机器学习纳米学位-猫狗大战

朱成亮

2019 年 02 月 16 日

I. 问题的定义

项目概述

本项目源自 Kaggle 平台上的一个竞赛项目，目的是训练出一个模型来分辨图像中的猫和狗，属于计算机视觉领域。

近年来随着深度学习的研究发展，在图像处理领域取得了很大的成功，基于 CNN(卷积神经网络)构建的深度学习模型(比如 Xception，VGG19，NASNetLarge 等模型)对于图像识别分类有很高的准确率，可以很好的解决此问题。

卷积神经网络是一种多层神经网络，擅长处理图像的相关机器学习问题。CNN 通过一系列方法，成功将数据量庞大的图像识别问题不断降维，最终使其能够被训练。

CNN 网络中的卷积层和池化层主要是对图片的特征进行抽取，比如浅层的卷积层和池化层先抽取图片的一些简单几何信息，比如边界、直线和角点等，深层的卷积层和池化层抽取复杂的抽象信息如猫狗特征的信息，最后全连接层是对分类的处理。

重新设计和训练一个好的模型不仅需要很强的理论和实践能力，而且还要耗费大量的时间和精力，对目前的我来说不具备可行性。因此本项目会使用已经在 ImageNet 训练好的模型 Inception V3 作为基准模型，然后通过迁移学习的方案来解决本项目图片二分类问题。

项目工具选择 Keras，Keras 是一个很流行的深度学习框架，Keras 的开发重点是支持快速的实验，能够以最小的时延把你的想法转换为实验结果，是做好研究的关键。而且 Keras Applications 模块提供了预先训练的 Inception V3 模型。

项目的数据集来源于 Kaggle 竞赛的数据，此数据集可以从 Kaggle 上下载 Dogs vs. Cats Redux: Kernels Edition。

问题陈述

本项目是一个典型的图片二分类问题，即输入一张图片，系统需要判别其属于已知两个分类狗和猫中的哪一类。

输入：一张彩色图片

输出：是猫还是狗

本项目会使用已经在 ImageNet 训练好的模型 Inception V3 作为基准模型，构建一个新模型来读入图片，并判断图片中是狗的概率。

本项目还有以下一些细节问题需要解决：

- Kaggle 提供的数据集中有少部分图片是异常的，比如分辨率太低，或者图片不是猫狗的图片，训练集中需要剔除这些异常图片。
- 需要将训练集中不同类别的图片分别放在不同的文件夹 cat 和 dog 中。
- 数据集中的图片大小不是固定的，但是 CNN 输入节点个数是固定的，所以图片输入前，需要对图片进行 resize 操作。

评价指标

对数损失，即对数似然损失(Log-likelihood Loss)，也称逻辑回归损失(Logistic Loss)或交叉熵损失(Cross-entropy Loss)。它度量真实条件概率分布和假定条件概率分布之间的差异，是分类问题中常用的一种损失函数。

Kaggle官方评估标准：LogLoss=$-\frac{1}{n}\sum_{i=1}^{n}[y_i\log(\hat{y}_i)+(1-y_i)\log(1-\hat{y}_i)]$,

n 测试集图片数量；

$\hat{y}_i$ 图片预测为狗的概率；

y_i 如果是狗则为1，是猫则为0；

log() 自然对数；

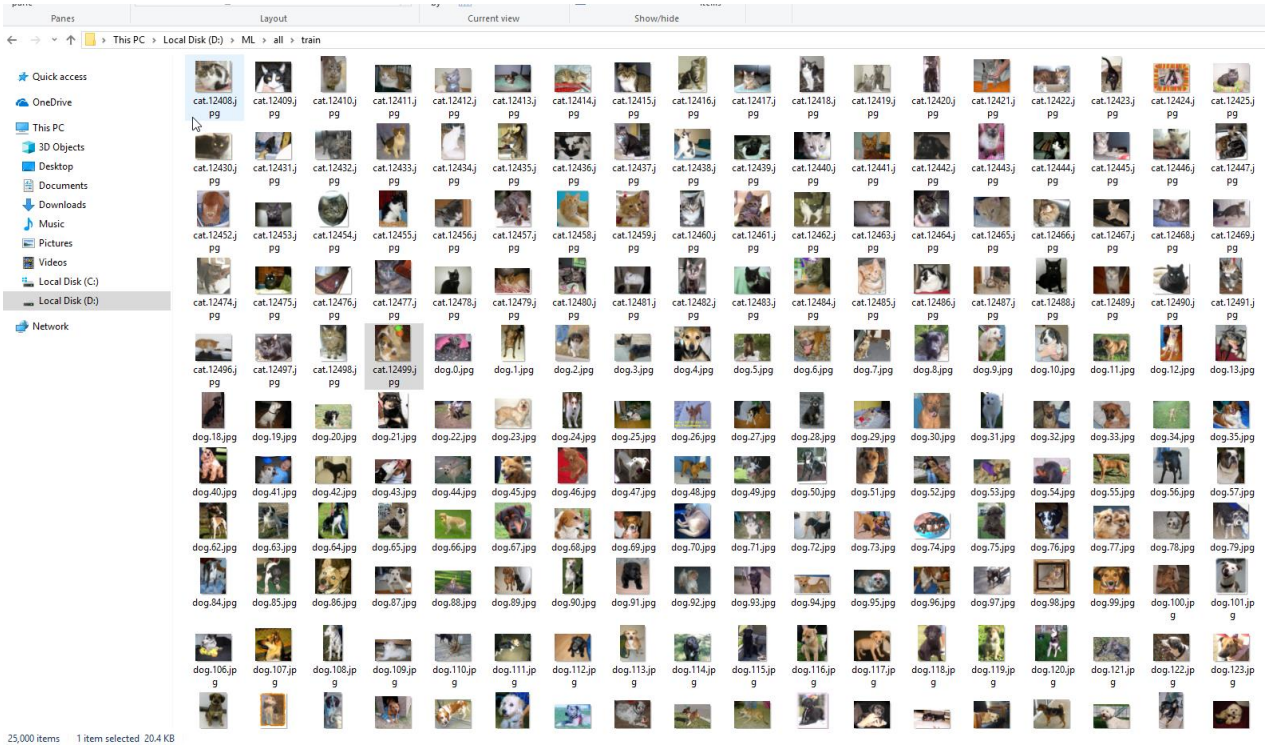
对数损失越小，代表模型越好。

本项目最低要求是Kaggle Public Leaderboard 前10%，按目前提交的数据LogLoss要低于0.06127。

II. 分析

数据的探索

从Kaggle下载的数据有训练集train.zip、测试集test.zip以及测试结果表格sample_submission.csv。



In [40]:
```
# 训练集
image_files = os.listdir(DIR_IMG_TRAIN)
cats = [img for img in filter(lambda x:x[:3] == 'cat', image_files)]
dogs = [img for img in filter(lambda x:x[:3] == 'dog', image_files)]
total = np.concatenate((cats,dogs))

print("Number of Dat Images: {0}".format(len(cats)))
print("Number of Dog Images: {0}".format(len(dogs)))

x =np.array([len(cats),len(dogs)])
plt.bar(['CAT','DOG'],[x[0],x[1]])
plt.xlabel('Category')
plt.ylabel('COUNT')
plt.title('Category and Image count of Train data')
plt.show()
```

```
Number of Dat Images: 12500
Number of Dog Images: 12500
```

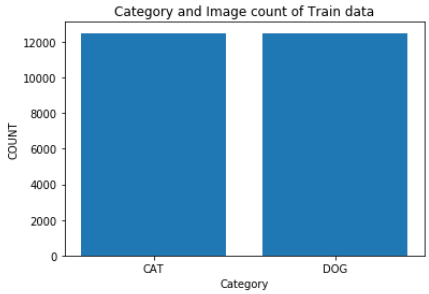


图1　训练集和测试集

- 该数据集中训练集有25000张已经标记好的猫和狗的图片，猫狗各一半；标记是图片文件名的一部分type_number.jpg，编号从0开始，例如cat.0.jpg，如图1所示。
- 测试集有 12500 张未被标记的图片，图片文件名是以数字命名，编号从 1 开始，例如 1.jpg，如图 1 所示。
- 随机观察了训练集中的部分图片，有少部分异常图片，需要从训练集中剔除这些图片，如图 2 所示（通过人工和程序随机查看探索图片）。
- sample_submission.csv，最终将测试集的预测结果写入此文件，上传至 Kaggle 进行打分。
- 图片中的场景各不相同，一般拍自生活场景，如图 2 所示。图片的有和人的合影，光照条件和模糊度等也不同。

