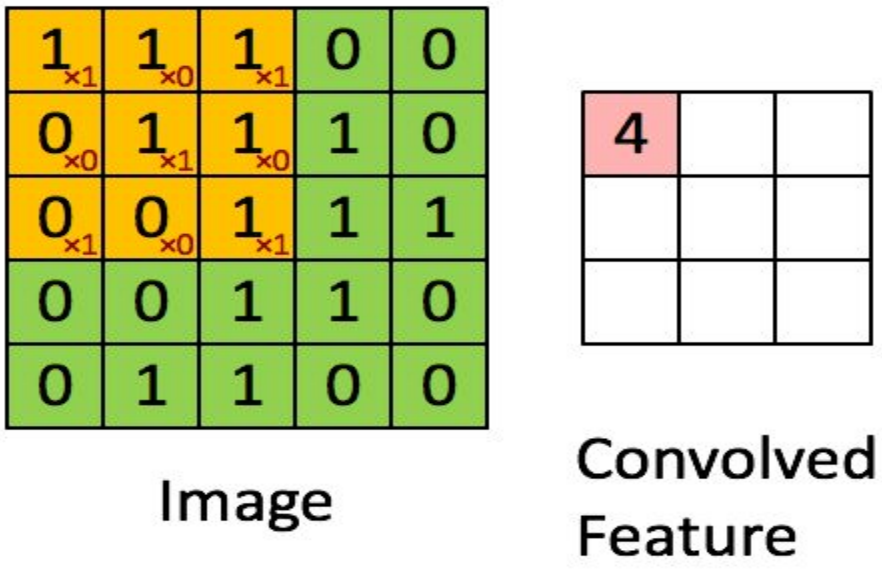


一个卷积神经网络由若干卷积层、Pooling 层、全连接层组成。它的常用架构模式为：

INPUT（输入层）-> [[CONV（卷积层）]*N -> POOL?（池化层）]*M -> [FC（全连接层）]*K

也就是 N 个卷积层叠加，然后叠加一个 Pooling 层（可选），重复这个结构 M 次，最后叠加 K 个全连接层。

卷积层是卷积核在上一级输入层上通过逐一滑动窗口计算而得，卷积核中的每一个参数都相当于传统神经网络中的权值参数，与对应的局部像素相连接，将卷积核的各个参数与对应的局部像素值相乘之和（通常还要再加上一个偏置参数），得到卷积层上的结果。卷积过程如下图所示：



Pooling 层的主要的作用是下采样，通过去掉 Feature Map 中不重要的样本，进一步减少参数数量。Pooling 的方法很多，最常用的是 Max Pooling 和 Average Pooling。Max Pooling 实际上就是在 n*n 的样本中取最大值，作为采样后的样本值；Average Pooling 则是邻域内特征点只求平均。根据相关理论，特征提取的误差主要来自两个方面：

1. 邻域大小受限造成的估计值方差增大；
2. 卷积层参数误差造成估计均值的偏移。

一般来说，average-pooling 能减小第一种误差，更多的保留图像的背景信息，max-pooling 能减小第二种误差，更多的保留纹理信息。

全连接层（fully connected layers，FC）则在整个卷积神经网络中起到"分类器"的作用。如果说卷积层、池化层等操作是将原始数据映射到隐层特征空间的话，全连接层则起到将学到的"分布式特征表示"映射到样本标记空间的作用。

与其他深度学习结构相比，卷积神经网络在图像识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。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。

2.2.2 迁移学习

通过使用之前在大数据集上经过训练的预训练模型，我们可以直接使用相应的结构和权重，将它们应用到我们正在面对的问题上，这种方法被称作为"迁移学习"，即将预训练的模型"迁移"到我们正在应对的特定问题中。

这种方式也符合我们我们人类自身的学习和进化模式，我们后人在科技、哲学、艺术、人文等各个领域所取得的进展，无一不是站在前人肩膀上。使用迁移学习，我们就可以使用前人取得的重大进展，而不需要从零开始训练一个神经网络，可以节省大量工夫。

迁移学习是机器学习的一种重要技术。在实践中，通常涉及到使用来自 ResNet、Inception 等模型的预训练的权重初始化模型，然后要么将其用作特征提取器，要么就在一个新数据集上对最后几层进行微调。使用迁移学习，这些模型可以在任何我们想要执行的相关任务上得到重新利用。

