

猫狗大战

机器学习工程师纳米学位毕业项目

桂骏马

2018 年 6 月 24 日

1. 定义

1.1 项目概览

这是 2013 年 Kaggle 一个娱乐的竞赛项目，猫狗图片分类。提供的数据集包含训练集 25000 张猫和狗图片，测试集 15000 张猫和狗的图片。用训练集图片数据，训练一个机器学习的模型，预测测试集中图片是狗还是猫。

这是一个标准的图像分类问题，本项目采用基于卷积神经网络 CNN 的深度学习（Deep Learning）算法来构建模型。

神经网络是 2012 年重新兴起的热门算法，Geffory Hinton 的学生 Alex Krizhevsky 提出 AlexNet，在 2012 年 ILSVRC 图像分类竞赛中一举夺得冠军，top-5 错误率比上一年的冠军（SVM 算法，手动提取特征）25.7%下降了 10%左右 达到 15.3%，而且远远领先当年的第二名（26.2%）。从此之后基于神经网络的深度网络模型占据了每年的 ILSVRC 图像分类竞赛 top 排行榜。

1.2 问题陈述

Kaggle 提供的图片集，样式风格多样，图片大小不同，猫和狗的品种也很多。很好地体现真实物理世界的复杂性，对模型特征选取的挑战很高。神经网络自动学习特征的优点，非常适合做图片分类任务。

输入图片像素 RGB 数据，构建深层神经网络，第一层学习基本的边沿形状，后续层学习更复杂的结构特征如对象的轮廓，纹理，颜色等。根据学习到的特征对应的分类标签，经过大量的图片数据训练出模型，应用最终的模型对测试集数据预测每张图片是狗的概率（狗的概率以 1 表示，猫的概率以 0 表示）

1.3 评价标准

Kaggle 猫狗大战的评分标准采用对数损失 LogLoss，越小越好。

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

- n is the number of images in the test set
- \hat{y}_i is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- $\log()$ is the natural (base e) logarithm

log loss 作为对预测概率的评分标准是非常合适的

1. 真实概率即测试集图片是狗的概率分布是服从伯努利分布（0-1 分布）
2. 对数损失是度量了真实概率分布与预测概率分布之间的差异，这里的差异度量用的是 KL 散度，等效于求最大似然估计。所以对数损失值越小越好！

2. 分析

2.1 样本数据分析

训练集 25000 张图片，分别是 12500 张猫的图片 and 12500 张狗的图片。平衡样本，猫和狗各一半。有利于训练模型正确识别猫和狗的特征，不会偏向于一类。图片以类别+数字作为文件名，图片格式是 jpg。

猫的图片，前 10 张图片如下。

cat.0.jpg: dimension (374, 500, 3)

cat.1.jpg: dimension (280, 300, 3)

cat.2.jpg: dimension (396, 312, 3)

cat.3.jpg: dimension (414, 500, 3)

cat.4.jpg: dimension (375, 499, 3)

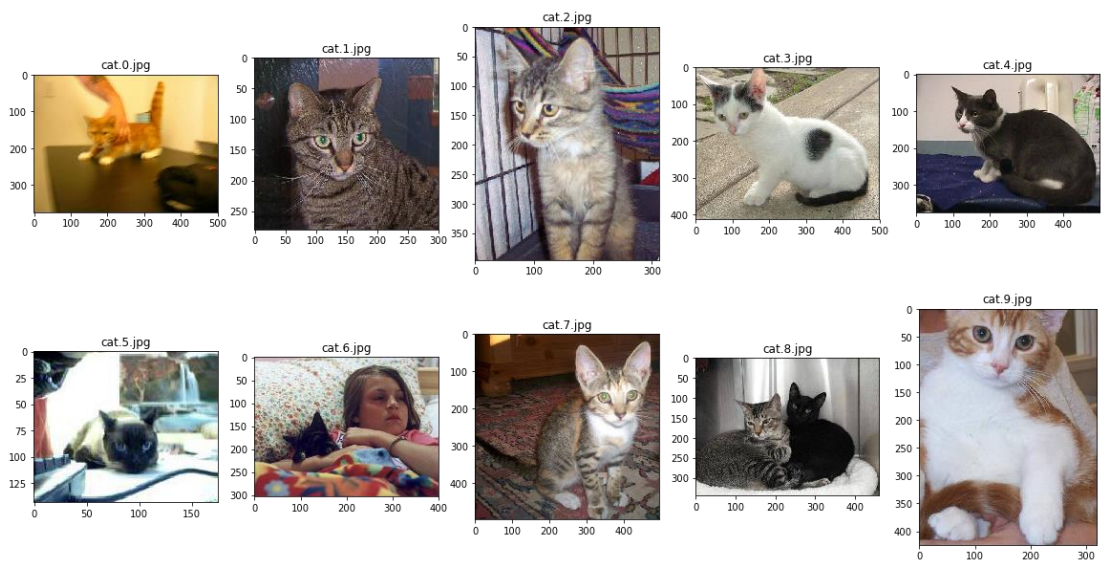
cat.5.jpg: dimension (144, 175, 3)

cat.6.jpg: dimension (303, 400, 3)

cat.7.jpg: dimension (499, 495, 3)

cat.8.jpg: dimension (345, 461, 3)

cat.9.jpg: dimension (425, 320, 3)



狗的图片，前 10 张图片如下。

dog.0.jpg: dimension (375, 499, 3)

dog.1.jpg: dimension (499, 327, 3)

dog.2.jpg: dimension (199, 187, 3)

dog.3.jpg: dimension (375, 499, 3)

dog.4.jpg: dimension (287, 300, 3)

dog.5.jpg: dimension (376, 499, 3)

dog.6.jpg: dimension (488, 499, 3)

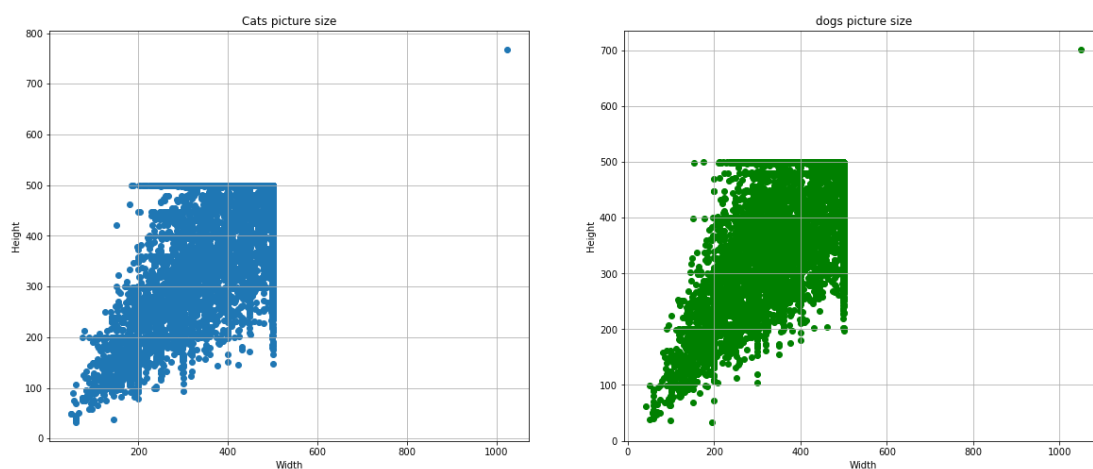
dog.7.jpg: dimension (264, 299, 3)

dog.8.jpg: dimension (500, 469, 3)

dog.9.jpg: dimension (500, 368, 3)



在训练过程中，要对训练集图片进行清洗和各种变换。需要对训练集图片进一步了解其分布特征。我们对所有训练集图片尺寸分布画出散点图如下，横坐标是宽度(width)，纵坐标是高度(height)。