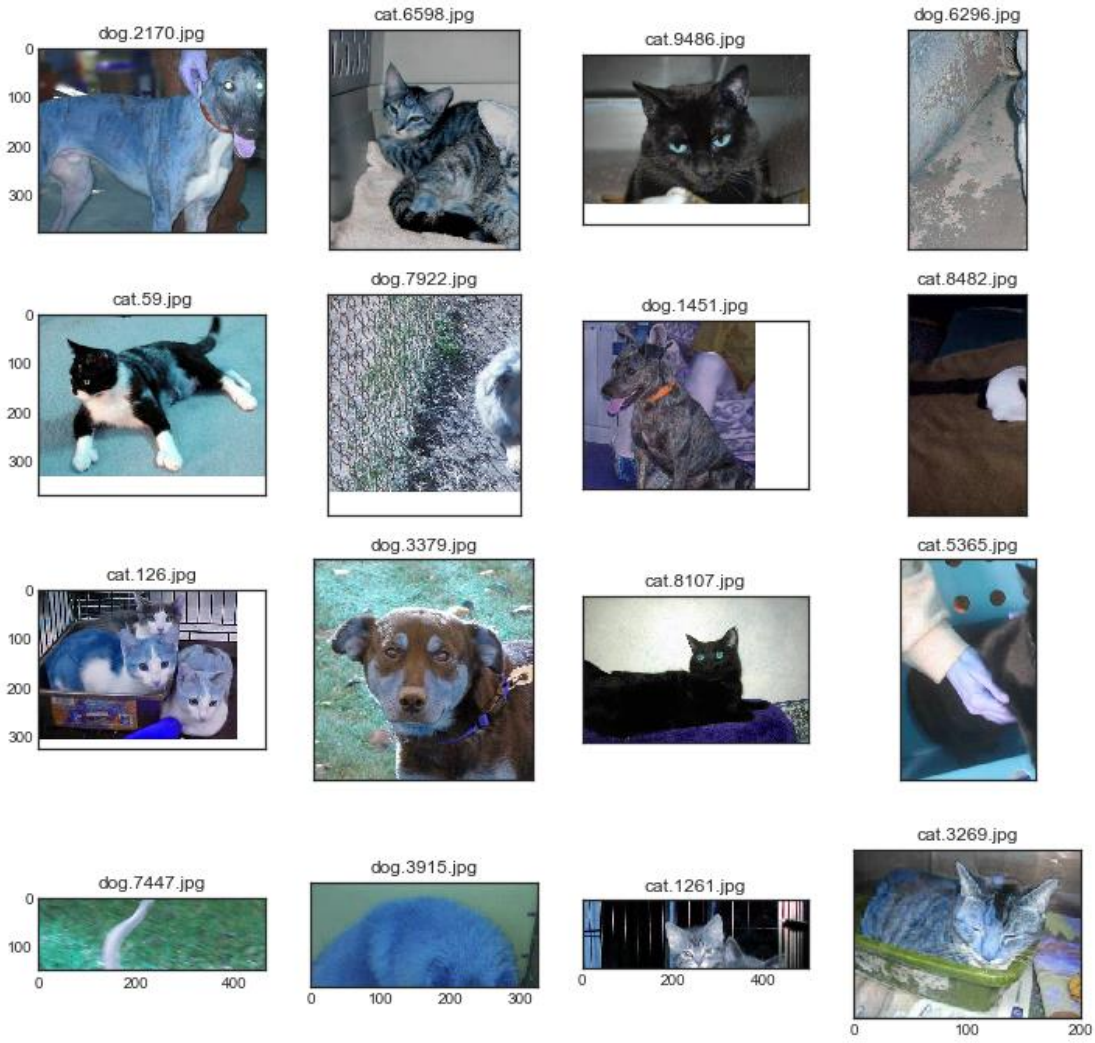
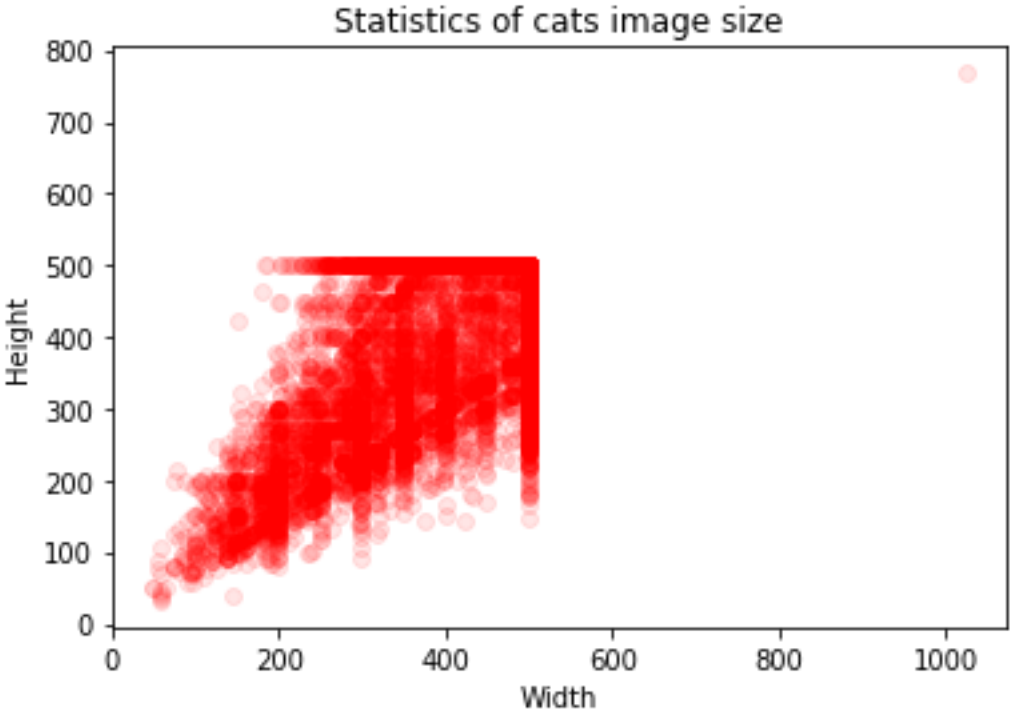


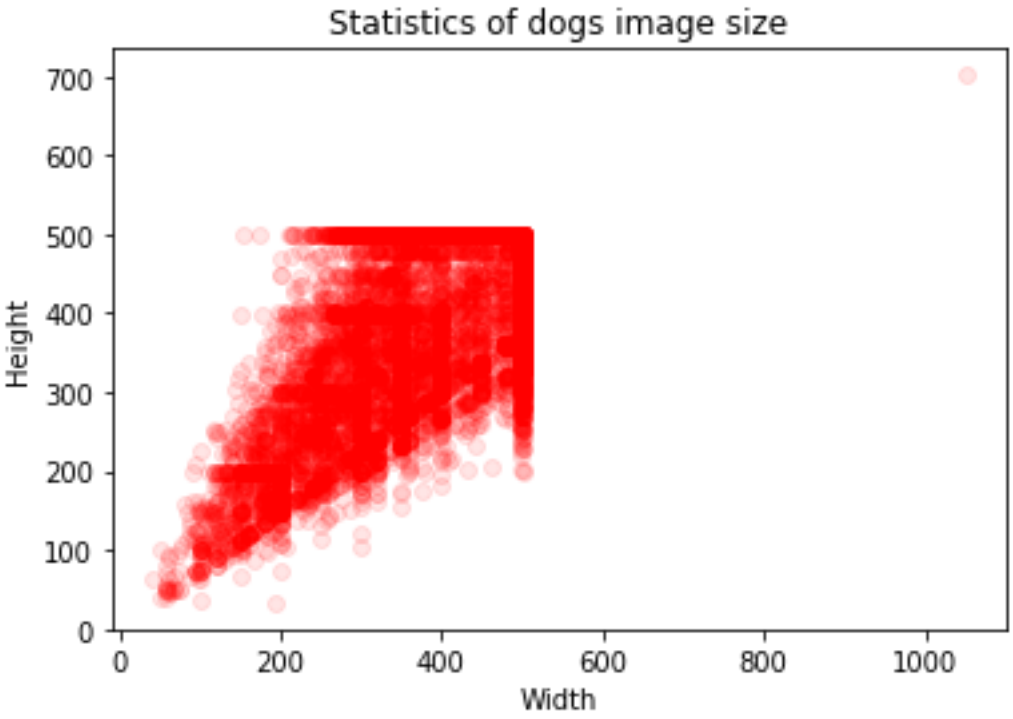
图 2 异常图片和不同图片场景

探索性可视化

图片的尺寸大小分布以及异常图片的分析处理是非常重要的，本节主要从这两方面可视化探索数据集。

图3显示的分别是训练集和测试集中图片的宽高尺寸三点图。


Statistics of cats image size
Height
Width
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000


Statistics of dogs image size
Height
Width
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