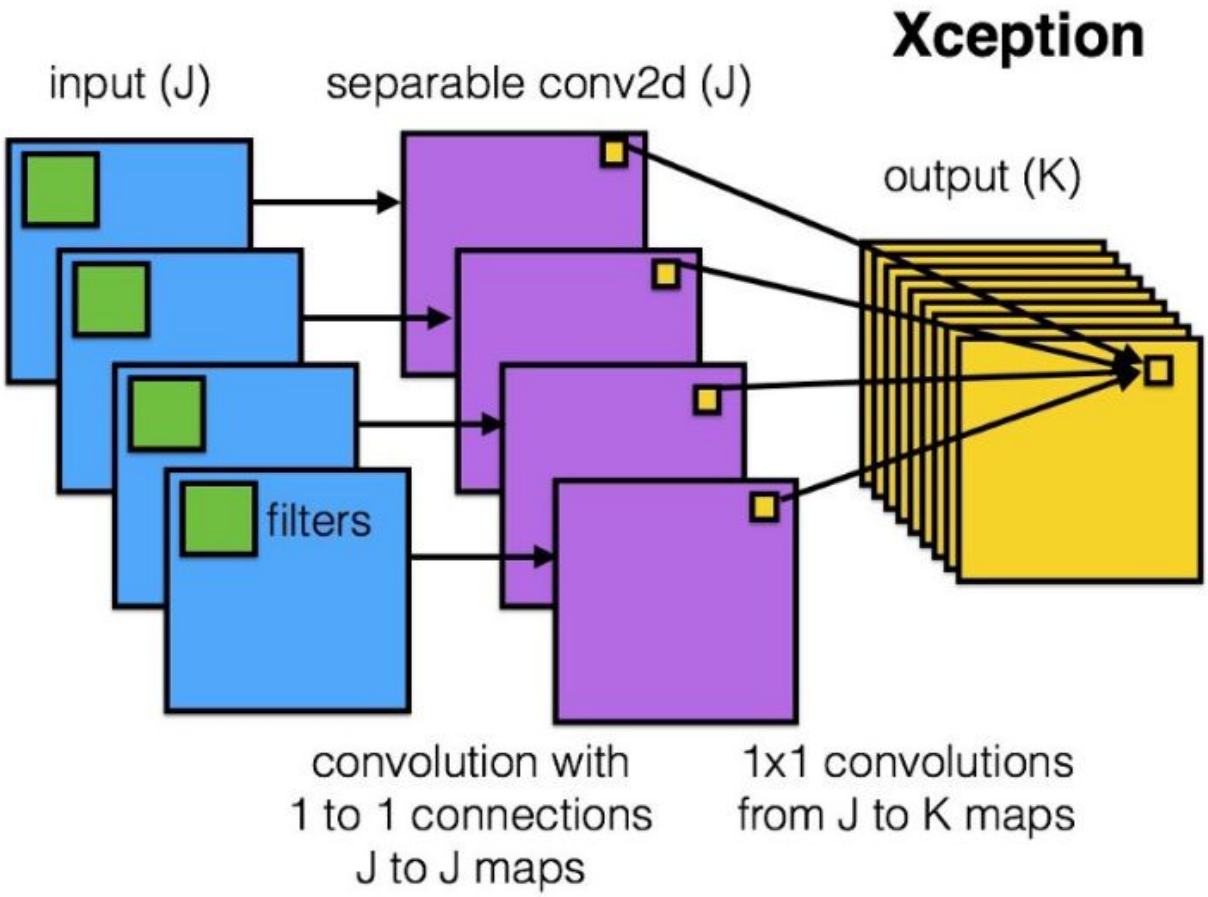
2.3 基准模型

本项目的本质是图像分类，keras 提供了几种开箱即用的用于图像分类的 CNN 模型：

- VGG16
- VGG19
- ResNet50
- Inception V3
- Xception

根据 keras 基准模型的性能对比，拟采用 Top-1 准确率最高、模型参数和深度适中的 Xception 模型作为基准模型。

Xception 表示「extreme inception」，它是从 Inception 发展而来，将 Inception 的原理推向了极致。在 Inception 中，使用 1×1 的卷积将原始输入投射到多个分开的更小的输入空间，而且对于其中的每个输入空间，都使用一种不同类型的过滤器来对这些数据的更小的 3D 模块执行变换。Xception 则更进一步。不再只是将输入数据分割成几个压缩的数据块，而是为每个输出通道单独映射空间相关性，然后再执行 1×1 的深度方面的卷积来获取跨通道的相关性。



在 ImageNet 数据集上，Xception 的表现稍稍优于 Inception v3，而且在一个有 17000 类的更大规模的图像分类数据集上的表现更是好得多。最重要的是，它的模型参数的数量和 Inception 一样多，说明它的计算效率也更高。

2.4 项目目标

本文目标是 Kaggle 排行榜（public Leaderboard）前 10%，也就是 LogLoss 要达到 0.06127。

三、方法

3.1 数据预处理

3.1.1 调整文件目录层级

首先从 Kaggle 网站下载本项目所需的文件 all.zip，从中解压出验证集图片和测试集图片，并放入 data 目录。为了使用 sklearn.datasets.load_files 以及 flow_from_directory 等工具，需要将测试集的图片按猫和狗分为两个目录存储。为了节省空间，可以为每个图片创建符号链接，分别放在其类别对应的目录下。同时创建符号连接的方式也方便后续从测试集中剔除异常值。

```
import os

# unzip files, make directries
if os.path.exists('all.zip') and not (os.path.isdir('data')):
    os.system('unzip -o all.zip; unzip -o train.zip; unzip -o test.zip')
    os.system('mkdir data; mv train test data')

# make symbol links
if not os.path.exists('data/train2'):
    os.system('mkdir -p data/train2/dog data/train2/cat')
    for filename in os.listdir('data/train'):
        if 'dog' in filename:
            os.symlink('../../train/%s' % filename, 'data/train2/dog/%s' % filename)
        else:
            os.symlink('../../train/%s' % filename, 'data/train2/cat/%s' % filename)

if not os.path.exists('data/test2'):
    os.mkdir('data/test2')
    os.symlink('../test/', 'data/test2/test')
```