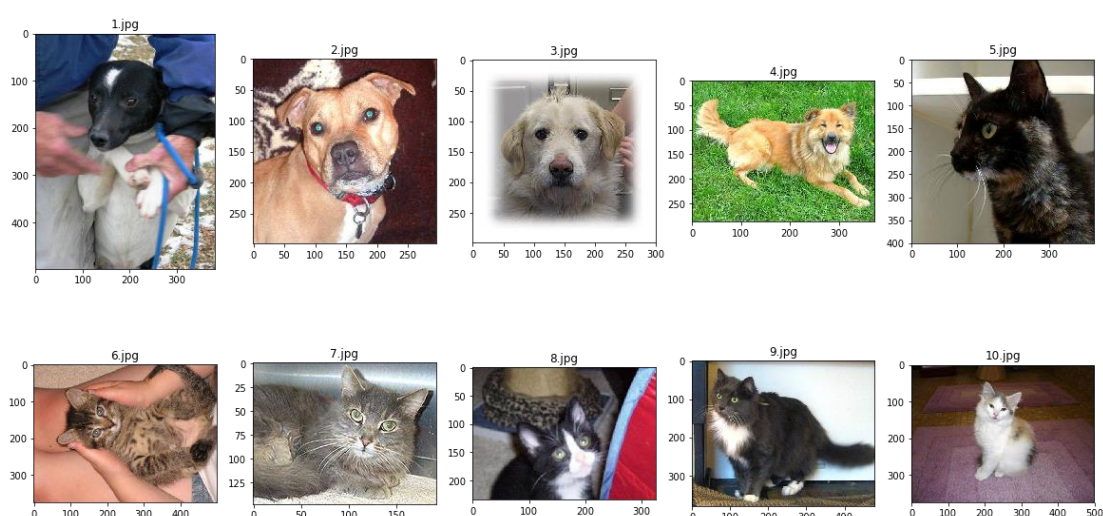
猫的图片尺寸大部分宽度范围在(200-500)，高度范围(200-500)。

狗的图片尺寸也同样大部分宽度范围在(200-500)，高度范围(200-500)。

测试集 15000 张图片，以数字作为文件名，没有标签，图片格式是 jpg。

前 10 张图片如下

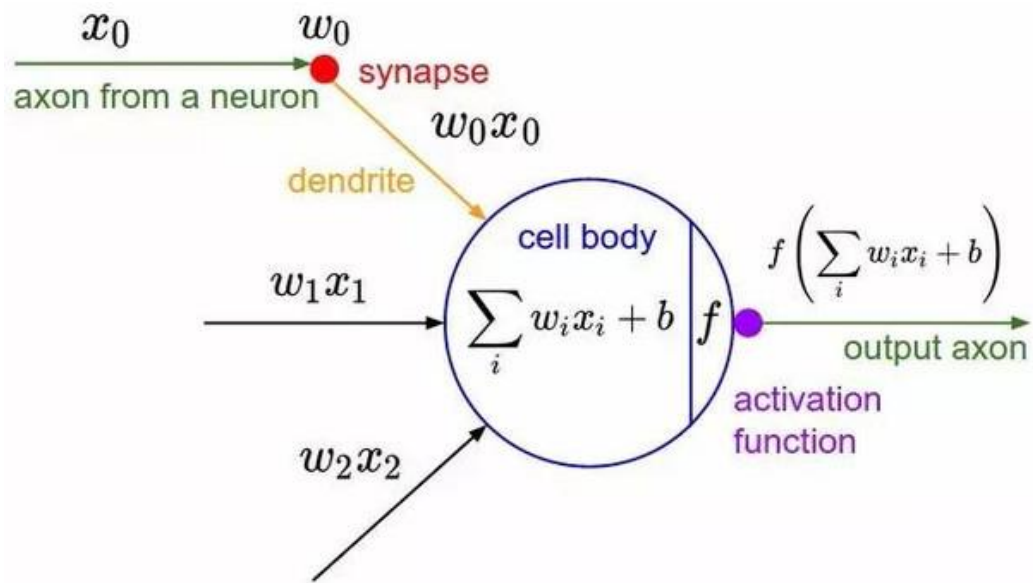
- 1.jpg: dimension (499, 381, 3)
- 2.jpg: dimension (299, 296, 3)
- 3.jpg: dimension (299, 300, 3)
- 4.jpg: dimension (288, 374, 3)
- 5.jpg: dimension (400, 399, 3)
- 6.jpg: dimension (375, 499, 3)
- 7.jpg: dimension (148, 192, 3)
- 8.jpg: dimension (234, 325, 3)
- 9.jpg: dimension (380, 480, 3)
- 10.jpg: dimension (374, 500, 3)



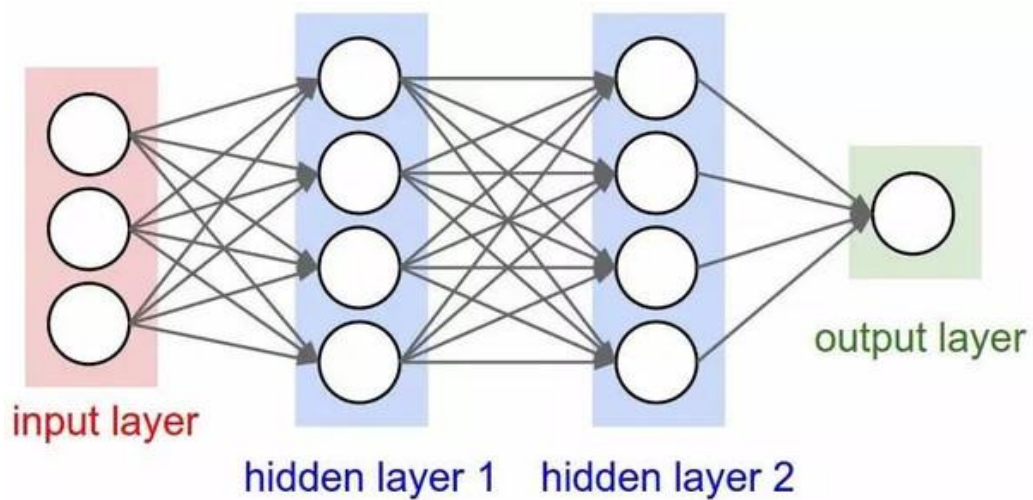
2.2 算法分析

卷积神经网络原理的介绍

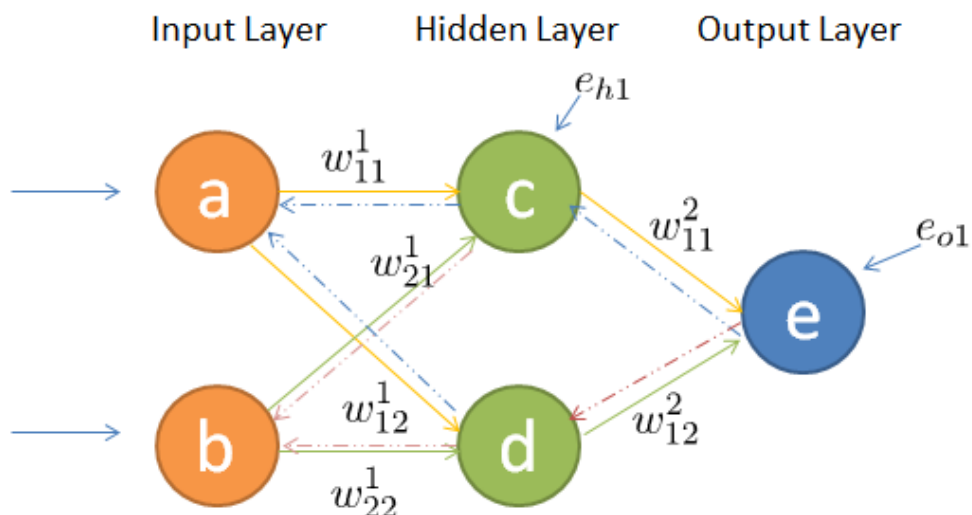
1943 年，McCulloch 和 Pitts 提出 **M-P** 神经元模型。1956 年 Frank Rosenblatt 提出了感知机 (Perceptron) 模型加入了权值计算。神经元接收来自 n 个其他神经元传递过来的输入信号 x ，这些输入信号通过带权重 w 的连接 (connection) 进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过“激活函数” (activation function) f 处理产生神经元的输出。感知机成为神经网络的基础单元。



后续发展出多层神经网络，拟合更为复杂的多维非线性任意函数。



1986 年 David Rumelhart 等学者出版的《平行分布处理:认知的微观结构探索》一书。书中完整地提出了 BP 算法,系统地解决了多层网络中隐单元连接权的学习问题,并在数学上给出了完整的推导。这是神经网络发展史上的里程碑, BP 算法迅速走红,掀起了神经网络的第二次高潮



反向传播是利用链式法则递归计算表达式的梯度的方法，基于初始化权值前向计算出损失函数，得出预测值和真实值误差，并采用 Mini-batch 随机梯度下降的方法更新权值。

1988 年 Yann Lecun 在贝尔实验室 提出卷积神经网络（Convolutional Neural Networks）对手写字母达到很高的识别率。2012 年 Alex Krizhevsky 提出 AlexNet（5 个卷积层+2 个全连接层+1 个 softmax 层）夺得 ImageNet 冠军，迎来基于神经网络的深度学习的全面兴起，并应用到各个领域。

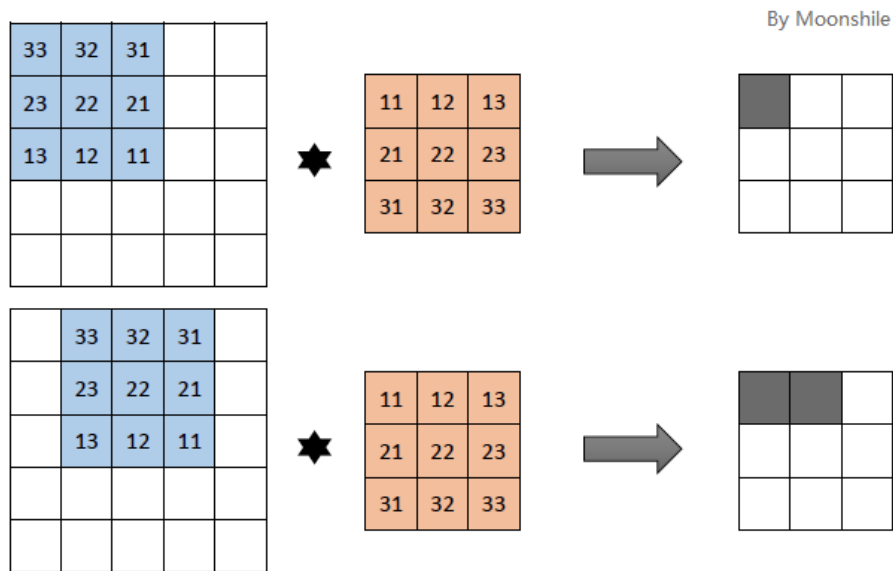
卷积神经网络 (CNN) 通常包含以下几种层

- 卷积层 (Convolutional layer)，卷积神经网络中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法优化得到的。卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等，更多层的网络能从低级特征中迭代提取更复杂的特征。
- 激活层 (Activation layer)，这一层神经的活性化函数 (Activation function) 现在通常使用 ReLU，某些情况下使用 sigmoid 等
- 池化层 (Pooling layer)，通常在卷积层之后会得到维度很大的特征，将特征切成几个区域，取其最大值或平均值，得到新的、维度较小的特征。
- 全连接层 (Fully-Connected layer)，把所有局部特征结合变成全局特征，用来计算最后每一类的 Loss。