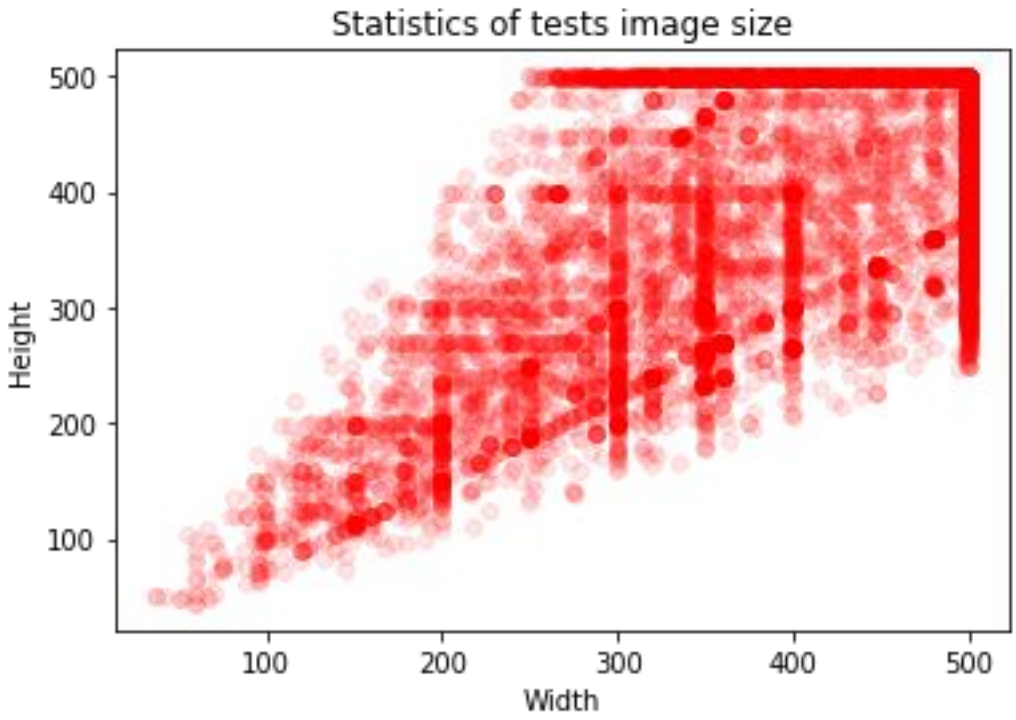


图3 训练集和测试集图片尺寸

从以上图3可以看出训练集和测试集中的图片尺寸不一致，主要分布在宽(200~500px)，高(200~500px)。后续在模型训练前需要对图片size统一进行预处理。

图4是对通过对训练集中图片的色彩-像素比进行 IQR 分析。

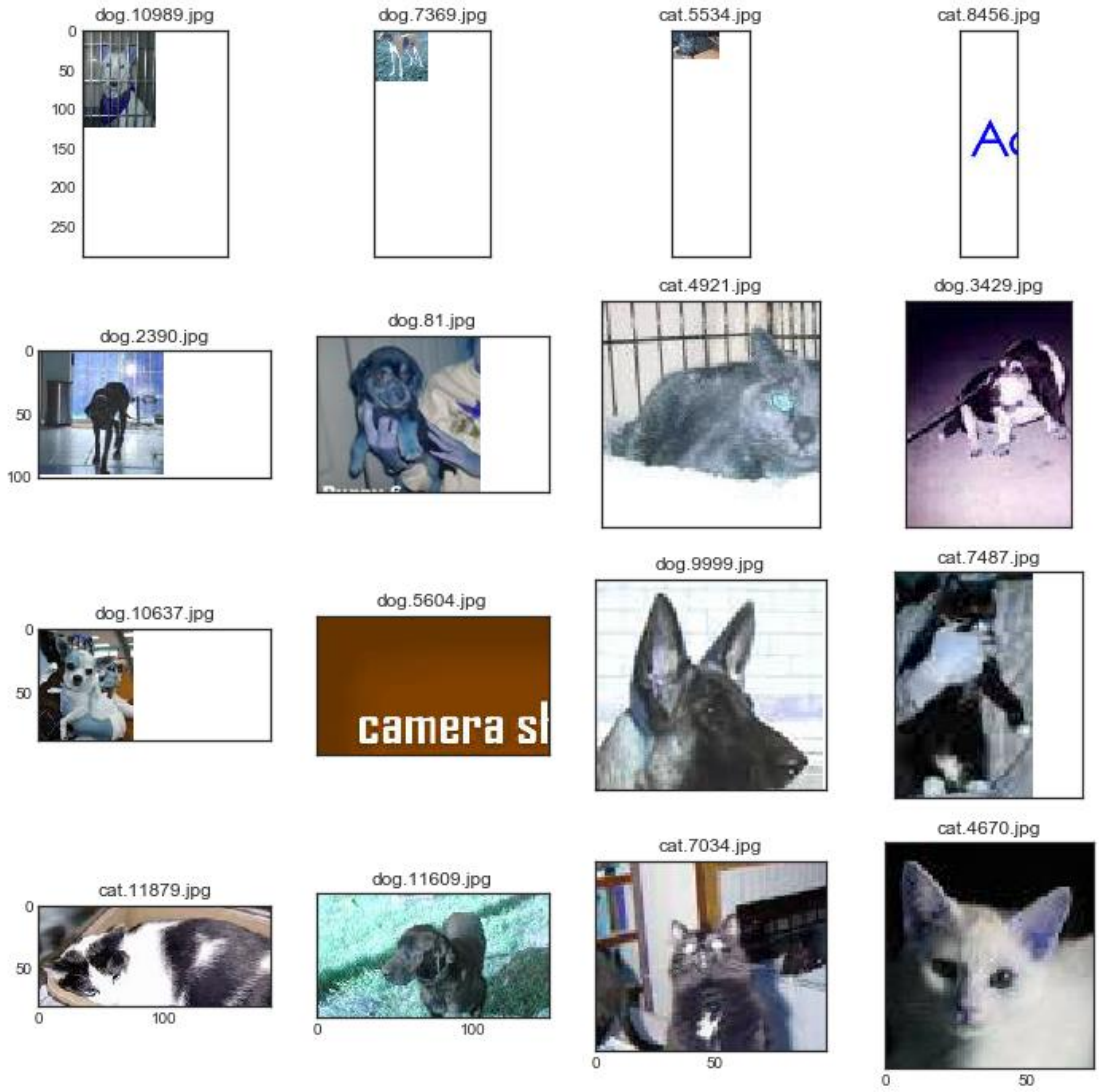


图4 IQR分析

通过分析结果可以发现很多分辨率低、无关的图片，因此异常图片可能需要在模型训练前从训练集中剔除掉。不过在数据预处理中，本项目会直接使用预训练的Inception V3识别异常图片。

算法和技术

本项目是典型的图片二分类问题，在计算机视觉领域，目前解决这类问题的核心技术框架是深度学习（Deep Learning），近年来随着计算机硬件和数据的发展，深度学习取得显著的发展，尤其是CNN更是图片识别领域中最具代表性的算法。

1. 卷积神经网络(CNN)

CNN在图像识别上的优势主要体现为：

- 可以缩减计算过程中的权重数量；
- 有容错能力；
- 有自动学习，特征归纳的特性；
- 不受物体在图片中位置的影响；

通常一个卷积神经网络是由输入层（Input）、卷积层（Convolution）、池化层（Pooling）、全连接层（Fully Connected）组成。

在输入层输入原始数据，卷积层中进行的是卷积过程，用它来进行提取特征；全连接层就是将识别到的所有特征全部连接起来，并输出到分类器（如Sigmoid）。

输入层 - Input Layer

输入层是整个CNN网路的输入。在处理图像时，输入层是图片解析而成的图片像素值的多维矩阵。如果是RGB图片，维度为3，如果是灰度图片，维度为1；

卷积层 - Convolutional Layer

卷积层主要用来进行卷积运算。对于输入的数据，卷积运算以一定的间隔(stride)滑动图像过滤器的窗口，将各个位置上的滤波器元素和窗口的输入元素相乘在求和，并将结果保持到对应位置。

扫描完整张图片后，就会得到图片的一个特征，因此过滤器越多，得到的特征就越多。

因此卷积层的主要作用是对图片进行特征提取，我们通常会使用多层卷积层来得到更深层次的特征图。

ReLU Layer- Rectified Linear Units

在每一个卷积层后都会有一个非线性层，该层的主要目的是系统中引入非线性特征，一般使用非线性函数ReLU作为激活函数。ReLU函数执行的操作是对输入应用max(0,x)操作，对于所有小于0的输入，输出为0。

池化层 - Pooling Layer

在ReLU层后一般为池化层，即降采样层。池化方式主要有最大池化（取出最大值）、平均池化（计算平均值）等方式。

Dropout Layer

Dropout层，是为了防止过度拟合问题。在学习过程中，每一次传递信号，会随机删除神经元，被删除的神经元不进行此次信号的传递，以此抑制过拟合。

全连接层 - Fully Connected Layer

全连接层一般位于CNN网路的最后，往往会有一两个全连接层，该层的输入可以是卷积层、ReLU层和池化层的结果，输出一个n维向量，即n个分类结果。

2. 迁移学习

迁移学习（Transfer Learning），就是能让现有的模型算法稍加调整即可应用于一个新的领域和功能的一项技术。迁移学习能够将适用于大数据的模型迁移到小数据上，实现个性化迁移。迁移学习一般有四种实现方法:

样本迁移（Instance-based Transfer Learning）

样本迁移即在数据集（源领域）中找到与目标领域相似的数据，把这个数据放大多倍，与目标领域的数据进行匹配。其特点是：需要对不同例子加权；需要用数据进行训练。

特征迁移（Feature-based Transfer Learning）

特征迁移是通过观察源领域图像与目标域图像之间的共同特征，然后利用观察所得的共同特征在不同层级的特征间进行自动迁移。

模型迁移（Model-based Transfer Learning）

模型迁移利用上千万的图象训练一个图象识别的系统，当我们遇到一个新的图象领域，就不用再去找几千万个图象来训练了，可以原来的图像识别系统迁移到新的领域，所以在新的领域只用几万张图片同样能够获取相同的效果。模型迁移的一个好处是可以和深度学习结合起来，我们可以区分不同层次可迁移的度，相似度比较高的那些层次他们被迁移的可能性就大一些。

关系迁移（Relational Transfer Learning）

把相似的关系进行迁移，比如生物病毒传播到计算机病毒传播的迁移，比如师生关系到上司下属关系的迁移。

3. Inception V3

2012 年 AlexNet 做出历史突破以来，直到 GoogLeNet 出来之前，主流的网络结构突破大致是网络更深（层数），网络更宽（神经元数），但是纯粹的增大网络的缺点：

a) 参数太多，若训练数据集有限，容易过拟合；

b) 网络越大计算复杂度越大，难以应用；

c) 网络越深，梯度越往后穿越容易消失，难以优化模型。

那么解决上述问题的方法当然就是增加网络深度和宽度的同时减少参数，Inception 就是在这样的情况下应运而生。

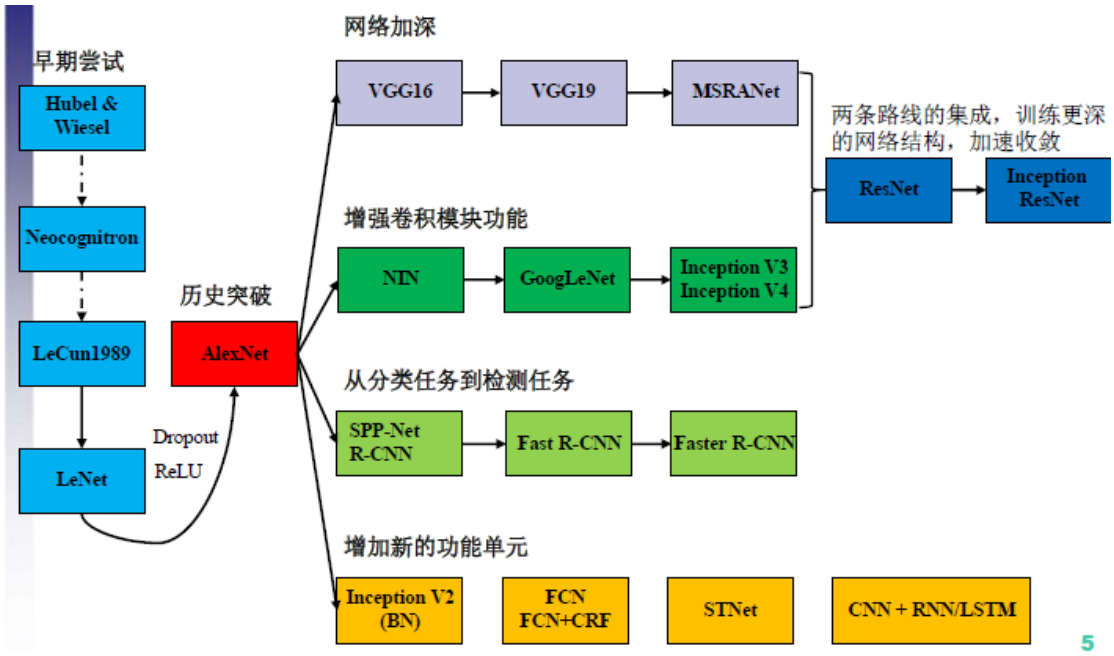


图 5 CNN 结构演化图

Google Inception Net 在 2014 年的 ImageNet Large Scale Visual Recognition Competition (ILSVRC) 中取得第一名，该网络以结构上的创新取胜，通过采用全局平均池化层取代全连接层，极大的降低了参数量，是非常实用的模型，一般称该网络模型为 Inception V1。随后的 Inception V2 中，引入了 Batch Normalization 方法，加快了训练的收敛速度。在 Inception V3 模型中，通过将二维卷积层拆分成两个一维卷积层，不仅降低了参数数量，同时减轻了过拟合现象。图 6 是 Inception V3 的结构图。

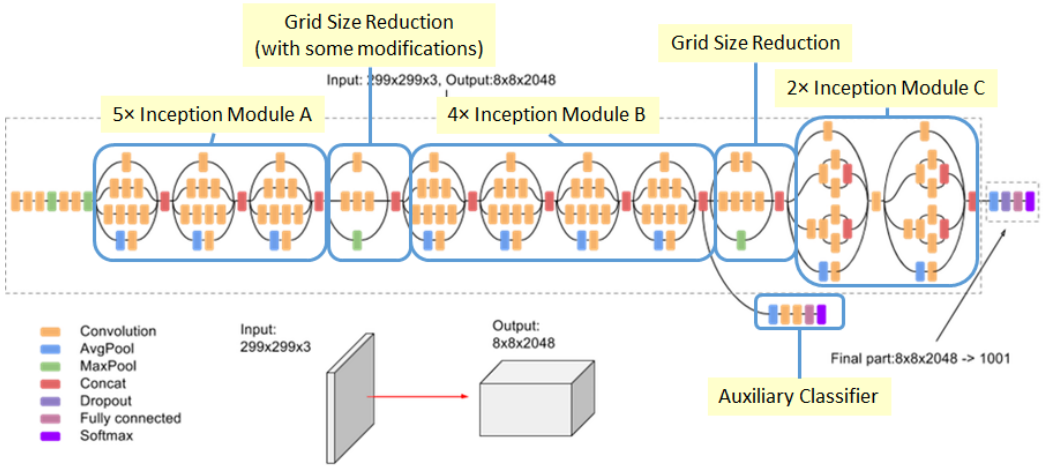


图 6 Inception V3 结构图

4. Adam优化器

2014 年 12 月，Kingma 和 Lei Ba 两位学者提出了 Adam 优化器，结合了 AdaGrad 和 RMSProp 两种优化算法的优点。

1. 实现简单，计算高效，对内存需求少
2. 参数的更新不受梯度的伸缩变换影响
3. 超参数具有很好的解释性，且通常无需调整或仅需很少的微调
4. 更新的步长能够被限制在大致的范围内（初始学习率）
5. 能自然地实现步长退火过程（自动调整学习率）
6. 很适合应用于大规模的数据及参数的场景
7. 适用于不稳定目标函数
8. 适用于梯度稀疏或梯度存在很大噪声的问题综合

Adam 在很多情况下算作默认工作性能比较优秀的优化器。

本项目会使用已经在 ImageNet 训练好的模型 Inception V3 作为基准模型，通过迁移学习（属于模型迁移）的方案来解决图片二分类问题。主要工具选择 Keras，以 Tensorflow 作为后端。Keras 是一个很流行的深度学习框架，而且 Keras Applications 模块提供了在 ImageNet 上预训练的 Inception V3 模型。

基准模型

图像识别在工业界已经有非常突出的研究成果和应用，近几年来产生了一些非常优秀的CNN的模型，比如Inception V3，Xception，VGG19，NASNetLarge 等。本项目将会使用Inception V3作为基准模型。

本项目最低要求是Kaggle Public Leaderboard 前10%，按目前提交的数据 LogLoss要低于0.06127。

III. 方法

数据预处理

本文使用ImageNet预训练的预处理模型Inception V3 Top-30对训练集的图片进行识别。

Keras预处理模型Inception V3是在ImageNet上预训练的，有1000个分类结果。本项目只需要对猫和狗进行分类，以达到识别出是否是猫或狗图片，因此需要先获取1000个分类中的猫和狗的分类编号保存下来。

使用Keras Inception V3模型，weights设置为”ImageNet”，top使用30（综合比较过10和50的结果，最终选择30的结果比较合适），对train里面的所欲图片进行识别，然后把识别结果的30个值和提前保存下来的猫狗分类编号比较，确认预测出的30个值是否有包含在猫狗分类编号列表里；如果包含，则说明是正常图片，否则认为是异常图片。

异常图片识别的具体步骤如下：

1. 获取ImageNet的1000个分类列表及其编号，并把其中是猫和狗的编号保存到列表中（总共有125个），以便在后续识别中判断是不是猫或狗。

   ['n02085620', 'n02085782', 'n02085936', 'n02086079' ……]

2. 定义并创建目录结构以便后面模型训练使用。定义的目录结构如下：

```
-- data
    -- review
    -- train
    -- test
    -- train_new
        -- cats
        -- dogs
    -- valid
        -- cats
        -- dogs
```