最终目录层级组织如下：

```
data
├── test
│   ├── 1.jpg
│   ├── .....
│   └── 12500.jpg
├── test2
│   └── test -> ../test
├── train
│   ├── cat.1.jpg
│   ├── .....
│   └── dog.12500.jpg
└── train2
    ├── cat
    │   ├── cat.1.jpg -> ../../train/cat.1.jpg
```

```
15.     │   ├── ......
16.     │   └── cat.12500.jpg -> ../../data/cat.12500.jpg
17.     └── dog
18.         ├── dog.1.jpg -> ../../train/dog.1.jpg
19.         ├── ......
20.         └── dog.12500.jpg -> ../../train/dog.12500.jpg
```

3.1.2 剔除训练集中的异常值

最初并没有找到剔除异常值的捷径，经老师提点后参考了[知乎专栏]上的一种方法，利用预处理模型来剔除异常值。

其主要思路是，ImageNet 上训练的模型，已具有高精度的图片分类能力，我们可以利用它来预测训练集图片。由于 ImageNet 上的图片类别包含猫和狗，但是按猫和狗又细分有很多类，我们可以提取 ImageNet 分类中的猫和狗的类别，如果预测的结构与提起的类别匹配，则说明该图片有很大可能性是猫或者狗。但是预测结构难免会有一定的误差，怎样来减少误差呢？我们还需要结合 ImageNet 的 Top-N 指标。Top-N 的含义是，算法给出 N 个可能的分类，N 个分类的概率之和代表正确分类的准确率。我们可以增大 N 的值，来提高匹配猫狗的准确率。

首先载入从网上下载的 ImageNet 分类 csv 文件，提取其中的猫狗分类。

```
1.  def get_imagenet_classes(file_path):
2.      dogs = []
3.      cats = []
4.      with open(file_path,'r') as f:
5.          content = csv.reader(f)
6.          for line in content:
7.              if line[1] == '狗':
8.                  dogs.append(line[0])
9.              elif line[1] == '猫':
10.                 cats.append(line[0])
11.     return (dogs, cats)
```

这里使用在 imagenet 上训练过的 Xception 模型来预测训练集。在此之前读入图片到内存，归一化处理，resize 到 Xcepiton 默认的大小。

```
1.  image_size = (299, 299)
2.
3.  def read_image_from_path(path):
4.      img = image.load_img(path, target_size=image_size)
5.      x = image.img_to_array(img)
6.      x = np.expand_dims(x, axis=0)
7.      x = preprocess_input(x)
8.      return x
9.
10. def read_image_from_paths(paths):
11.     preprocess_imgs = [read_image_from_path(path) for path in tqdm(paths)]
12.     return (preprocess_imgs)
13.
14. # 获取归一化后的图片
15. preprocess_images = read_image_from_paths(train_files_paths)
```

使用模型预测训练集，我们可以得到每个图片的 Top-N 预测结果。对于每个图片，如果 Top-N 分类中有任何一个类别属于猫或者狗，那么这个图片就有一定概率是猫或者狗；反之，如果没有任何一项属于猫或者狗，那么就判定这张图片就是非猫非狗的异常值了。这里的 N 越高，最终的结果可会越准确，经多次尝试，取 N=50。

使用预训练模型预测训练集图片，获取每张图片的 top-50 预测。

```
from keras.applications.xception import decode_predictions
def do_model_predict(model, imgs, top=10):
    pred = model.predict(imgs)
    return decode_predictions(pred, top=top)[0]

def get_predict_result(model, imgs, top=10):
    result = [do_model_predict(model, x, top) for x in tqdm(imgs)]
    return result

base_model = Xception(include_top=True, weights='imagenet')
# 预测训练集
pred = get_predict_result(base_model, preprocess_images, 50)
```

于是，结下来任务就是计算每张图片的 Top-50 分类中，匹配到猫或者狗的类别总数。如果总数为零，就标记为异常数据。

```
# 获取预测结果中，能匹配到猫狗类型的类别总数
def get_matched_classes_nums_list(pred):
    mathed = []
    # 每一个图片的 top 50 预测结果：
    for item in pred:
        # top 50 中所有的类别
        classes = [x[0] for x in item]
        # 所有的分类中，匹配到猫狗类型的类别总数
        mathed.append(sum([1 for x in classes if x in imagenet_classes]))
    return mathed

matched_classes_nums = get_matched_classes_nums_list(pred)

# 获取异常数据的文件名
def get_unmatched_files_paths(class_nums_list):
    unmatched = []
    for index, value in enumerate(mathed_classes_nums):
        if value == 0:
            unmatched.append(train_files_paths[index])
    return unmatched

unmatched_images_paths = get_unmatched_files_paths(matched_classes_nums)
print("{} images unmathed".format(len(unmatched_images_paths)))
```