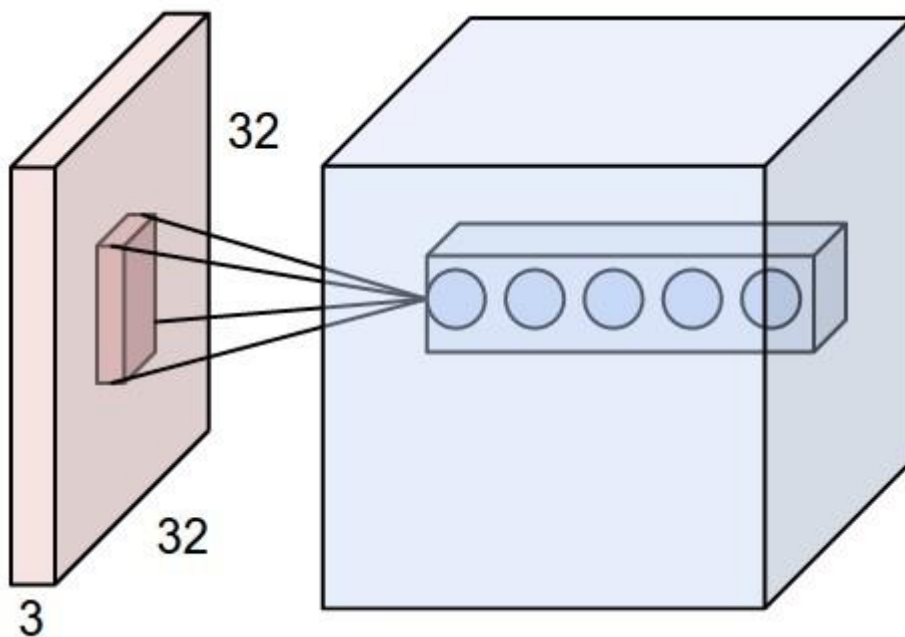
卷积神经网络通过两种方法可以大幅降低全连接神经网络的参数数目

1. 局部感受野

假定一个大小为 5×5 的图像，和一个 3×3 的卷积核。卷积核实际上有 9 个神经元，他们的输出又组成一个 3×3 的矩阵，称为特征图。第一个神经元连接到图像的第三个 3×3 的局部，第二个神经元则连接到第二个局部。连续扫描完整幅图像，完成一次图像的卷积运算。



卷积层的参数包含一系列过滤器 (filter)，每个过滤器训练一个深度，有几个过滤器输出单元就具有多少深度，对应不同的特征图 (feature map)

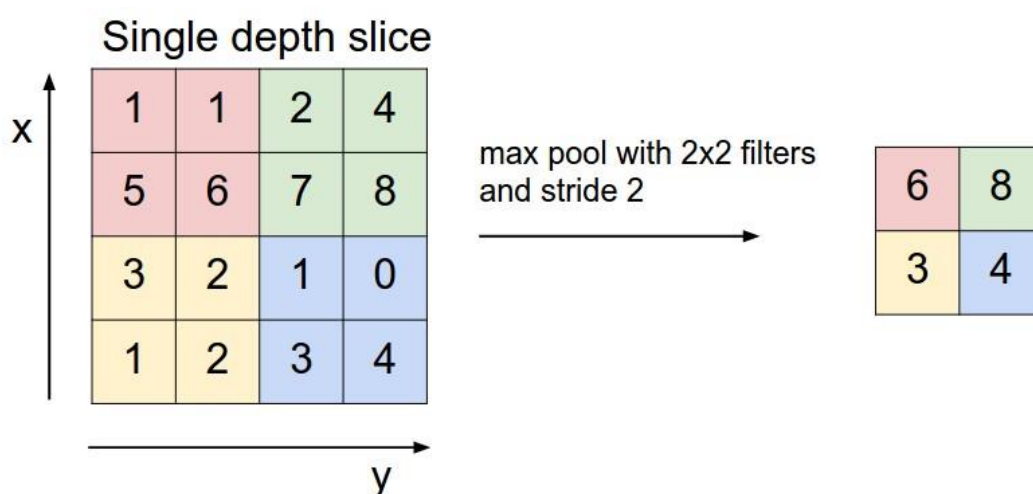


2. 参数共享(Parameter Sharing)

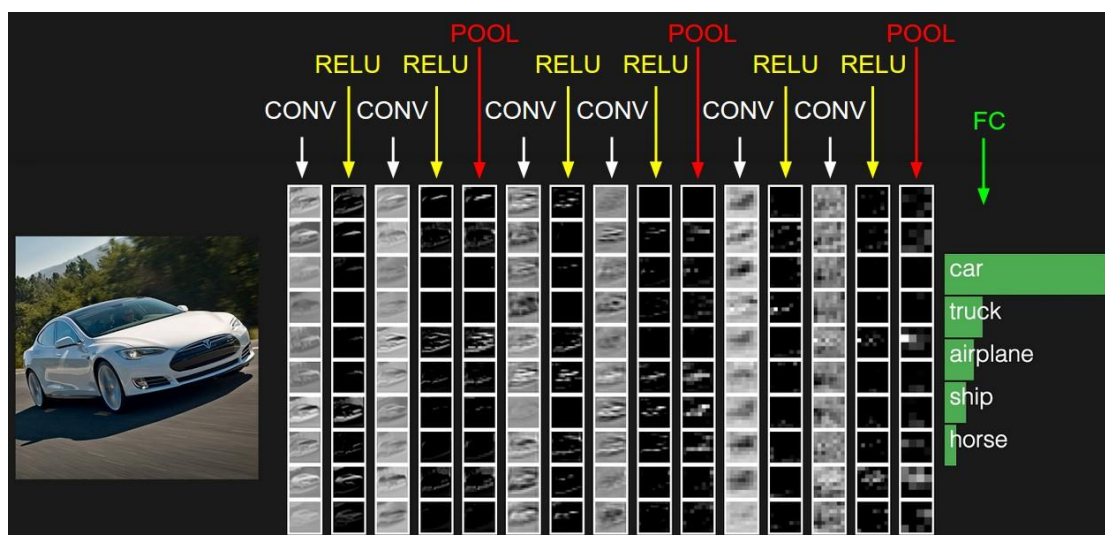
- 基于一个假设：图像不同区域的共同特征的过滤器（filter）是相同的，不会随着位置而改变特征的代表。
- 权值共享使得我们能更有效的进行特征抽取，因为它极大的减少了需要学习的自由变量的个数。极大的减少了计算量，便于实现大规模深层神经网络。

池化层(Pooling Layer)

- 池化（pool）即下采样（downsamples），目的是为了减少特征图的 size。池化操作对每个深度切片独立，规模一般为 2×2 ，相对于卷积层进行卷积运算。
- 池化操作将保持特征图(feature map)深度大小不变，即不改变输出特征的数目。



一个卷积神经网络各层应用实例：



第一层卷积层提取一些低级的特征如边缘、线条和角等，后面的卷积层从低级特征中迭代提取更复杂的特征组合，最后的全连接层+Softmax 输出分类。

2.3 卷积神经网络算法实现

对于这个项目采用 Google 的 tensorflow [4] + Keras [5] 框架来构建模型。Keras 是一个崇尚极简、高度模块化的神经网络库，并可以同时运行在 TensorFlow 和 Theano 上。可以通过编程的方式调试模型结构和各种超参数，简化了编程的复杂度。图像处理需要进行大量的 tensor 矩阵运算操作，并且还要调用 GPU 进行并行计算。

算法主要步骤：

- 图像数据预处理，切分训练集，验证集，转换成 input_tensor。
- 模型计算图构建，考虑模型的层，损失函数，优化器。
- 训练模型，采用 mini-batch 梯度下降算法更新模型权值。
- 根据 val-loss, accuray 等指标，调整模型参数，优化模型。
- 对测试集图片预测概率，输出结果。

2.4 基准测试

本项目是 Kagge 竞赛题，采用 Kaggle 的 log loss 作为评分标准。由于硬件计算资源的限制，CPU Intel I7-7700, RAM 16G, GPU Nvidia GTX-1060-6G. 目标 log loss < 0.06