3. 定义Keras Inception V3 model，weights设置为”ImageNet”，top使用30，如图7。

```
In [42]: # InceptionV3 模型
         from keras.applications.inception_v3 import InceptionV3
         from keras.preprocessing import image
         from keras.applications.inception_v3 import preprocess_input
         from keras.applications.inception_v3 import decode_predictions

         # 定义InceptionV3模型
         InceptionV3_model = InceptionV3(weights='imagenet')

         WARNING:tensorflow:From /Users/ext.richie.zhu/anaconda3/envs/deeplearning/lib/python3.7/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
         Instructions for updating:
         Colocations handled automatically by placer.
```

图7 Inception V3定义

3. 定义一个识别图片的方法，传入参数是：要被识别的图片集目录，模型，图片尺寸，top值。把识别结果值和提前保存下来的猫狗分类编号比较，看是否有包含在猫狗分类编号列表里；如果包含则是正常图片，否则认为是异常图片。

4. 除了识别是否是猫狗图片之外，同时也会把图片根据识别结果将是狗、猫的图片放到将要用来训练的dogs或cats目录里，把未识别出狗、猫的图片放到review目录里。步骤3和4的具体实现如图8所示。

In [136]:
```
def invalid_cat_dog_detector(file_dir, model, img_size, top=10):
    invalid_files = []

    files_tmp = os.listdir(file_dir)
    for file in tqdm(files_tmp[:]):
        if not (file.endswith('JPG') or file.endswith('jpg')):
            continue

        file_full_path = os.path.join(file_dir, file)
        img = image.load_img(file_full_path, target_size=img_size)
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)

        preds = model.predict(x)
        predictions = decode_predictions(preds, top=top)[0]

        isValid = False
        for p in predictions:
            if p[0] in dog_cat_classes:
                isValid = True
                break

        if isValid:
            if file[:3] == 'cat':
                shutil.copy(file_full_path, DIR_TRAIN_NEW_CAT)
            if file[:3] == 'dog':
                shutil.copy(file_full_path, DIR_TRAIN_NEW_DOG)
        else:
            invalid_files.append(file)
            shutil.copy(file_full_path, DIR_REVIEW)

    return invalid_files
```

In [137]:
```
invalid_files = invalid_cat_dog_detector(DIR_TRAIN, InceptionV3_model, IMG_SIZE, 30)
```

图8 异常处理方法

最终识别出43张异常图片，异常图片如图9所示。

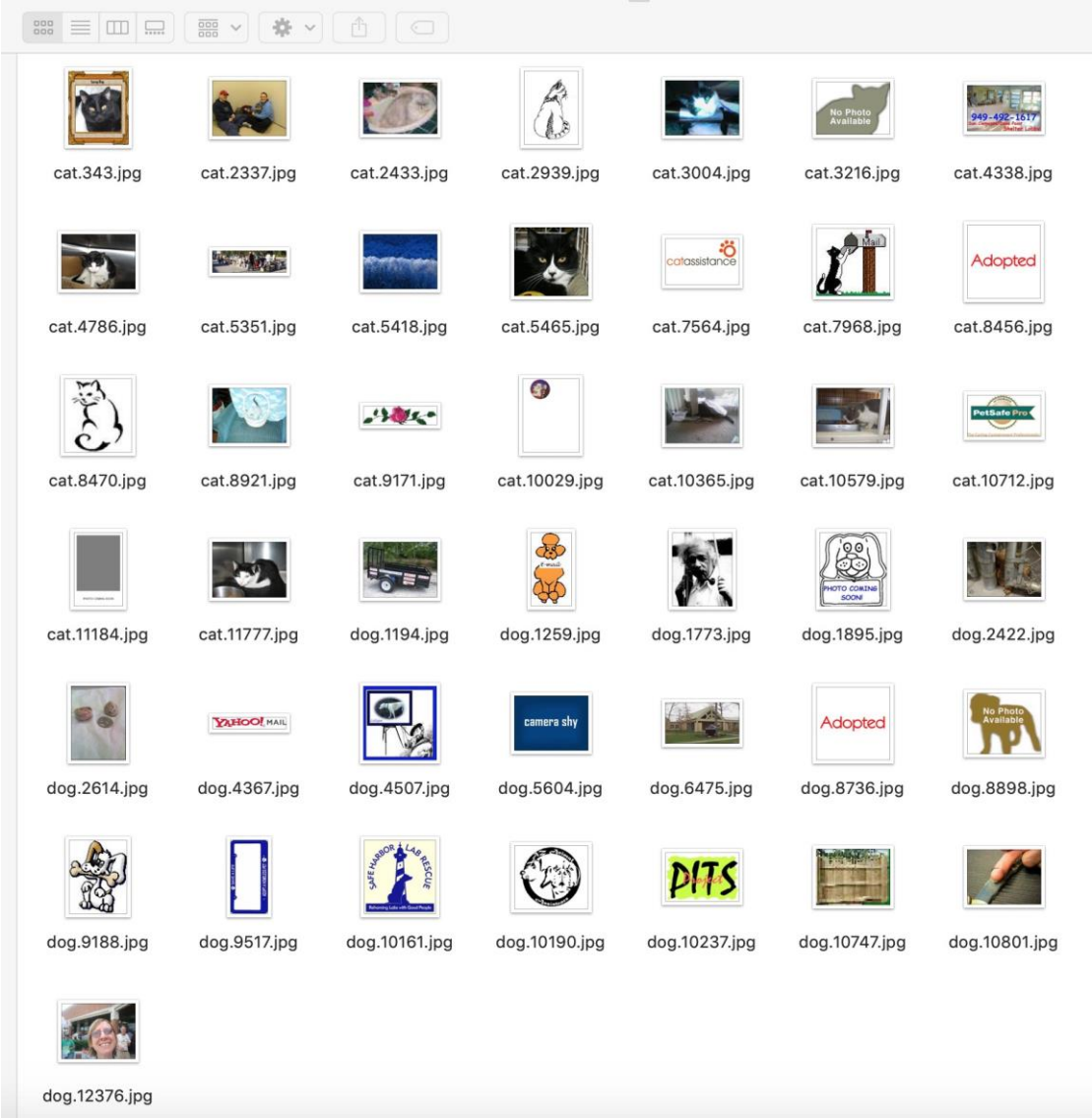


图9 Inception V3 top-30识别的异常图片

4. 需要从准备好的训练集中抽出20%的图片作为验证集放到valid目录里。因为图片内容本身是无序的，所以可以直接分别把训练集train_new里dogs和cats中的各2450个图片放到验证集valid下dogs和cats目录里。
5. 考虑到AWS Instance的成本问题，本项目中非猫、狗图片的清理是在本地机器做的，将问题图片复制到review目录里面。

执行过程

文件夹结构和图片预处理在本地机器经过多次试验处理后，开始迁移学习的模型构建和训练。本项目使用的是 AWS EC2 p2.xlarge GPU 机器训练，平均

每个 epoch 需要 15 分钟，而我的 PC CPU 训练，每个 epoch 则需要 3 个小时左右。具体步骤如下：

1. 配置 AWS Instance 环境

安装Keras和Tensoflow-GPU并根据数据预处理阶段整理好的文件夹结构，在AWS instance上创建相应的文件夹目录。

配置jupyter notebook服务并上传jupyter notebook训练文件。

通过kaggle API下载训练和测试集，因为本地上传这部分数据很慢。

2. 整理数据文件

把本地识别出的异常图片上传到AWS instance上问review目录，然后遍历下载的kaggle train文件夹里猫、狗文件，然后判断是否是在review里，如果不在review里则说明是正常图片，然后给分配到新的cats和dogs文件夹里，为后面训练使用。关键实现代码如图10所示。

```
files = os.listdir(DIR_TRAIN)
cats = [img for img in filter(lambda x:x[:3] == 'cat', files)]
dogs = [img for img in filter(lambda x:x[:3] == 'dog', files)]

reviewFiles = os.listdir(DIR_REVIEW)

for f in tqdm(cats):
    if f not in reviewFiles:
        src = os.path.join(DIR_TRAIN, f)
        shutil.copy(src, DIR_TRAIN_NEW_CAT)

for f in tqdm(dogs):
    if f not in reviewFiles:
        src = os.path.join(DIR_TRAIN, f)
        shutil.copy(src, DIR_TRAIN_NEW_DOG)
```

```
100%|██████████| 12500/12500 [00:01<00:00, 10106.93it/s]
100%|██████████| 12500/12500 [00:01<00:00, 9754.35it/s]
```

```
cat_files = [img for img in filter(lambda x:x[:3] == 'cat', os.listdir(DIR_TRAIN_NEW_CAT))]
dog_files = [img for img in filter(lambda x:x[:3] == 'dog', os.listdir(DIR_TRAIN_NEW_DOG))]
print(len(cat_files)) # 20% 约等于 2450 个
print(len(dog_files))
```

```
12477
12480
```