最终找到 34 张未能匹配猫狗分类的图片，使用 matplotlib 对其进行可视化输出：

```
def show_unmatched_images(paths, title):
    image_size = (299, 299)
```

```
    plt.style.use('seaborn-white')
    fig, ax = plt.subplots(7, 5, sharex='col', sharey='row', figsize=(12, 12))
    fig.suptitle(title, fontsize=16)
    for i in range(7):
        for j in range(5):
            index = i * 5 + j
            if (index < len(paths)):
                path = paths[index]
                img = image.load_img(path, target_size=image_size)
                ax[i, j].set_title(os.path.basename(path))
                ax[i, j].imshow(img)

show_unmatched_images(unmatched_images_paths, "unmatched images")
```

可以看出，利用 top-50 预测结果，筛选出的未成功匹配猫狗的图片中，大部分都被正确地识别为异常图片，但是也有不少猫或者狗的图片被识别错误。我们需要保留这被错误识别的猫狗图片，然后从训练集中剔除异常值。

```
# 保存被错误识别的猫狗图片（能看出是猫或者狗）
resolved = ['cat.12424.jpg','dog.8898.jpg','dog.9188.jpg',
            'cat.3216.jpg','cat.7968.jpg','cat.12476.jpg',
            'cat.10636.jpg','dog.1259.jpg','dog.1895.jpg',
            'cat.5071.jpg']

need_remove_images = []
for file_path in unmatched_images_paths:
    file_name = os.path.basename(file_path)
    if file_name not in resolved:
        need_remove_images.append(file_path)
```

移除异常值时，这里使用了一个小技巧，并没有真正删除文件，只是对符号链接做 unlink 操作：

```
def unlink_files(paths):
    for file_path in paths:
        if os.path.exists(file_path) and os.path.islink(file_path):
            os.unlink(file_path)

unlink_files(need_remove_images)
```

经此处理，从们从训练集中成功找到 24 个异常值并剔除。

3.1.3. 重新载入训练集

移除异常值后，我们重新载入训练集图片内存。并缩放到 Xception 默认输入大小 299 x 299。这里尝试过不同的尺寸：更小的尺寸（如 224 x 224）会加速训练的过程，但是最终在验证集上的得分更低，最终未能到达 Kaggle 目标；更高的尺寸（320 x 320）会加剧内存占用，导致代码报内存错误。

```
# 删除异常值后，重新载入训练集图片，并统计图片张数
train_files_paths, train_files_targes = load_train_dataset('data/train2')
train_files_labels = [1 if 'dog' in name else 0 for name in train_files_paths]

target_image_size = (299, 299)

X = np.array([cv2.resize(cv2.imread(path, cv2.IMREAD_COLOR), #立方差值
                         target_image_size,
                         interpolation=cv2.INTER_CUBIC)
              for path in train_files])
Y = np.array([1 if 'dog' in path else 0 for path in train_files]) # for sigmoid
```

3.1.4 数据增强

使用 keras 的内置 ImageDataGenerator 可以对输入的图片进行数据增强：随机剪切变换，随机缩放，左右、上下翻转，随机旋转一定角度。数据增强是扩充数据集、改善模型效果简单而有效的方法。

```
refoce_gen = ImageDataGenerator(shear_range=0.2,       #随机剪切变换强度
                                zoom_range=0.2,        #随机缩放幅度
                                rotation_range=45,     #随机旋转角度范围
                                horizontal_flip=True, #随机左右翻转
                                vertical_flip=True)    #随机上下翻转
gen = ImageDataGenerator()
train_generator = gen.flow(X_train, Y_train, batch_size=batch_size)
validation_generator = gen.flow(X_validation, Y_validation, batch_size=batch_size)
```

从理论上分析，本项目的训练集数据量达到了 25000，规模并不小，所以通过数据增强来获取的提升应该很有限。通过实验发现确实如此，此处的数据增强并没有显著的改善验证集上的表现，相反会使得导致验证集上的损失过大。其可能的原因应该是相关的参数组合不当缘故，由于时间原因未能充分探究。

3.2 执行过程

3.2.1 构建模型

3.2.1.1 基础模型

此前的尝试是在 Xception 模型后直接附加全连接层来训练，这样每次训练都要跑一个巨大的神经网络，训练 50 代将是一个漫长的过程，非常痛苦。并且由于卷集层是不可训练的，这些计算就是浪费。这也是之前 val_loss 震荡很大，一直出现严重过拟合的原因。

经老师提示，采用了知乎上一种方法，通过已训练好的 Imagenet 权重对数据进行特提取，提前把卷积层提取的特征向量存储到硬盘，然后再用提取的特征做一个简单的全连接。这样做的好处是一旦保存了特征向量，即使是在普通笔记本上也能轻松训练。

```
from keras.optimizers import *
from keras.applications import *
from keras.models import Model

input_tensor = Input(shape=(299, 299, 3))
x = input_tensor
x = Lambda(xception.preprocess_input)(x)
base_model = Xception(input_tensor=x, include_top=False, weights='imagenet')
base_model = Model(base_model.input, GlobalAveragePooling2D()(base_model.output))
base_model.summary()
```

3.2.1.2 使用基础模型进行预测

```
train = base_model.predict_generator(train_generator,
    steps=len(train_generator.filenames)//16, verbose=1)

test = base_model.predict_generator(test_generator,
    steps=len(test_generator.filenames)//25, verbose=1)
```