3. 方法和具体实现

3.1 模型选择

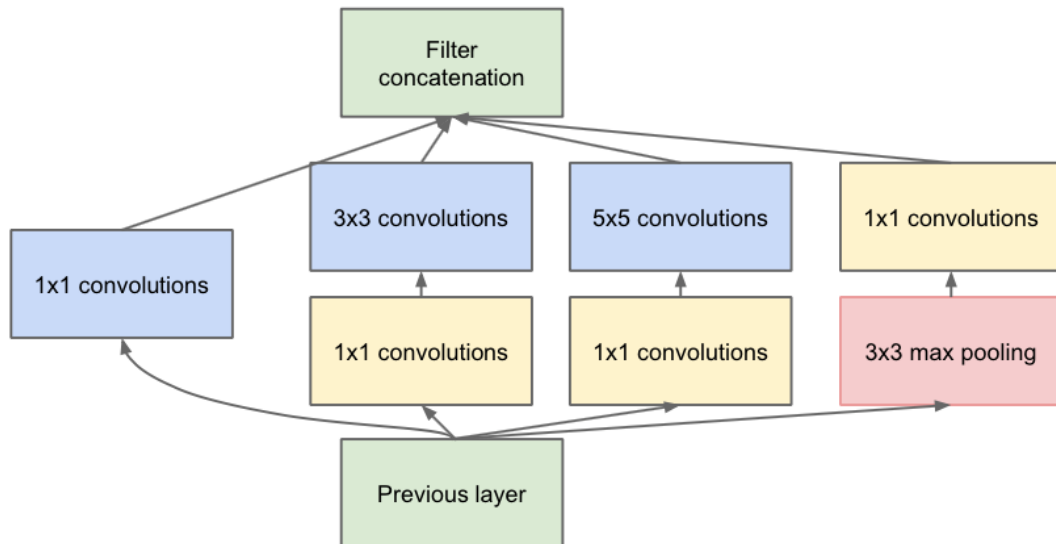
模型	大小	Top1 准确率	Top5 准确率	参数数目	深度
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.715	0.901	138,357,544	23
VGG19	549MB	0.727	0.910	143,667,240	26
ResNet50	99MB	0.759	0.929	25,636,712	168
InceptionV3	92MB	0.788	0.944	23,851,784	159
IncetionResNetV2	215MB	0.804	0.953	55,873,736	572
MobileNet	17MB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

VGG16 和 VGG19 是比较早期的模型，参数数目多，计算量大。所以考虑 InceptionV3 [1]， ResNet50 [2]， Xception [3] 这 3 个模型 进行迁移学习。

先探讨一下 2 个神经网络模型 InceptionV3 [1], ResNet50 [2]。分别代表着神经网络不同的发展方向或思路

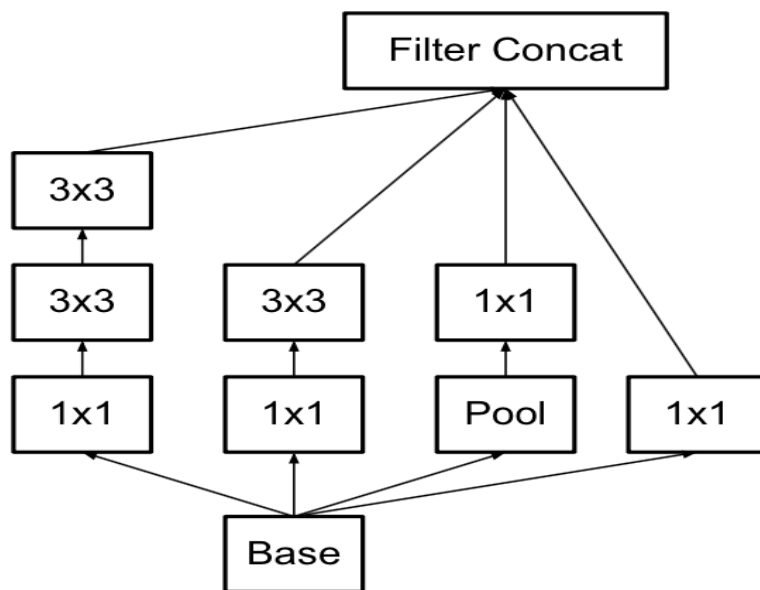
InceptionV3[1] 也称为 GoogleNet, 从 V1 一直发展到 V4。InceptionV1 主要思想体现在 2 个方面:

1. 采用不同的卷积核, 对前面层的特征图进行抽取。相当于增加网络的宽度。
2. 采用了 1×1 卷积核 进行降维, 减少计算量。和 VGG-16/VGG-19 相比层数增加很多, 而参数却少很多。



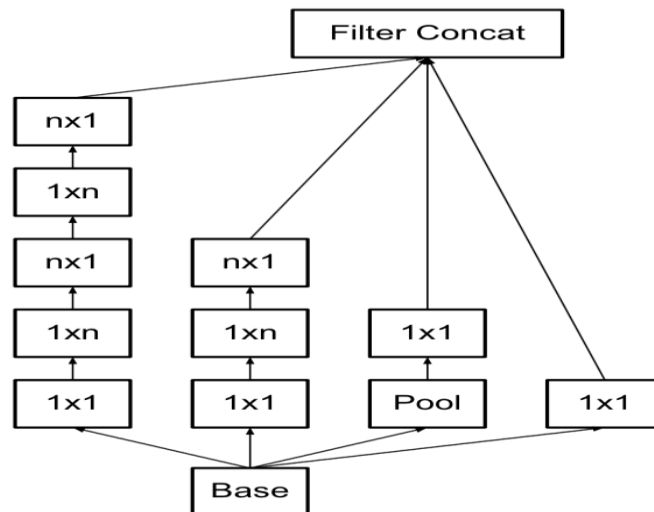
InceptionV2 的改进

1. 学习 VGG 用 2 个 3×3 的 conv 替代 inception 模块中的 5×5 , 既降低了参数数量, 也加速计算。
2. 加入了 BN 层, 减少了数据分布变化的影响。使每一层的输出都规范化到一个 $N(0, 1)$ 的高斯分布, 再通过两个可学习的变量恢复要学习的特征。



Inception v3 的改进

1. 将 7×7 分解成两个一维的卷积 ($1 \times 7, 7 \times 1$)， 3×3 也是一样 ($1 \times 3, 3 \times 1$)，这样的好处，既可以加速计算（多余的计算能力可以用来加深网络），又可以将 1 个 conv 拆成 2 个 conv，使得网络深度进一步增加，增加了网络的非线性。
2. 还有值得注意的地方是网络输入从 224×224 变为了 299×299 。



另外注意到 GoogleNet 为了解决深层神经网络的梯度消失，训练过程中的过拟合问题，并在训练时加速收敛，在网络的中间层加入了 2 个辅助的 Loss 和最终的 Loss 加权，作为整个网络的损失函数来训练网络。

ResNet50 [2] 也称为残差网络，2015 MSRA 何凯明团队提出的 Residual Networks，赢得 Imagenet 分类任务的冠军，MSRA 还用 residual networks 在 ImageNet 的 detection, localization，以及 COCO 数据集上的 detection 和 segmentation 等任务也极其瞩目。

神经网络模型发展从 Lenet \rightarrow AlexNet \rightarrow VGG16 \rightarrow VGG19，趋势是层数越来越深。但是网络达到一定深度后，训练结果的准确度(accuracy)却下降，这就是网络退化 (degradation)

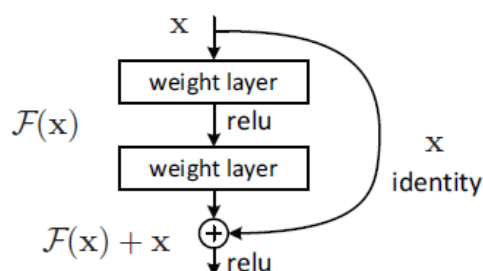
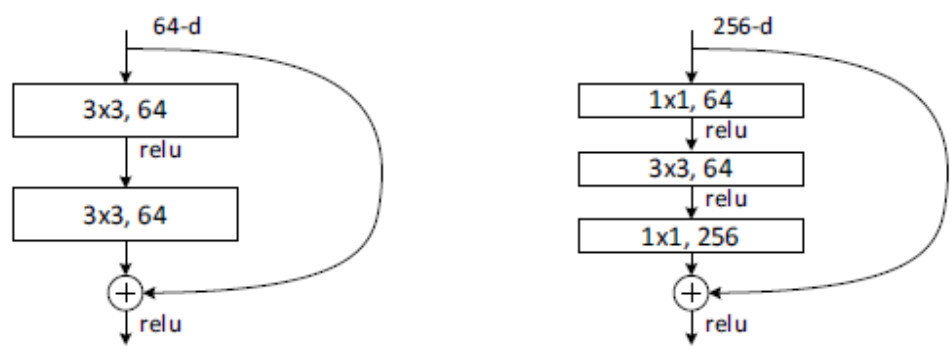


Figure 2. Residual learning: a building block.

ResNet 在层间额外增加同等映射 identity mapping，相当于增加前向/后向传播的 shortcut 路径，可以减少梯度在层间传播的损失。



左图是最初提出的基本 ResNet 模块，层层叠加构成 34 层网络。
右图是改进的 ResNet 模块，先通过 1x1 卷积降维，降低计算量，然后 3x3 卷积层，最后再通过 1x1 卷积和 shortcut 维度相同并相加；由改进后的模块，叠加构成 ResNet50, ResNet101, ResNet152 最深 152 层的网络。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

最后了解 Xception [3] 模型，是 Google 的 Francois Chollet 提出的，基于 Inception 改进的网络。Inception 假设 卷积层试图在 3D 空间学习过滤器，2 个空间维度（宽和高）以及 1 个通道维度，因此一个卷积核需要同时绘制跨通道相关性和空间相关性。Inception 模块背后的思想就是通过将这个过程分解成一系列相互独立的操作以使它更为便捷有效。进一步讲，典型的 Inception 模块首先处理跨通道相关性，通过一组 1×1 卷积，将输入数据绘制到 3 或 4 个小于原始输入的不同空间，然后通过 3×3 或者 5×5 卷积将所有相关性绘制到更小的 3D 空间。

Figure 1. A canonical Inception module (Inception V3).