图 10 AWS 上根据本地结果处理异常图片

3. 增强数据集

使用keras里ImageDataGenerator函数对数据进行增强处理，增强模型的泛化能力。比如对图片进行平移、旋转和缩放等。

4. 归一化处理

图片在计算机中一般是由RGB三种颜色表示的，每种颜色的取值范围在[0,255]，但是对于神经网络来说，直接输入这么大范围的离散数字非常不利于收敛，因此一般在输入网络之前会对结果进行相应的归一化。（评阅老师评语）

当输入数值不符合默认要求时，使用每个模型的预处理函数preprocess_input即可将输入图片处理成该模型的标准输入，Keras中preprocess_input()函数完成数据预处理的工作，inception模型中的预处理函数的操作是过对数据各个维度上的值进行重新调节，将图片的值归一化到[-1,1]的区间，这样可以加速收敛。

数据增强和归一化处理的关键代码如图11所示。

```
In [12]: train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                            rotation_range=40,
                                            width_shift_range=0.2,
                                            height_shift_range=0.2,
                                            shear_range=0.2,
                                            zoom_range=0.2,
                                            channel_shift_range=10,
                                            horizontal_flip=True,
                                            fill_mode='nearest')
         train_batches = train_datagen.flow_from_directory(DIR_TRAIN_NEW,
                                                           target_size=IMAGE_SIZE,
                                                           interpolation='bicubic',
                                                           class_mode='categorical',
                                                           shuffle=True,
                                                           batch_size=BATCH_SIZE)

         valid_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
         valid_batches = valid_datagen.flow_from_directory(DIR_VALID_NEW,
                                                           target_size=IMAGE_SIZE,
                                                           interpolation='bicubic',
                                                           class_mode='categorical',
                                                           shuffle=False,
                                                           batch_size=BATCH_SIZE)

         # show class indices
         print('****************')
         for cls, idx in train_batches.class_indices.items():
             print('Class #{} = {}'.format(idx, cls))
         print('****************')

         Found 20057 images belonging to 2 classes.
         Found 4900 images belonging to 2 classes.
         ****************
         Class #0 = cats
         Class #1 = dogs
         ****************
```

图 11 数据加载和预处理

5. 构建模型

a) Dropout 一般设置为 0.5，防止过拟合；

b) 激活函数使用 softmax，归一化指数函数，使得每一个元素的范围都在（0,1）之间，并且所有元素的和为 1。

c) input_shape，Inception V3 输入 size 是 299X299；

d) 迁移学习主要是利用预训练的模型中的结果，本项目中主要是重复利用图片特征提取的各部分，所以这些层不用重新训练。

e) 优化器选择 Adam，在算法技术一章已详细介绍。

f) 损失函数使用 categorical_crossentropy,和 softmax 配对使用。

关键代码如图 12 所示。

```
net = InceptionV3(include_top=False, weights='imagenet', input_tensor=None,
                  input_shape=IMAGE_SIZE)
x = net.output
x = Flatten()(x)
x = Dropout(0.5)(x)
output_layer = Dense(NUM_CLASSES, activation='softmax', name='softmax')(x)
net_final = Model(inputs=net.input, outputs=output_layer)
for layer in net_final.layers[:FREEZE_LAYERS]:
    layer.trainable = False
for layer in net_final.layers[FREEZE_LAYERS:]:
    layer.trainable = True
net_final.compile(optimizer=Adam(lr=1e-5),
                  loss='categorical_crossentropy', metrics=['accuracy'])
print(net_final.summary())
```

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_3 (InputLayer)         (None, 299, 299, 3)       0
__________________________________________________________________________________________________
conv2d_188 (Conv2D)          (None, 149, 149, 32)      864         input_3[0][0]
__________________________________________________________________________________________________
batch_normalization_188 (BatchN (None, 149, 149, 32)   96          conv2d_188[0][0]
__________________________________________________________________________________________________
activation_188 (Activation)  (None, 149, 149, 32)      0           batch_normalization_188[0][0]
__________________________________________________________________________________________________
conv2d_189 (Conv2D)          (None, 147, 147, 32)      9216        activation_188[0][0]
__________________________________________________________________________________________________
batch_normalization_189 (BatchN (None, 147, 147, 32)   96          conv2d_189[0][0]
__________________________________________________________________________________________________
activation_189 (Activation)  (None, 147, 147, 32)      0           batch_normalization_189[0][0]
__________________________________________________________________________________________________
conv2d_190 (Conv2D)          (None, 147, 147, 64)      18432       activation_189[0][0]
```

图 12 构建模型

6. 训练模型

对训练集进行分割，20%作为验证集，80%作为训练集；训练结果保存为 model-InceptionV3-final.h5。

因为图片内容本身是无序的，所以可以直接从训练集里面随便选2450个图片放到验证集里面。

```
In [11]: # 因为图片内容本身是无序的，所以可以直接分别把训练集里dogs和cats中的2450个图片放到验证集里。
def mv_valid_images(files, src_dir, target_dir):
    index = 1
    for f in tqdm(files):
        if index > 2450:
            return
        src = os.path.join(src_dir, f)
        target = os.path.join(target_dir, f)
        shutil.move(src, target)

        index = index + 1

In [12]: # cats
mv_valid_images(cat_files, DIR_TRAIN_NEW_CAT, DIR_VALID_NEW_CAT)

  0%|          | 0/12477 [00:00<?, ?it/s]

In [13]: # dogs
mv_valid_images(dog_files, DIR_TRAIN_NEW_DOG, DIR_VALID_NEW_DOG)

  0%|          | 0/12480 [00:00<?, ?it/s]
```

使用了数据增强处理，意味着我们的训练数据是不断变化的，根据提供给ImageDataGenerator的参数随机调整每批新数据。因此，需要利用Keras的.fit_generator函数来训练我们的模型。

```
# train the model
train_steps=train_batches.samples // BATCH_SIZE
valid_steps=valid_batches.samples // BATCH_SIZE
train_history = net_final.fit_generator(train_batches,
                        steps_per_epoch = train_steps,
                        validation_data = valid_batches,
                        validation_steps = valid_steps,
                        epochs = NUM_EPOCHS)

# save trained weights
net_final.save(WEIGHTS_FINAL)
```

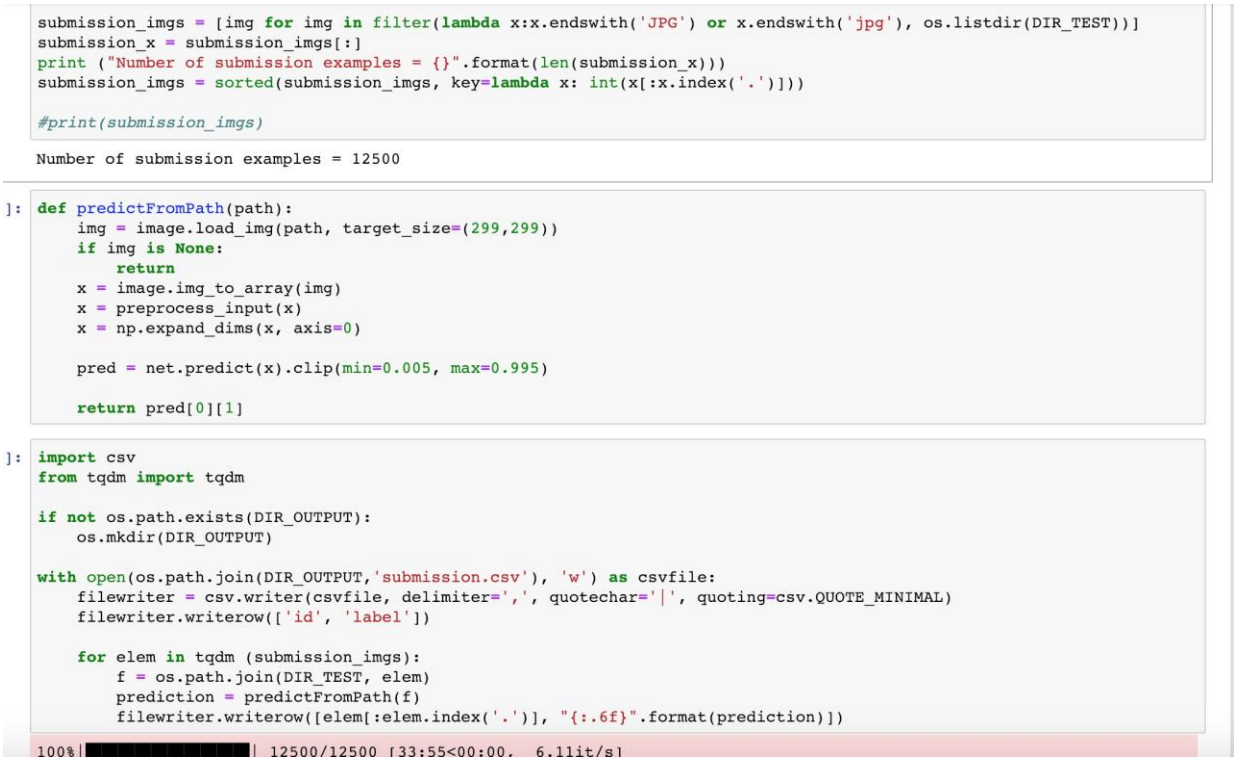
图 13 训练模型

训练结果和学习曲线如图 16 和图 18 所示。（在下面结果章节中）

7. 预测结果

AWS上训练完后保存模型，本项目把训练后的模型下载到本地，然后在本地进行test集预测结果。（为了省点钱）

首先获取test集里的文件，然后针对每个图片进行预测，预测的结果是为狗的概率，并将结果按照kaggle的要求写到csv文件中，并上传至kaggle评分。预测关键代码如下：



```
submission_imgs = [img for img in filter(lambda x:x.endswith('JPG') or x.endswith('jpg'), os.listdir(DIR_TEST))]
submission_x = submission_imgs[:]
print ("Number of submission examples = {}".format(len(submission_x)))
submission_imgs = sorted(submission_imgs, key=lambda x: int(x[:x.index('.')]))