3.2.1.3 保存特征权重

为了后续的训练，这里将提取出的特征向量先保存到文件。

```
import h5py
with h5py.File('weights.exception.hdf5', 'w') as f:
    f.create_dataset('train', data = train)
    f.create_dataset('test', data = test)
    f.create_dataset('label', data = train_generator.classes)
```

3.2.1.4 导入特征权重

通过读取保存的 h5 文件，可以按需导入特征权重：

```
from sklearn.utils import shuffle

X_train = []
X_test = []

with h5py.File('weights.exception.hdf5', 'r') as f:
    X_train.append(np.array(f['train']))
    X_test.append(np.array(f['test']))
    Y_train = np.array(f['label'])

X_train = np.concatenate(X_train, axis=1)
X_test = np.concatenate(X_test, axis=1)
```

3.2.1.5 增加 dropout 层及并分类

为了适合本项目猫狗的二分类，将输出层改为二分类的全连接，使用 sigmoid 激活函数。

```
input_tensor = Input(X_train.shape[1:])
m = Dropout(0.5)(input_tensor)
m = Dense(1, activation='sigmoid')(m)
model = Model(input_tensor, m)
model.summary()
```

模型概况如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2049
=================================================================
Total params: 2,049
Trainable params: 2,049
Non-trainable params: 0
```

在提取的卷积层特征向量的基础上，加上 dropout 层和一个全连接层。激活函数选用 sigmod，因为本项目是二分类问题；也可选用 softmax，但是需要对输入的训练集和验证集标签进行独热编码。这里的 Dropout 层参数是一个重要的超参数，后续会展开探索。

3.2.2 编译模型

```
## 编译模型
# model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
# model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
```

这里分别尝试了 rmsprop、adam、adadelta 三种不同的优化器，对比下来，adadelta 的收敛速度最快，而 adam 的对数损失最低，详细情况见后面 3.3.3.1 节的对比结果。

而损失函数之所以选择 binary_crossentropy，也就是对数损失函数 LogLoss，一是因为它与输出层选用的激活函数 sigmod 相对应，二是因为最终在 Kaggle 上的排名是也是按 Logloss 来评估的，这样我们就能根据验证集上的 val_loss 得分来估计 Kaggle 上的表现。

3.2.3 训练模型

设置迭代次数为 30 次。为防止过拟合，设定 val_loss 不再下降后的 3 个 epochs 后停止训练，并随时保存模型架构及最佳的权重值到文件，用于后续的预测。

```
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
epochs = 30
batch_size = 16

earlyStopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')
checkpointer = ModelCheckpoint(filepath='weights.best.xception.hdf5', verbose=1,
save_best_only=True)
callback_list = [checkpointer, earlyStopping]
history = model.fit(X_train, Y_train, validation_split = 0.2,
                    epochs = epochs, batch_size = batch_size, verbose=1,
                    callbacks=[checkpointer])
```

这里的 batch_size 是一个比较关键的超参数。越小的 batch_size，使得训练在更为平缓的局部最小值停止，最终的泛化能力也越好，同时收敛速度也会越慢。batch_size 越大则收敛速度越快，但同时也需要更多的迭代次数来达到较好的成绩。

另外，训练集和验证集的划分比例也是一个重要的参数，后面会展开探索。

3.2.3 探索超参数和优化器

这里主要探索 dropout、batch_size、 validation_split 参数及不同的优化器对 val_loss 的影响

为了方便测试不同参数，这里用一个函数将建立模型、编译及训练模型的过程封装成一个函数，参数为可能会尝试的超参数。

同时，将比较参数的过程也进行封装，这样就方便比较不同的参数。为了简便直观得看到参数的取值对 val_loss 的影响，这里用 pandas.DataFrame 将对比结果以表格的形式输出。

```
def specify_model_variables(optimizer='adadelta', dropout=0.2, batch_size=128,
                            validation_split=0.2, verbose=0, plot=False):
```

```
    input_tensor = Input(X_train.shape[1:])
    model = Model(input_tensor, Dropout(dropout)(input_tensor))
    model = Model(model.input, Dense(1, activation = 'sigmoid')(model.output))
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    earlyStopping = EarlyStopping(monitor='val_loss', patience=3, verbose=verbose,
mode='auto')
    checkpointer = ModelCheckpoint(filepath='weights.best.xception.hdf5',
verbose=verbose, save_best_only=True)
    callback_list = [checkpointer, earlyStopping]
    history = model.fit(X_train, Y_train, validation_split = validation_split,
                        epochs = 30, batch_size = batch_size, verbose=verbose,
                        callbacks=callback_list)
    if (plot == True):
        show_learning_curves(history)
    return (history.history['val_loss'], history.history['val_acc'])

def compare_model_variables(option_name, options_list):
    data = []
    for item in options_list:
        if (option_name == 'optimizer'):
            val_loss, val_acc = specify_model_variables(optimizer=item)
        elif (option_name == 'dropout'):
            val_loss, val_acc = specify_model_variables(dropout=item)
        elif (option_name == 'batch_size'):
            val_loss, val_acc = specify_model_variables(batch_size=item)
        elif (option_name == 'validation_split'):
            val_loss, val_acc = specify_model_variables(validation_split=item)
        else:
            print("unsupport option")
            return

        row = []
        row.append(np.average(val_loss))
        row.append(np.max(val_loss))
        row.append(np.min(val_loss))
        data.append(row)

    val_loss = pd.DataFrame(
        data=data,
        index=options_list,
        columns=['Avg', 'Max', 'Min']
    )
    val_loss.index.name = option_name
    return val_loss
```

3.2.3.1 不同的优化器