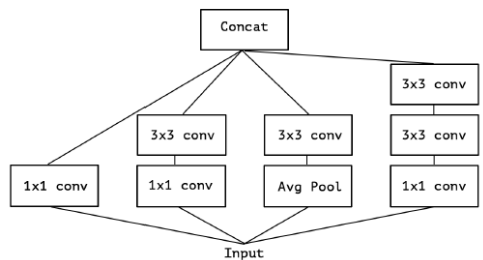


Figure 2. A simplified Inception module.

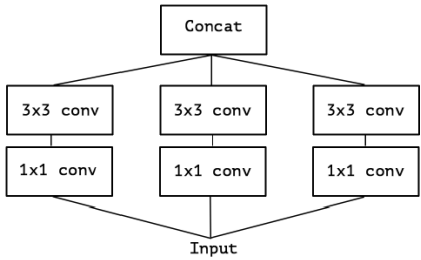


Figure 3. A strictly equivalent reformulation of the simplified Inception module.

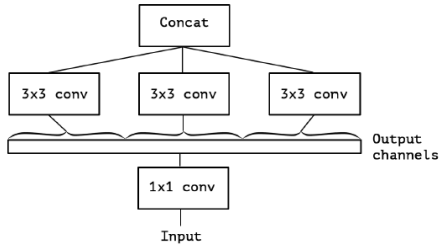
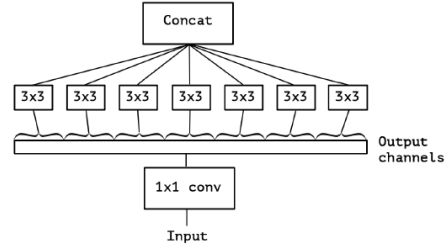
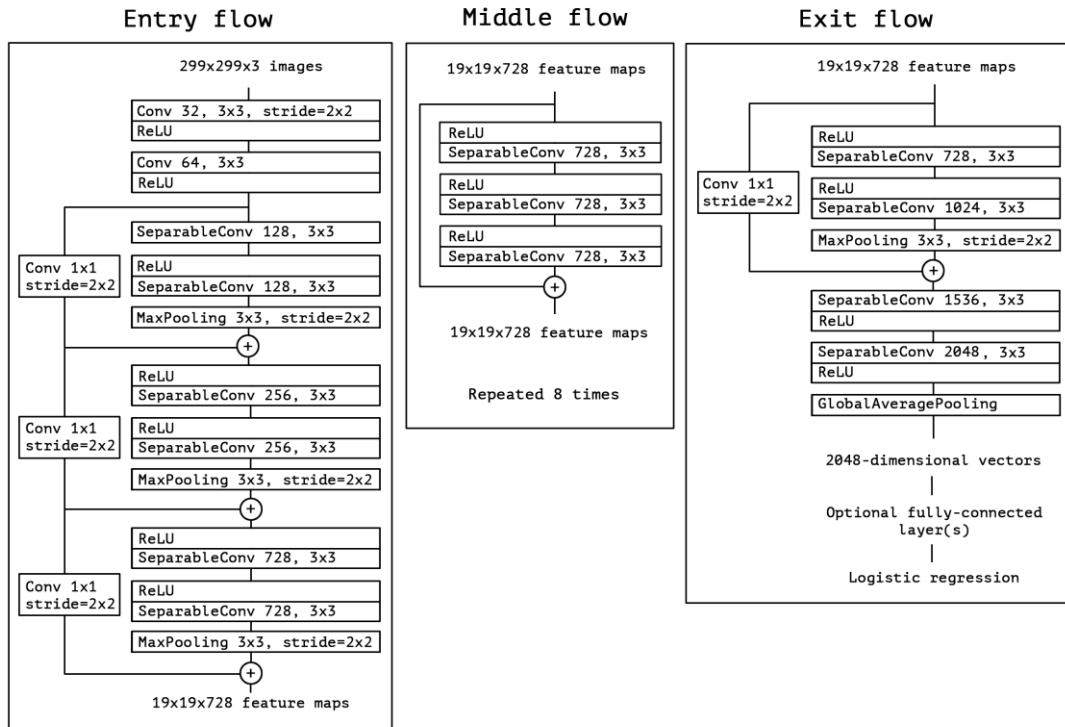


Figure 4. An “extreme” version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



Xception [3]基本的假设是使跨通道相关性和空间相关性的绘制有效脱钩。由于这种假设是 Inception 结构中极端化的假设，我们将它称作 Xception，意指极端 Inception。



Xception [3]结构由 36 个卷积层组成网络的特征提取基础，36 个卷积层被分成 14 个模块，除最后一个外，模块间有线性残差连接。一句话概括，Xception [3]结构是带有残差连接的深度可分卷积层的线性堆叠。

3.2 数据预处理

1. 先下载数据文件 train.zip, test.zip, 解压缩并存放本地文件夹。

```

# extract train image to folder 'train' and 'test'
train_path = 'train'
test_path = 'test'
# extract train
if not isdir(train_path):
    zipFile = zipfile.ZipFile('train.zip')
    zipFile.extractall()
    zipFile.close()

# move cat's image to folder 'train/cat/', dog's image to folder 'train/dog'
if not isdir('train/cat'):
    os.mkdir('train/cat')
if not isdir('train/dog'):
    os.mkdir('train/dog')

for filename in os.listdir('train'):
    match_obj1 = re.search(r'cat.[0-9]*.jpg', filename)
    if match_obj1:
        shutil.move('train/'+filename, 'train/cat/')

    match_obj2 = re.search(r'dog.[0-9]*.jpg', filename)
    if match_obj2:
        shutil.move('train/'+filename, 'train/dog/'+filename)

# extract test image to folder 'test'
if not isdir('test'):
    zipFile = zipfile.ZipFile('test.zip')
    zipFile.extractall()
    zipFile.close()
    os.mkdir('test/test')
    for file in os.listdir('test'):
        if not os.path.isdir(file):
            shutil.move('test/'+file, 'test/test/')

```

训练集 25000 张图片，一半是猫，一半是狗，平衡样本。

2. 仔细查看 train 文件夹下猫和狗的图片，发现其中有些图片是异常的。既不是猫也不是狗，或者背景占的比例很大干扰识别。训练集的异常值是需要清洗的，否则会导致模型学习到错误的特征。

常见的数值类型数据集比如房价或销售额等，可以通过四分位差的方法找到明显偏离正常范围的异常值。而图片数据的异常，没办法通过 RGB 亮度值来区分。

人工来挑出异常值的方法，也是不可取的。因为 2,5000 张图片，靠人眼一张张图片去看，花费时间太长，也不是可靠，科学的方法。且如果训练集的图片数量更多，不可能让人去挑出异常图片。

可以想到的方法是采用 imagenet 预训练模型，来对图片进行预测，判断该图片不是猫和狗，而是异常值。imagenet 输出 1000 个分类，猫的分类有 7 种，狗的分类多达 118 种。Imagenet 的评价标准是 Top1, Top5 accuracy, Top1 是预测概率最高的类别，Top5 是预测概率前 5 的类别。

Imagenet 狗和猫的具体类别如下

```
dogs = [  
    'n02085620', 'n02085782', 'n02085936', 'n02086079'  
, 'n02086240', 'n02086646', 'n02086910', 'n02087046'  
, 'n02087394', 'n02088094', 'n02088238', 'n02088364'  
, 'n02088466', 'n02088632', 'n02089078', 'n02089867'  
, 'n02089973', 'n02090379', 'n02090622', 'n02090721'  
, 'n02091032', 'n02091134', 'n02091244', 'n02091467'  
, 'n02091635', 'n02091831', 'n02092002', 'n02092339'  
, 'n02093256', 'n02093428', 'n02093647', 'n02093754'  
, 'n02093859', 'n02093991', 'n02094114', 'n02094258'  
, 'n02094433', 'n02095314', 'n02095570', 'n02095889'  
, 'n02096051', 'n02096177', 'n02096294', 'n02096437'  
, 'n02096585', 'n02097047', 'n02097130', 'n02097209'  
, 'n02097298', 'n02097474', 'n02097658', 'n02098105'  
, 'n02098286', 'n02098413', 'n02099267', 'n02099429'  
, 'n02099601', 'n02099712', 'n02099849', 'n02100236'  
, 'n02100583', 'n02100735', 'n02100877', 'n02101006'  
, 'n02101388', 'n02101556', 'n02102040', 'n02102177'  
, 'n02102318', 'n02102480', 'n02102973', 'n02104029'  
, 'n02104365', 'n02105056', 'n02105162', 'n02105251'  
, 'n02105412', 'n02105505', 'n02105641', 'n02105855'  
, 'n02106030', 'n02106166', 'n02106382', 'n02106550'  
, 'n02106662', 'n02107142', 'n02107312', 'n02107574'  
, 'n02107683', 'n02107908', 'n02108000', 'n02108089'  
, 'n02108422', 'n02108551', 'n02108915', 'n02109047'  
, 'n02109525', 'n02109961', 'n02110063', 'n02110185'  
, 'n02110341', 'n02110627', 'n02110806', 'n02110958'  
, 'n02111129', 'n02111277', 'n02111500', 'n02111889'  
, 'n02112018', 'n02112137', 'n02112350', 'n02112706'  
, 'n02113023', 'n02113186', 'n02113624', 'n02113712'  
, 'n02113799', 'n02113978' ]  
  
cats=[  
    'n02123045', 'n02123159', 'n02123394', 'n02123597'  
, 'n02124075', 'n02125311', 'n02127052' ]
```

