

猫狗大战

机器学习毕业项目报告

Patrick Zhong (钟远东)

2018年3月9日

1. 问题定义

概述

本项的目的是通过深度学习技术来让计算机去区别猫狗图片。深度学习是机器学习的一个重要分支，而机器学习，简单来说就是让算法来进行自我学习，从训练过程中得到经验并自我改进。机器学习的应用有很多，例如预测数据、自动驾驶、修改图像、还有本项目所要完成的图像识别等。机器学习是现今发展最快和价值最高的领域之一，而深度学习则代表了这领域中最前沿和最受瞩目的分支之一。

深度学习的应用场景中，图像识别的应用最为广泛，涵盖了保安设备、医疗、农业、和生产业等。图像识别技术可以有效代替人类重复用肉眼筛选的工作，如帮助医生判断眼疾、或帮助农业人员筛选农作物。除此以外，图像识别技术也默默地存在日常生活中，例如Facebook的景点识别功能，还有iPhone手机相册里的头像分类功能。



这次需要完成的任务虽然貌似简单，只是纯属地识别猫猫狗狗，可是，如果能计算机能被训练去区别动物，那也代表计算机可以被训练去识别其它东西，例如识别网络上色情和暴力的资讯，从而保护下一代孩子的思想发育。所以，深度学习的图像识别技术是有趣而有意义的。

问题陈述

本项目要解决的问题是让计算机学习识别猫和狗、并准确地把这两个类别区分开来。从专业的技术角度来说，这是一个计算机视觉二分类的问题。其实，机器学习的原理跟人类的学习过程很像，举个例子，除了天生异禀的转世灵童外，一般小孩子都需要通过学习来对世界万物有所认知。做个比喻，有一天，一个小孩子看到一只追着老鼠飞奔的小动物，就问妈妈，那只长尾巴的东西叫什么，为什么要追着老鼠跑呢。妈妈说，噢，孩子，那叫猫，它的天职是抓老鼠。孩子继续问，那猫猫跳来跳去，不怕受伤么？妈妈接着说，噢，孩子，不怕，因为猫猫有九条命。这时，小孩子就会记着，有着长尾巴喜欢抓老鼠并有九条命的小动物叫猫猫。过了几天，小孩子碰到一只带着长尾巴的动物向他汪汪地叫，就问妈妈，为什么这只猫要叫呢，把他的小心肝都快吓掉了。妈妈说，噢，傻孩子，那不叫猫，那叫狗狗。小孩子不解，问道，不是有长尾巴毛茸茸的就叫猫么？妈妈答，孩子，你年纪尚轻，不是所有有尾巴的都叫猫，狗是汪汪地叫，而猫是喵喵地叫。听道，孩子点点头，表示懂了。当然，这是一个单纯的比喻，实际上，人类的大脑是一个深奥的超级计算机。例如，孩子除了从妈妈的教诲中学习，他的大脑里的神经元也会不停地工作，自动从日常的观察中去识别并区分动物的各种特征，如身体形状、毛的颜色、眼睛的大小、嘴巴的长短等等。所以，人类能轻松地通过眼睛，不需要太多思考，只靠潜意识和直觉就能识别不同的东西。



图片来源：SENAMADUREIRA.COM

那计算机又如何通过深度学习来进行图像识别呢？其实，深度学习的基本原理就是借鉴人类大脑的神经元的工作原理。首先，需要准备大量的数据（猫狗图片 + 标签），用来让机器训练和学习，就像小孩子要看过猫猫狗狗才可能知道什么叫猫猫狗狗。然后，构建一个神经网络模型（也可以叫算法），让机器自动通过重复尝试、犯错、和反省的过程中学习和改进，就像小孩子错把狗当成猫，然后给妈妈指正一样。最后，等机器完成了训练，让机器去识别一些它没有看过的图片（一般叫验证集），来评估算法的表现。如果表现不佳的话，对算法和训练集数据进行各种的优化和调整，直到结果满意为止。