```
compare_model_variables('optimizer', ['adam', 'adadelta', 'rmsprop'])
```

optimizer	Avg	Max	Min
adam	0.021688	0.047435	0.018209
adadelta	0.021515	0.046990	0.018312
rmsprop	0.020502	0.030503	0.018504

可以看到，'adam' 的效果要稍稍好于'adadelta' 和 'rmsprop'。

3.2.3.2 不同的 dropout 值

```
1.  compare_model_variables('dropout', np.arange(0.1,1,0.1))
```

dropout	Avg	Max	Min
0.1	0.022773	0.046502	0.018423
0.2	0.020875	0.044995	0.017899
0.3	0.021292	0.046482	0.018134
0.4	0.022200	0.047580	0.018439
0.5	0.021597	0.047966	0.018381
0.6	0.022298	0.049096	0.018599
0.7	0.023776	0.050966	0.019483
0.8	0.024109	0.055757	0.019904
0.9	0.028104	0.063775	0.021970

对比 val_loss 的最小值，可以看到 dropout=0.2 的效果是最佳的。

3.2.3.2 不同的 batch_size

```
1.  df = compare_model_variables('dropout', np.arange(0.1,1,0.1))
```

batch_size	Avg	Max	Min
8	0.021103	0.022001	0.020289
16	0.020759	0.025029	0.019336
32	0.019999	0.025877	0.018281
64	0.019947	0.029395	0.018326
128	0.021786	0.045531	0.018243
256	0.026464	0.082115	0.018735
512	0.028462	0.133027	0.018280

随着 batch_size 的增加，收敛速度会加快，但是由于

3.2.3.2 不同的验证集分割比例

validation_split	Avg	Max	Min
0.1	0.026785	0.046564	0.023572
0.2	0.022156	0.047203	0.018543
0.3	0.020213	0.053555	0.016921
0.4	0.022366	0.061218	0.018430
0.5	0.021956	0.066335	0.017545
0.6	0.023609	0.086893	0.017971
0.7	0.025489	0.112159	0.018107
0.8	0.031141	0.148452	0.019859
0.9	0.048170	0.262581	0.021368

四、结果

4.1 模型评估

通过以上对不同参数和优化器的取值对比，选择一组最佳的参数，用于训练模型。

```
1. specify_model_variables(optimizer='adam', dropout=0.2, batch_size=32,
   validation_split=0.3, verbose=1, plot=True)
```

```
Train on 17483 samples, validate on 7493 samples
Epoch 1/30
17483/17483 [==============================] - 2s 134us/step - loss: 0.0695 - acc: 0.9876 - val_loss: 0.0252 - val_acc: 0.9933

Epoch 00001: val_loss improved from inf to 0.02517, saving model to weights.best.xception.hdf5
Epoch 2/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0228 - acc: 0.9941 - val_loss: 0.0197 - val_acc: 0.9947

Epoch 00002: val_loss improved from 0.02517 to 0.01972, saving model to weights.best.xception.hdf5
Epoch 3/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0188 - acc: 0.9947 - val_loss: 0.0178 - val_acc: 0.9947

Epoch 00003: val_loss improved from 0.01972 to 0.01779, saving model to weights.best.xception.hdf5
Epoch 4/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0163 - acc: 0.9952 - val_loss: 0.0169 - val_acc: 0.9945

Epoch 00004: val_loss improved from 0.01779 to 0.01693, saving model to weights.best.xception.hdf5
Epoch 5/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0151 - acc: 0.9957 - val_loss: 0.0166 - val_acc: 0.9952

Epoch 00005: val_loss improved from 0.01693 to 0.01660, saving model to weights.best.xception.hdf5
Epoch 6/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0141 - acc: 0.9959 - val_loss: 0.0169 - val_acc: 0.9944

Epoch 00006: val_loss did not improve from 0.01660
Epoch 7/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0131 - acc: 0.9962 - val_loss: 0.0166 - val_acc: 0.9951

Epoch 00007: val_loss did not improve from 0.01660
Epoch 8/30
17483/17483 [==============================] - 2s 98us/step - loss: 0.0115 - acc: 0.9967 - val_loss: 0.0171 - val_acc: 0.9948