利用ResNet50网络进行ImageNet分类

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker', 0.1122357), (u'n02504458', u'African_elephant', 0.06118448)]
```

参考 Keras[5]文档的例子，`decode_predictions(preds, top=3)[0]` 得到图片预测类别。显然 预测准确率 `accuracy Top 30 > Top 20 > Top 10 > Top 5 > Top 1`。因为预测的类别越多，其中包含真实类别的概率肯定越高，预测 accuracy 越高。如果进一步扩大预测类别范围，比如说 Top 50, Top 100, accuracy 还会略有提高，但是考虑到计算量将会增加很多，就只取到 Top 30 类别进行预测。

先采用 ResNet50 [2] 模型预测，如果图片预测结果 Top30 分类 list 中没有猫或者狗的类别，判断该图片是异常图片。第一次挑出来的异常图片比较多，有些是误判。再分别采用 InceptionV3 [1], Xception [3] 模型预测，以同样的方式在之前选出的异常图片集合里挑出异常图片。共进行 3 次判断挑选，减少误判和异常图片的数量。

最终从训练集猫的图片挑出 46 张异常图片如下



从训练集狗的图片挑出 18 张异常图片如下



3. 要切分成 train, validation 2 个部分。通常把 80% 样本作为 train, 20% 作为 validation. 切分的时候要注意 train, validation 也要保持猫和狗的样本平衡, 以免最终学习到的特征偏向于狗或者猫。
4. 要把样本随机打乱, 排除样本顺序对训练学习产生影响, 提高模型的泛化能力。相当于洗牌, 如果发的牌不好, 体现不出真实水平。

3.3 迁移学习模型构建

1. 先采用 InceptionV3[1] 模型, 作为迁移学习的 base model。
 - 去掉 top layer, 采用 imagenet 预训练权值, 且冻结权值, Global Average Pooling 导出特征量。
 - InceptionV3 模型的默认输入图片 size 是 299x299, TensorFlow 后端采用 channels_last 输入维度顺序。即 input_shape=(299, 299, 3)
 - 增加 Dropout, 随机失活。每个 batch 训练时, 以 0.5 的概率连接后面的全连接层, 而以 0.5 的概率丢弃连接。相当于训练时随机选择部分特征, 而不是全部特

征。Dropout 有效地防止过拟合。

- 全连接层的激活函数采用 sigmoid, 猫的概率接近 0, 狗的概率接近 1。
- 考虑到我的 GPU 显存 total 只有 6GB, 没办法一次载入所有 train, validation 图片, 要分 batch 传入图片数据。Generator 不断循环, 每次产生 1 个 batch_size 的数据, 作为神经网络的 input_tensor。经过实验 batch_size=50 比较合适, 如果 GPU memory 容量大, batch_size=100 也可以。Batch_size 太小, 训练的次数多, 效率低, 且容易造成训练后期 loss, accuracy 值震荡, 不易收敛。
- 数据预处理, InceptionV3 模型要求对输入数据进行归一化处理, 处理后范围(-1, 1)
- EarlyStopping 也是一种防止过拟合的方法, 当 val_loss 不再降低时, 停止训练。

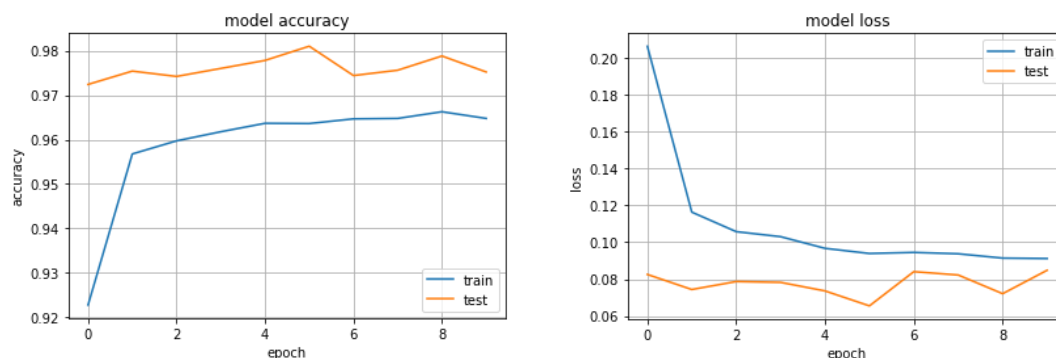
```
earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')

model_InceptionV3.compile(loss = 'binary_crossentropy', optimizer = 'Adam', metrics=['accuracy'])
model_InceptionV3_history = model_InceptionV3.fit_generator(train_generator(X_train, y_train, batch_size=batch_size),
                                                           steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                                           validation_data = val_generator(X_val, y_val, batch_size=batch_size),
                                                           validation_steps=len(y_val)//batch_size,
                                                           callbacks = [earlystop],
                                                           verbose=2)
```

```
Epoch 1/10
- 240s - loss: 0.2065 - acc: 0.9227 - val_loss: 0.0825 - val_acc: 0.9724
Epoch 2/10
- 240s - loss: 0.1164 - acc: 0.9567 - val_loss: 0.0743 - val_acc: 0.9754
Epoch 3/10
- 240s - loss: 0.1058 - acc: 0.9597 - val_loss: 0.0787 - val_acc: 0.9742
Epoch 4/10
- 240s - loss: 0.1031 - acc: 0.9617 - val_loss: 0.0782 - val_acc: 0.9760
Epoch 5/10
- 240s - loss: 0.0967 - acc: 0.9637 - val_loss: 0.0736 - val_acc: 0.9778
Epoch 6/10
- 240s - loss: 0.0939 - acc: 0.9636 - val_loss: 0.0654 - val_acc: 0.9810
Epoch 7/10
- 240s - loss: 0.0945 - acc: 0.9646 - val_loss: 0.0840 - val_acc: 0.9744
Epoch 8/10
- 240s - loss: 0.0938 - acc: 0.9648 - val_loss: 0.0822 - val_acc: 0.9756
Epoch 9/10
- 240s - loss: 0.0914 - acc: 0.9663 - val_loss: 0.0721 - val_acc: 0.9788
Epoch 10/10
- 240s - loss: 0.0912 - acc: 0.9648 - val_loss: 0.0848 - val_acc: 0.9752
```

验证集最好结果 0.0654 - val_acc: 0.9810

可视化训练曲线如下图。



2. 采用 ResNet50 [2] 模型，进行迁移学习。

需要注意 ResNet50 [2] 2 个不同点

- ResNet50 [2]模型的默认输入尺寸时 224x224，TensorFlow 后端采用 channels_last 输入维度顺序。即 input_shape=(224, 224, 3)
- ResNet50 [2]的数据预处理 是减去均值，除以方差。不同于 InceptionV3 模型。

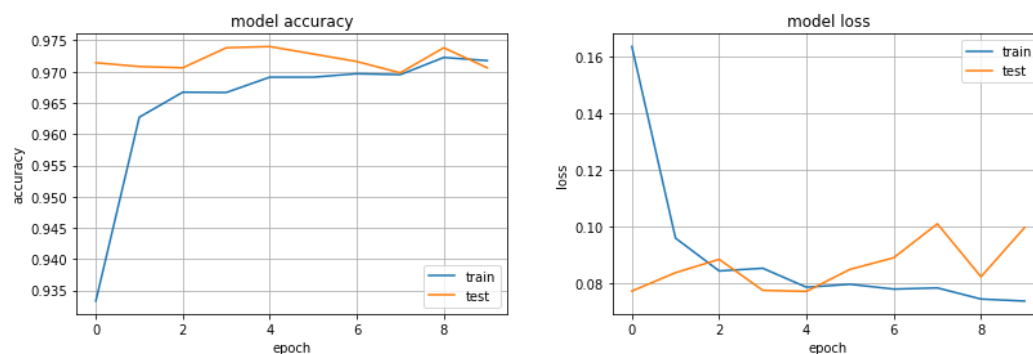
```
earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')

model_ResNet50.compile(loss = 'binary_crossentropy', optimizer = 'Adam', metrics=['accuracy'])
model_ResNet50_history = model_ResNet50.fit_generator(train_gen_224(X_train, y_train, batch_size=batch_size),
                                                    steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                                    validation_data = val_gen_224(X_val, y_val, batch_size=batch_size),
                                                    validation_steps=len(y_val)//batch_size,
                                                    callbacks = [earlystop],
                                                    verbose=2)
```

```
Epoch 1/10
- 179s - loss: 0.1637 - acc: 0.9333 - val_loss: 0.0772 - val_acc: 0.9714
Epoch 2/10
- 177s - loss: 0.0959 - acc: 0.9627 - val_loss: 0.0837 - val_acc: 0.9708
Epoch 3/10
- 176s - loss: 0.0843 - acc: 0.9667 - val_loss: 0.0884 - val_acc: 0.9706
Epoch 4/10
- 176s - loss: 0.0853 - acc: 0.9667 - val_loss: 0.0775 - val_acc: 0.9738
Epoch 5/10
- 176s - loss: 0.0786 - acc: 0.9691 - val_loss: 0.0771 - val_acc: 0.9740
Epoch 6/10
- 176s - loss: 0.0796 - acc: 0.9691 - val_loss: 0.0849 - val_acc: 0.9728
Epoch 7/10
- 178s - loss: 0.0779 - acc: 0.9697 - val_loss: 0.0890 - val_acc: 0.9716
Epoch 8/10
- 178s - loss: 0.0783 - acc: 0.9695 - val_loss: 0.1010 - val_acc: 0.9698
Epoch 9/10
- 177s - loss: 0.0744 - acc: 0.9723 - val_loss: 0.0823 - val_acc: 0.9738
Epoch 10/10
- 176s - loss: 0.0737 - acc: 0.9718 - val_loss: 0.0997 - val_acc: 0.9706
```

验证集最好结果 val_loss: 0.0771 - val_acc: 0.9740, accuracy 和 loss 比 InceptionV3 迁移模型略差。

可视化训练曲线如下图



3. 采用 Xception [3] 预训练模型，来构建迁移学习。

- Xception [3] 模型默认输入图片大小为 299x299
- 数据预处理和 InceptionV3 [1] 相同