#print(submission_imgs)
```

```
Number of submission examples = 12500
```

```
]: def predictFromPath(path):
    img = image.load_img(path, target_size=(299,299))
    if img is None:
        return
    x = image.img_to_array(img)
    x = preprocess_input(x)
    x = np.expand_dims(x, axis=0)

    pred = net.predict(x).clip(min=0.005, max=0.995)

    return pred[0][1]
```

```
]: import csv
from tqdm import tqdm

if not os.path.exists(DIR_OUTPUT):
    os.mkdir(DIR_OUTPUT)

with open(os.path.join(DIR_OUTPUT,'submission.csv'), 'w') as csvfile:
    filewriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
    filewriter.writerow(['id', 'label'])

    for elem in tqdm (submission_imgs):
        f = os.path.join(DIR_TEST, elem)
        prediction = predictFromPath(f)
        filewriter.writerow([elem[:elem.index('.')], "{:.6f}".format(prediction)])
```

```
100%|████████████| 12500/12500 [33:55<00:00,  6.11it/s]
```

图 14 预测关键代码

完善

1. 开始向 AWS 上上传数据集花费太多时间，后面发现可以使用 kaggle API 直接下载，大大加快速度。
2. 开始把所有的数据探索、数据预处理、训练和预测都放在同一个 notebook 文件里，导致每次文件加载很慢而且一些方法和引入的包不好变通。后面把这几部分都单独分开，实际中这些也应该分开；因为每部分其实都可以尝试不同的 package 方法。
3. 本项目同时也尝试不用增强数据来训练，结果和期望的不太一样，原以为用增强数据的效果肯定会更好。但实际的结果是，不做数据增强的模型效果要更好，达到了惊人的 99.99%。不带数据增强训练的结果和 acc / loss 曲线如图 15 所示。（这个有点意外，保持怀疑态度）

```
Epoch 1/10
313/313 [==============================] - 730s 2s/step - loss: 0.1458 - acc: 0.9411 - val_loss: 0.0374 - val_acc: 0.
9883
Epoch 2/10
313/313 [==============================] - 684s 2s/step - loss: 0.0268 - acc: 0.9905 - val_loss: 0.0304 - val_acc: 0.
9897
Epoch 3/10
313/313 [==============================] - 686s 2s/step - loss: 0.0113 - acc: 0.9968 - val_loss: 0.0288 - val_acc: 0.
9926
Epoch 4/10
313/313 [==============================] - 683s 2s/step - loss: 0.0061 - acc: 0.9979 - val_loss: 0.0284 - val_acc: 0.
9934
Epoch 5/10
313/313 [==============================] - 683s 2s/step - loss: 0.0039 - acc: 0.9989 - val_loss: 0.0313 - val_acc: 0.
9920
Epoch 6/10
313/313 [==============================] - 684s 2s/step - loss: 0.0030 - acc: 0.9991 - val_loss: 0.0276 - val_acc: 0.
9934
Epoch 7/10
313/313 [==============================] - 683s 2s/step - loss: 0.0025 - acc: 0.9992 - val_loss: 0.0284 - val_acc: 0.
9932
Epoch 8/10
313/313 [==============================] - 683s 2s/step - loss: 0.0010 - acc: 0.9998 - val_loss: 0.0291 - val_acc: 0.
9936
Epoch 9/10
313/313 [==============================] - 682s 2s/step - loss: 0.0011 - acc: 0.9997 - val_loss: 0.0336 - val_acc: 0.
9928
Epoch 10/10
313/313 [==============================] - 684s 2s/step - loss: 9.0770e-04 - acc: 0.9999 - val_loss: 0.0313 - val_ac
c: 0.9934
```

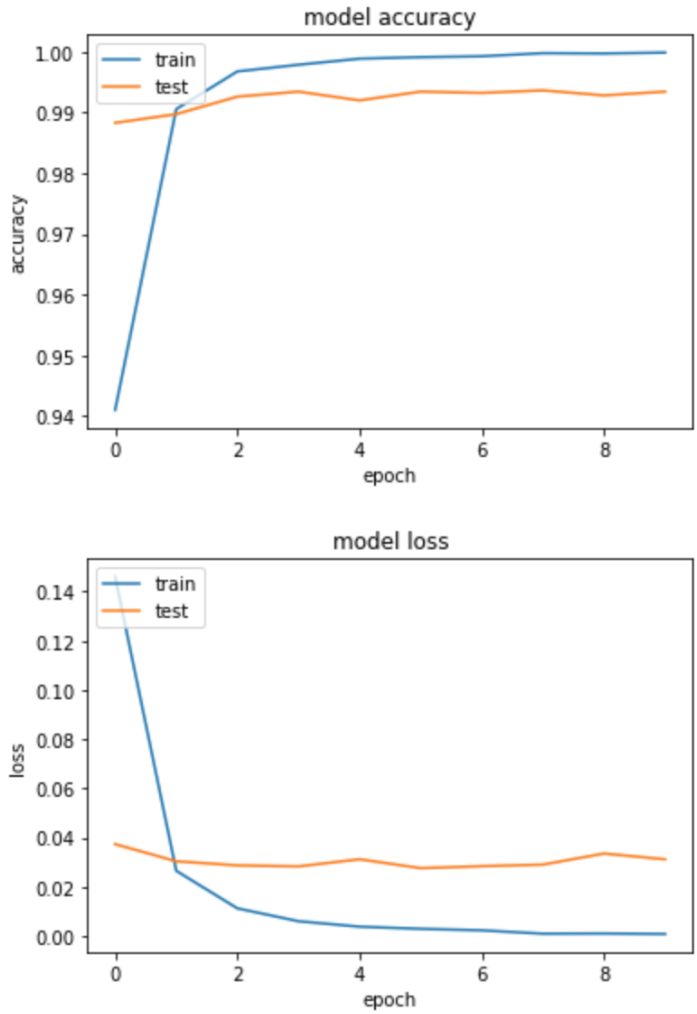


图15 不做数据增强的训练结果

本项目由于训练时间太长，而且开始就得到了不错的结果，故只对是否进行数据增强处理进行的比较，没有对其他参数进行比较。

除了比较是否对数据进行增强处理，也可以尝试改变不同的dropout，优化器，激活函数，损失函数等等。通过不断的尝试和实践，最终会取得比较好的模型。

IV. 结果

模型的评价与验证

1. 训练结果

图16是10次训练结果，从结果可以看出准确率在不断上升，loss在不断下降，训练的结果很好。（虽然没有完全收敛，但结果已经比较好了。加入到后续改进的一点，增加epoch，应该可以得到更好的结果）。