Epoch 00008: val_loss did not improve from 0.01660
Epoch 00008: early stopping
```

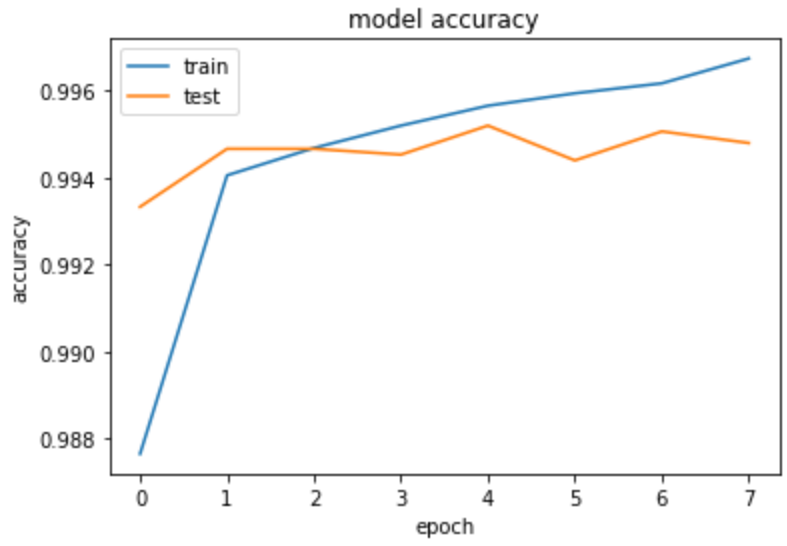
模型在训练过程 8 个 epoch 后停止，在训练第 5 个 epoch 时 val_loss 达到最低，在验证集上准确率为 99.52%，对数损失率为 0.01660。这个结果比未进行参数调优之前，提升了不少。

4.2 模型训练过程可视化

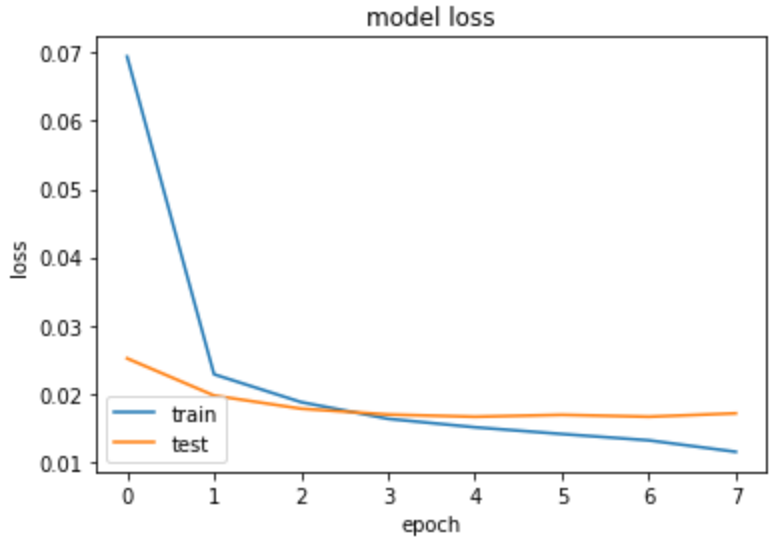
```
def show_learning_curves(history):
    # list all data in history
    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='lower left')
    plt.show()

show_learning_curves(history)
```

训练集与验证集上的准确率：



训练集与验证集上的对数损失率：



可以看出随着迭代次数的增加，训练集上准确率逐渐提升到 99.6%，对数损失率不断下降到0.02以下；而验证集上的上准确率提升到 99.4% 后渐趋平稳，对数损失率下降到 0.02 后也趋于平稳。总体而言，曲线都相对比较平滑。这说明该模型的训练是比较稳定的，既没有出现过拟合，也没有出现欠拟合。

4.3 模型预测与验证

4.3.1 加载模型与权重

由于模型的架构和最佳权重均已保存在到文件，这里可以直接从文件加载。

```
from keras.models import model_from_json
with open('xception.json', 'r') as f:
    model = model_from_json(f.read())
model.load_weights('weights.best.xception.hdf5')
```

4.3.2 预测测试集并保存到文件

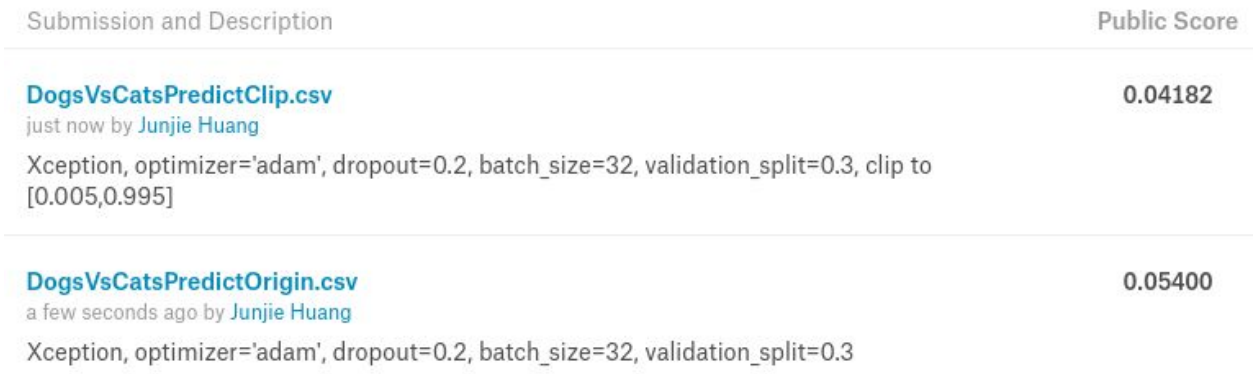
```
y_pred = model.predict(X_test，verbose=1)
y_pred_clip = y_pred.clip(min = 0.005, max = 0.995)