```

earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')

model_Xception.compile(loss = 'binary_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
model_Xception_history = model_Xception.fit_generator(train_generator(X_train, y_train, batch_size=batch_size),
                                                    steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                                    validation_data = val_generator(X_val, y_val, batch_size=batch_size),
                                                    validation_steps=len(y_val)//batch_size,
                                                    callbacks = [earlystop],
                                                    verbose=2)

```

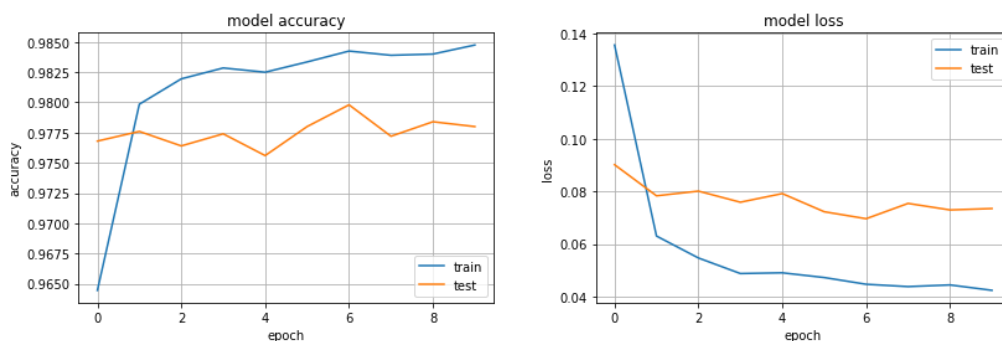
```

Epoch 1/10
- 367s - loss: 0.1356 - acc: 0.9645 - val_loss: 0.0901 - val_acc: 0.9768
Epoch 2/10
- 364s - loss: 0.0630 - acc: 0.9799 - val_loss: 0.0783 - val_acc: 0.9776
Epoch 3/10
- 364s - loss: 0.0546 - acc: 0.9820 - val_loss: 0.0801 - val_acc: 0.9764
Epoch 4/10
- 364s - loss: 0.0487 - acc: 0.9829 - val_loss: 0.0758 - val_acc: 0.9774
Epoch 5/10
- 364s - loss: 0.0490 - acc: 0.9825 - val_loss: 0.0791 - val_acc: 0.9756
Epoch 6/10
- 364s - loss: 0.0472 - acc: 0.9834 - val_loss: 0.0722 - val_acc: 0.9780
Epoch 7/10
- 363s - loss: 0.0447 - acc: 0.9843 - val_loss: 0.0696 - val_acc: 0.9798
Epoch 8/10
- 364s - loss: 0.0438 - acc: 0.9839 - val_loss: 0.0754 - val_acc: 0.9772
Epoch 9/10
- 364s - loss: 0.0444 - acc: 0.9840 - val_loss: 0.0729 - val_acc: 0.9784
Epoch 10/10
- 363s - loss: 0.0424 - acc: 0.9848 - val_loss: 0.0734 - val_acc: 0.9780

```

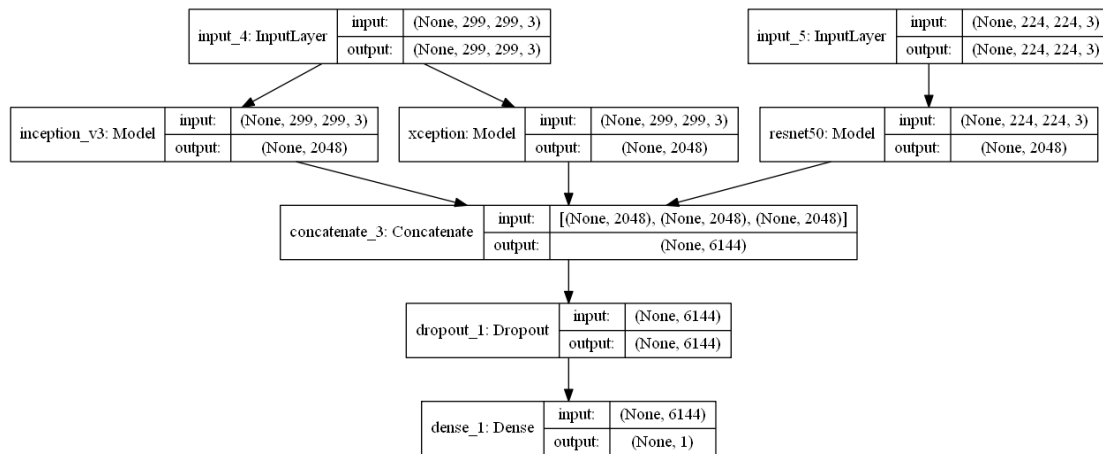
验证集最好结果 val_loss: 0.0696 - val_acc: 0.9798, accuracy 和 loss 介于 InceptionV3 迁移模型, ResNet50 迁移模型两者之间

训练曲线如下图



综上所述, InceptionV3 [1], ResNet50 [2], Xception [3] 3 个模型各有特点。单独作为 base model 构建迁移学习的训练准确度大致相当, 那么我们就集成 3 个模型来提高准确度。

由于 ResNet50 [2] 的 input shape, 预处理方式不同, 要生成 2 个 input tensor, 端到端的集成模型结构图如下



集成模型的构建方法，实现代码如下。

```
# in case, computer crash in a specified epoch, save checkpoint of model
# restore a previous model to continue training from the specified epoch.
# tune_model = Load_model('./model/tune_model_V1.h5')
if not isdir('model'):
    os.mkdir('model')
checkpointer = ModelCheckpoint('./model/tune_model_V1.h5', monitor='val_loss', verbose=0, save_best_only=True,
                               save_weights_only=False, mode='auto', period=1)

earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')

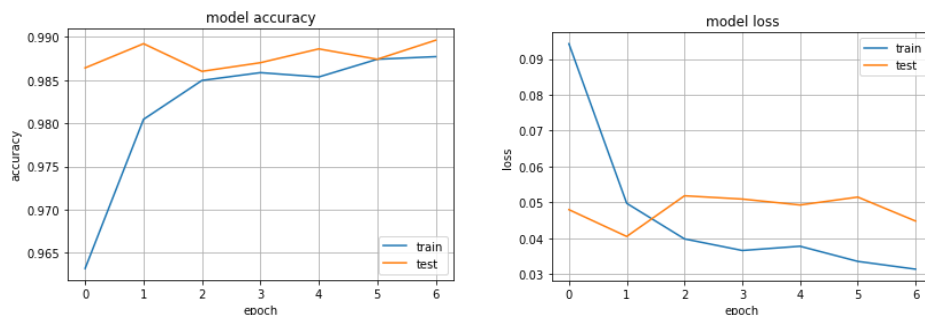
logger = CSVLogger('./model/log_V1.csv', separator=',', append=True)

# if restore previous model, may train from specified epoch by parameters "initial_epoch= "
model_history = tune_model.fit_generator(train_generator(X_train, y_train, batch_size),
                                         steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                         validation_data = val_generator(X_val, y_val, batch_size),
                                         validation_steps=len(y_val)//batch_size,
                                         callbacks = [earlystop,checkpointer,logger],
                                         verbose=2)
```

```
Epoch 1/10
- 777s - loss: 0.0942 - acc: 0.9632 - val_loss: 0.0480 - val_acc: 0.9864
Epoch 2/10
- 770s - loss: 0.0498 - acc: 0.9805 - val_loss: 0.0405 - val_acc: 0.9892
Epoch 3/10
- 770s - loss: 0.0398 - acc: 0.9850 - val_loss: 0.0518 - val_acc: 0.9860
Epoch 4/10
- 770s - loss: 0.0366 - acc: 0.9859 - val_loss: 0.0509 - val_acc: 0.9870
Epoch 5/10
- 770s - loss: 0.0378 - acc: 0.9854 - val_loss: 0.0493 - val_acc: 0.9886
Epoch 6/10
- 774s - loss: 0.0336 - acc: 0.9874 - val_loss: 0.0515 - val_acc: 0.9874
Epoch 7/10
- 772s - loss: 0.0314 - acc: 0.9877 - val_loss: 0.0448 - val_acc: 0.9896
```

训练最好结果 val_loss: 0.0405, val_acc: 0.9892, 比单独采用 InceptionV3 [1], ResNet50 [2], Xception [3] 迁移模型有显著提高, val_loss 下降 0.02-0.03, val_acc 提高 0.1 左右

训练曲线如下图



可以看出已经过拟合，第 3 个 epoch 开始, val-loss 增加, val-accuracy 降低。无法再有效地学习猫和狗的分类特征，提高验证集上的准确度。

为了进一步提高模型的泛化能力，对训练集的图像进行数据增强，使用下面这些数据增强方法

1. 随机剪裁
2. 随机旋转
3. 随机剪切
4. 水平翻转
5. 垂直翻转
6. 颜色，饱和度，明暗变换
7. 随机变换参数为剪裁比例 0.9，旋转角度 20，剪切角度 20，水平翻转(True)，垂直翻转(True)，颜色 20，饱和度 0.2，明暗变换 0.2。