```
Epoch 1/10
313/313 [==============================] - 717s 2s/step - loss: 0.1861 - acc: 0.9233 - val_loss: 0.0220 - val_acc: 0.
9924
Epoch 2/10
313/313 [==============================] - 685s 2s/step - loss: 0.0700 - acc: 0.9747 - val_loss: 0.0191 - val_acc: 0.
9942
Epoch 3/10
313/313 [==============================] - 683s 2s/step - loss: 0.0501 - acc: 0.9824 - val_loss: 0.0185 - val_acc: 0.
9947
Epoch 4/10
313/313 [==============================] - 680s 2s/step - loss: 0.0396 - acc: 0.9866 - val_loss: 0.0154 - val_acc: 0.
9951
Epoch 5/10
313/313 [==============================] - 679s 2s/step - loss: 0.0367 - acc: 0.9881 - val_loss: 0.0161 - val_acc: 0.
9953
Epoch 6/10
313/313 [==============================] - 685s 2s/step - loss: 0.0306 - acc: 0.9904 - val_loss: 0.0152 - val_acc: 0.
9955
Epoch 7/10
313/313 [==============================] - 679s 2s/step - loss: 0.0249 - acc: 0.9915 - val_loss: 0.0189 - val_acc: 0.
9953
Epoch 8/10
313/313 [==============================] - 683s 2s/step - loss: 0.0213 - acc: 0.9933 - val_loss: 0.0134 - val_acc: 0.
9957
Epoch 9/10
313/313 [==============================] - 679s 2s/step - loss: 0.0228 - acc: 0.9925 - val_loss: 0.0174 - val_acc: 0.
9957
Epoch 10/10
313/313 [==============================] - 696s 2s/step - loss: 0.0156 - acc: 0.9945 - val_loss: 0.0181 - val_acc: 0.
9959
```

图 16 训练结果

2. **学习曲线**

学习曲线如图17所示，可以看出没有过拟合，结果很不错。不过曲线还没有完全收敛，应该还有提升空间，这个已经作为需要改进的一个点。

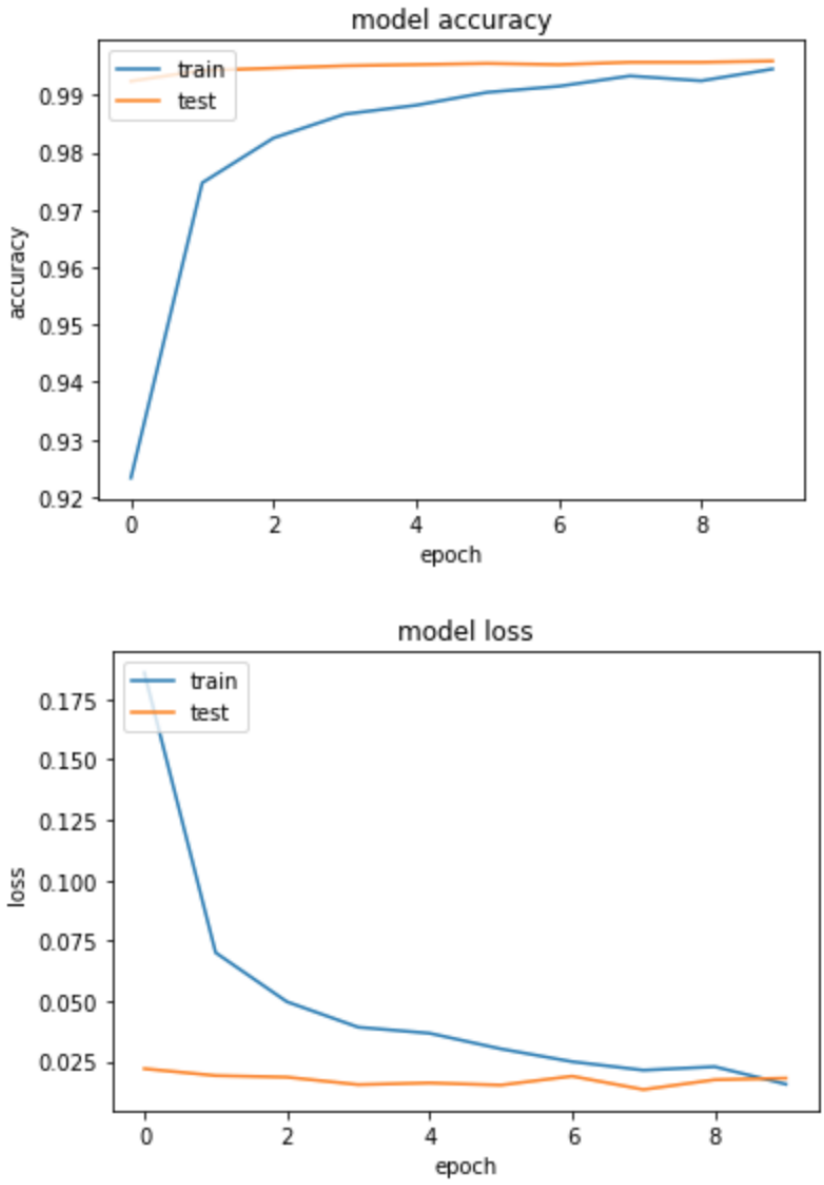


图 17 学习曲线

3. **预测结果可视化**

使用训练后的模型验证可视化结果，如图18所示。可以看出识别出的准确度非常之高，部分图片肉眼都很难识别清楚。


Cat: 0.995000
Dog: 0.995000
Cat: 0.995000
Dog: 0.995000
Dog: 0.995000
Cat: 0.995000
Dog: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Cat: 0.995000
Dog: 0.995000
Lois

图18 预测结果

4. 预测结果 Kaggle 验证

Kaggle的验证结果如图19所示，得分还是比较高的，超出项目的要求前10%即要小于0.06127。

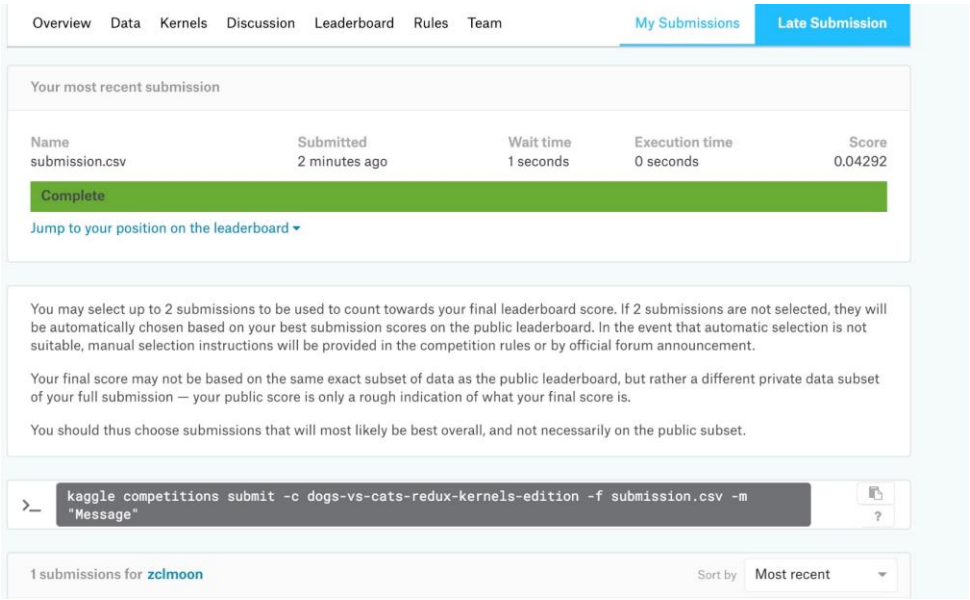


图 19 Kaggle 验证结果

合理性分析

从上面的图 17、图 18 和图 19 训练结果和训练曲线可以看出得到很好的 Accuracy 和 Loss 曲线，本地测试结果准确度很高，而且 Kaggle 验证也达到前 10%。

对项目的思考

通过本项目的实践，了解到迁移学习在实际应用中非常重要。不需要花费很大的成本就可以获得非常好的适合自己领域的模型。而实际应用中，大多公司都不具备研发设计新模型的能力，但迁移学习可以让所有的公司能够很容易的应用。

迁移学习有一个很重要的问题是自己领域的数据集和预训练模型使用数据集的差异，这个差异决定了迁移学习的难度和准确率。

另一个总结是，AWS GPU instance 真是好东西；曾尝试在自己的 Mac Bookpro 训练，预计需要两天的时间，这基本不太现实。使用了 AWS p2.xlarge 两个小时就训练完了，加上后期的修改，尝试总共也就花了十几美元。

需要作出的改进

1. 可以考虑把多个预训练模型融合一起试试，可能会得到更好的结果
2. 可以尝试微调 dropout 试试结果
3. 可以继续增加 epoch 比如增大到 15~20 看看结果；从目前的 10 epoch 结果来看，还是没有完全收敛，还有提升的空间。
4. 一个比较费解的问题：有的同学每个 epoch 能在几秒钟就训练完，而我的每个 epoch 则要十几分钟，一次训练都得两三个小时，费了不少美刀。（我用的 aws p2xlarge）。

V. 参考文献

[1] Inception Net-V3 结构图 https://www.jianshu.com/p/3bbf0675cfce

[2] Keras Document: https://keras.io/applications/#available-models

[3] Transfer learning & The art of using Pre-trained Models in Deep Learning: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In arXiv,2015.

[5] Keras Cats Dogs Tutorial: https://jkjung-avt.github.io/keras-tutorial/

[6] Building powerful image classification models using very little data: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[7] 吴恩达 deeplearning.ai】深度学习(7)：卷积神经网络：https://blog.csdn.net/sinat_35473930/article/details/78771589
[8] Dogs vs Cats Competition： https://www.kaggle.com/josecyn/dogs-vs-cats-inceptionv3-w-keras
[9]迁移学习: https://blog.csdn.net/cumttzh/article/details/79792010
[10] 基于深度学习和迁移学习的识花实践：https://cosx.org/2017/10/transfer-learning/
[11] Inception-v3 的设计思路小结 https://www.cnblogs.com/eniac1946/p/8669937.html
[12]深入浅出——网络模型中 Inception 的作用与结构全解析 https://blog.csdn.net/u010402786/article/details/52433324/
[13] 深度神经网络 Google Inception Net-V3 结构图 https://www.jianshu.com/p/3bbf0675cfce
[14] 简单认识 Adam 优化器 https://www.jianshu.com/p/aebcaf8af76e