def save_predictions_as_csv(predictions, file_name):
    sv_content = 'id,label\n'
    for index, prob in tqdm(enumerate(predictions)):
        csv_content += '%d,%f\n' % (index + 1, prob)
    with open(file_name, 'w') as f:
        f.write(csv_content)

save_predictions_as_csv(y_pred, 'DogsVsCatsPredictOrigin.csv')
save_predictions_as_csv(y_pred_clip, 'DogsVsCatsPredictClip.csv')

```

这里借鉴了老师提供的一个小技巧：将预测值限制到了 [0.005, 0.995] 区间内。原因是 kaggle 官方的评估标准是 LogLoss，对于预测正确的样本，0.995 和 1 相差无几，但是对于预测错误的样本，0 和 0.005 的差距非常大。这个技巧使让我的得分由 0.05400 提升到了 **0.04182**，在 Public Leaderboard 上排名为 **20/1314**，到达了 kaggle 前 10% 的项目目标。



4.3.3 图片预测结果及可视化

这里随机展示一些测试集中的图片的预测结果，并将其可视化输出。

```
def show_random_train_images(title):
    img_size = (299, 299)
    np.random.seed(20181018)
    randarr = np.random.randint(low=0, high=12500, size=16)
    scorearr = []
    indexarr = []
    with open('DogsVsCatsPredictClip.csv', 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            if row[0] != 'id' and int(row[0]) in randarr:
                indexarr.append(int(row[0]))
                scorearr.append(row[1])
    print(len(scorearr))
    patharr = ["data/test/%s.jpg" % i for i in indexarr]
    plt.style.use('seaborn-white')
    fig, ax = plt.subplots(4, 4, sharex='col', sharey='row', figsize=(12, 12))
    #fig.canvas.set_window_title(title)
    fig.suptitle(title, fontsize=16)
    for i in range(4):
        for j in range(4):
            index = i * 4 + j
            path = patharr[index]
            img = image.load_img(path, target_size=img_size)
            ax[i,j].set_title(os.path.basename(path) + '\n' + scorearr[index] )
            ax[i,j].imshow(img)

show_random_train_images('random images in train set')
```

随机展示 16 张测试集图片及其预测结果，可以图片基本都正确分类。

五、结论

5.1 思考

通过完成本项目，对迁移学习和神经网络有了更深入的了解，同时对 keras、matplotlib 等相关工具库的使用也更加熟悉，python 编码能力有了一定提升。

本文选取 Xception 模型并使用其在 ImageNet 数据上的权重，思路是没有问题的。但是一开始直接在 Xception 后面接入全连接层，导致每次训练都，既耗时又耗力，同时效果也不怎么好，出现了严重的过拟合。后来在老师的提示下，采用保存特征权重的方法，既提升了训练的效率，也提升了训练的效果。

在使用 Xcepiton 模型结合 ImageNet Top-N 预测结果剔除异常值的过错中，发现模型对异常值检测准率率没有想象的那么高。一共检出 34 个异常值，其中有 10 个是错误的。这同时应该还有不少异常值没有被检测出来。这说明单个模型存在着死角。更好的方式就是同时使用多个欲训练模型来同时检测，只要有一个模型判定其为异常值，就将其加入异常值列表，最后利用可视化工具呈现，再按照本文的方法通过人工校验筛选，这样应该可以识别出更多的异常值。

通过尝试使用不同的优化器以及 dropout，batch_size，validation_split 等超参数，分别比较其不同取值对验证集误差的影响，选取了一组最优参数，使得最终的 LogLoss 有了较大的改善。这说明参数的调节对于神经网络模型来说至关重要，决定模型最终的泛化能力。怎样快速选取一些合理的参数值让模型能有一个不错的初始效果，怎样快速找到影响较大的参数并超着最优的方向调节，这不仅需要对机器学习基础知识透彻理解，还需要大量的工程实践。

由于时间有限，很多想法暂时还未能一一尝试验证，后续会还会继续跟进。

5.2 改进

后续改进的主要途径：

- 尝试将多个模型进行融合以提升效果
- 尝试更小的学习率
- 深入的调查数据增强相关的参数组合

5.3 鸣谢

项目的目标虽暂时达到，但是个人在机器学习这条路上才刚刚开始。在这里要感谢 Udacity 的在线课程帮我开启了通往新世界大门，感谢各位 mentor 对我的指导和鼓励，帮我通过了前期的各项关卡，让一个野生程序员找到了家的感觉。最后感谢我的妻子和刚满三个月的儿子，你们给了我在深夜抽空训练模型完成项目的动力。

参考文献

1. 《TensorFlow - 实战 Google 深度学习框架》（第2版）.郑宇才、梁博文、顾思宇 著.
2. 《Deep Learning with Keras》[Italy] Antonio Gulli. [India] Sujit Pal
3. Diederik P.Kingma, Jimmy. Ba.Adam: A Method for Stochastic Optimization
4. 手把手教你如何在Kaggle猫狗大战冲到 Top2 杨培文
5. 毕业设计 Dogs vs Cats For Udacity P7 (异常值检验)
6. 基于Pre-trained模型加速模型学习的6点建议 lqfarmer