```
# in case, computer crash in a specified epoch, save checkpoint of model
# restore a previous model to continue training from the specified epoch.
# tune_model = Load_Model('./model/tune_model_V1.h5')
if not isdir('model'):
    os.mkdir('model')
checkpointer = ModelCheckpoint('./model/tune_model_V2.h5', monitor='val_loss', verbose=0, save_best_only=True,
                              save_weights_only=False, mode='auto', period=1)

earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='auto')

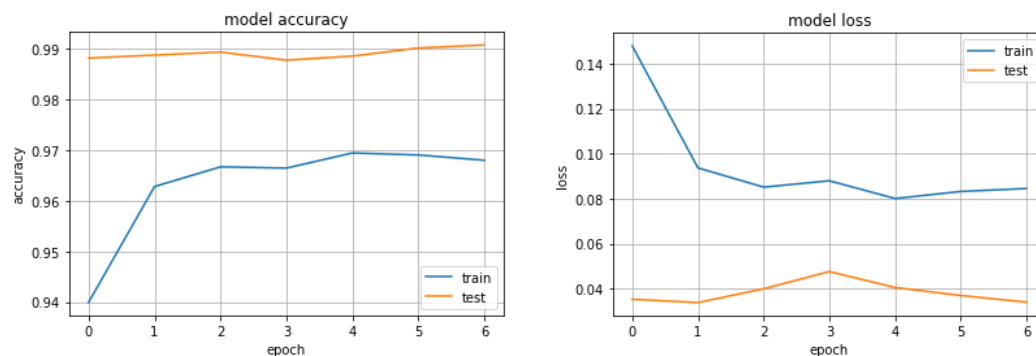
logger = CSVLogger('./model/log_V2.csv', separator=',', append=True)

# if restore previous model, may train from specified epoch by parameters "initial_epoch="
model_history = tune_model.fit_generator(train_generator(X_train, y_train, batch_size,
                                                         0.9,20,20,True,True,(20,0.2,0.2)),
                                       steps_per_epoch=len(y_train)//batch_size, epochs=epochs,
                                       validation_data = val_generator(X_val, y_val, batch_size),
                                       validation_steps=len(y_val)//batch_size,
                                       callbacks = [earlystop,checkpointer,logger],
                                       verbose=2)
```

Epoch 1/10
- 777s - loss: 0.1479 - acc: 0.9399 - val_loss: 0.0353 - val_acc: 0.9882
Epoch 2/10
- 771s - loss: 0.0937 - acc: 0.9628 - val_loss: 0.0338 - val_acc: 0.9888
Epoch 3/10
- 771s - loss: 0.0851 - acc: 0.9667 - val_loss: 0.0399 - val_acc: 0.9894
Epoch 4/10
- 772s - loss: 0.0879 - acc: 0.9665 - val_loss: 0.0476 - val_acc: 0.9878
Epoch 5/10
- 771s - loss: 0.0801 - acc: 0.9695 - val_loss: 0.0405 - val_acc: 0.9886
Epoch 6/10
- 770s - loss: 0.0832 - acc: 0.9691 - val_loss: 0.0370 - val_acc: 0.9902
Epoch 7/10
- 762s - loss: 0.0845 - acc: 0.9680 - val_loss: 0.0340 - val_acc: 0.9908

验证集最好结果 val_loss: 0.0340 - val_acc: 0.9908，比第一个集成版本(未做图片随机变换)有微小提高。

训练曲线如下图，可以看到过拟合略有改善。



4. 结果

综合各个迁移模型的结果如下表

Model Name	train_acc	train_loss	val_acc	val_loss	Kaggle LogLoss
InceptionV1(transfer)	0.9636	0.0939	0.981	0.0654	no submission
ResNet50(transfer)	0.9691	0.0786	0.974	0.0771	no submission
Xception(transfer)	0.9843	0.0447	0.9798	0.0696	no submission
V1(merge)	0.98045	0.049754	0.9892	0.040507	0.05278
V2(merge+data augmentation)	0.968	0.084518	0.9908	0.034015	0.05118

可以把卷积神经网络模型的训练过程，通俗地比拟成学生高考。

Train 相当于学生参加高考前的模拟考试成绩，Validation 相当于高考成绩。

平时的模拟训练，题库越大（样本空间），随着学生做的模拟卷增加（epochs），高考成绩会提高。

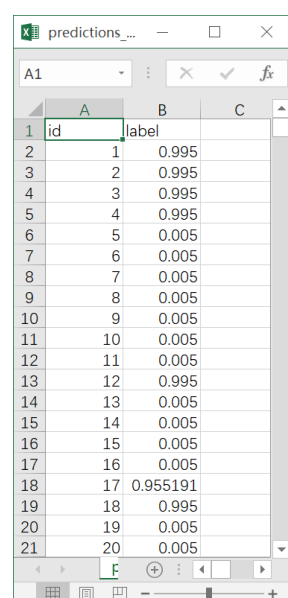
而如果学校提供的模拟卷风格相同，则做多了效果也有限。学生把题目都快背下来，就是过拟合。

这时需要引进不同的题库和模拟卷风格，让学生适应不同的出题风格。就相当于数据增强，可以在提高学生高考成绩的稳定。相当于抑制过拟合的效果。

当然，如果满分 100，从 97 稳定提高到 99，甚至 99 以上会越来越难。要求每次考 100 分几乎不可能。

采用数据增强 Data Augmentation 的最终集成模型 V2，Logloss 0.05118，可以排到 Top 68/1314

对测试集 15000 张图片的预测概率如下表。



	A	B	C
1	id	label	
2	1	0.995	
3	2	0.995	
4	3	0.995	
5	4	0.995	
6	5	0.005	
7	6	0.005	
8	7	0.005	
9	8	0.005	
10	9	0.005	
11	10	0.005	
12	11	0.005	
13	12	0.995	
14	13	0.005	
15	14	0.005	
16	15	0.005	
17	16	0.005	
18	17	0.955191	
19	18	0.995	
20	19	0.005	
21	20	0.005	

5. 总结

通过这个猫狗图片识别的项目，从问题描述，数据分析，构建模型，结果预测系统的学习了运用 卷积神经网络模型 解决实际问题。以在 ImageNet 预训练模型 InceptionV3[1], ResNet50[2], Xception[3] 作为 base model, 构建端到端的深度神经网络模型。经过超参数调整后，取得不错的效果 Log Loss 0.05118。要注意 数据增强方法不能大幅度破坏图片的分类特征，比如随机剪裁可能会把关键特征的部分裁掉，影响特征学习的结果。旋转、剪切角度不能太小，否则无法取得实际效果。样本随机打乱是必要的，否则样本的分布特性会影响模型学习结果。

分析测试集图片最终预测结果，其中预测错误的，或者概率偏差较大的图片是有问题的。即使是人来判断也会难以判断，举例如下：

图片 32 预测概率 0.553，只有背面，真的看不到猫和狗的特征。



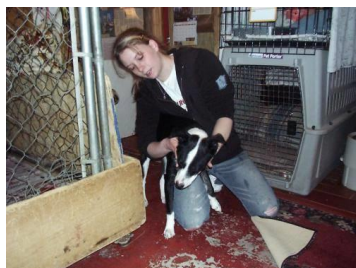
图片 813 预测概率 0.258，误判为猫，背景干扰造成的。



图片 1267，预测概率 0.0132，判断为猫，这个图片本来就有问题



图片 1384，预测概率 0.171，虽然是误判为猫。不过图片不清晰，很多地方被遮挡。



图片 2636，预测概率 0.062，完全看不清图片中是猫还是狗



图片 3015, 预测概率 0.316 判断为猫, 背景干扰太大, 狗某些特征被遮盖。



值得思考, 后续有待改进的地方

1. 这个项目中的数据增强每次随机选取一种变换(剪裁 Crop, 旋转 Rotate, 剪切 Shear, 水平翻转, 垂直翻转, 颜色变换)对训练结果提升不多。还需要再尝试增加其它数据增强的方法, 如 Gamma 变换, 随机噪声等, 并考虑几种变换的组合操作。
2. 这个项目采用的合并 InceptionV3[1], ResNet50[2], Xception[3] 的输出特征向量后, 增加 Dropout, 全连接层构建的端到端的集成模型。最终训练结果在 Kaggle total 1314 提交榜单, 只排到 Top 68, 且训练过程 val_loss 有明显的震荡, 第一轮之后基本没有显著降低。还需要尝试其它抑制过拟合的方法, 进一步提高预测准确度, 减少 Logloss。

参考文献

[1] InceptionV3

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna
Rethinking the Inception Architecture for Computer Vision
arXiv:1512.00567

[2] ResNet50

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Deep Residual Learning for Image Recognition
arXiv:1512.03385

[3] Xception

François Chollet
Deep Learning with Depthwise Separable Convolutions
arXiv:1610.02357

[4] tensorflow 中文社区

http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html

[5] Keras:基于 Python 的深度学习库

<http://keras-cn.readthedocs.io/en/latest/>