出于学习目的，这个项目会尽可能用简单、易懂、和高效的方法地完成，另外，避免把问题复杂化，从而达到学习的目的和节省时间成本。

评价指标

衡量解决方案以log loss为标准, log loss约小越好。

- n 是测试集里面的图片数量

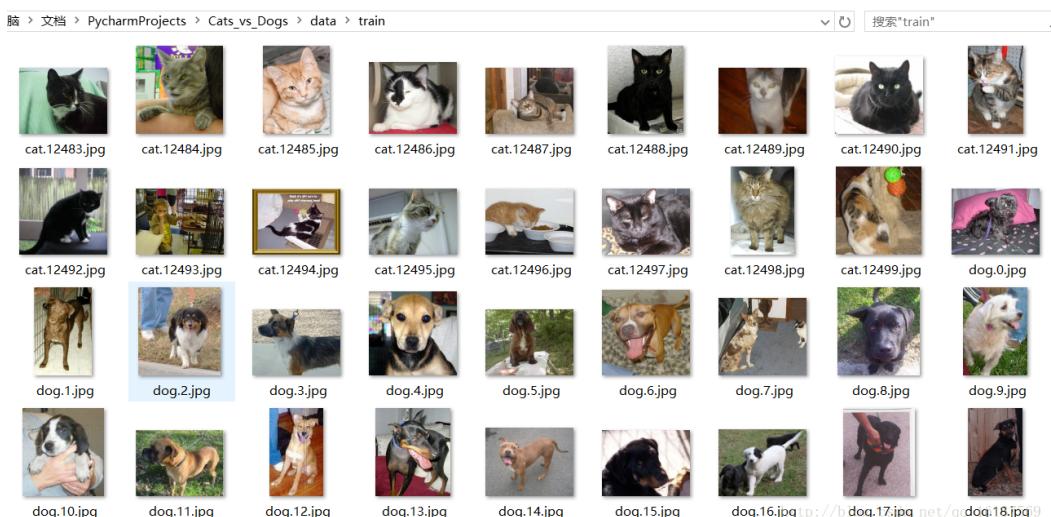
$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)],$$

- \hat{y}_i 是图片为狗狗的预测机率
- y_i 是1的话, 代表图片是狗狗, 是0的话, 代表是猫猫
- $\log()$ 是底数为e的自然对数

2. 分析

数据的探索

本项目采用Kaggle[猫狗大战](#)比赛的图片集，下载地址可以从readme文档获取。图片集分成两个文件夹，一个是train文件夹，一个是test文件夹。train文件夹一共包含了25000张猫狗图片，每张图片文件名开头带有label名称（例如cat.107.jpg或dog.3023.jpg）。而test文件夹包含了12500猫狗图片，图片以id数字来命名而没有label。test数据是最终验证模型用的，所以可以暂时忽略，需要着重分析的是train文件夹里的图片。

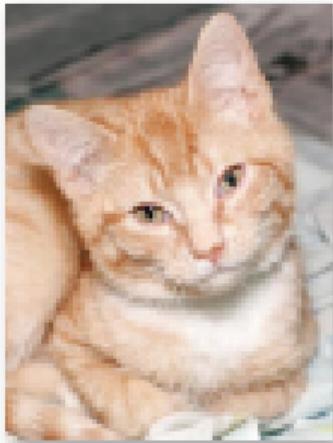


从数量来看, train文件夹里的猫和狗的图片数量均等, 分别为12500张。从图片的大小来说, 如果以宽度为基准来排列, 范围为:

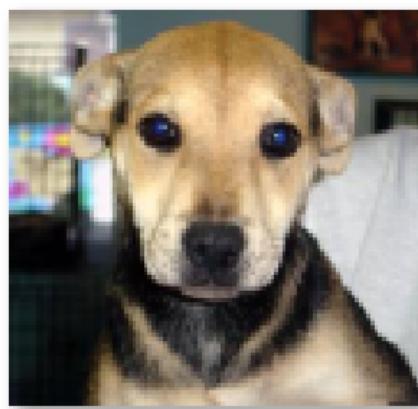
- cat: 50x49 到 1023x768 (宽度x高度)
- dog: 42x62 到1050x702 (宽度x高度)

对于尺寸不一，后面可以通过设置Keras的flow_from_directory()方法里面的target_size参数，把输入统一转成224x224。至于如何分割训练集和验证集，由于Model.fit_generator()没有自动split功能，所以只能将train文件夹里面的猫和狗图片分别按比例（20%）分到另外一个validation文件夹。

训练图集大部分的猫狗图片都是非常清晰的，图片里面很清楚可以看到动物的各个特征，例如鼻子、胡子、眼睛、轮廓等等。而且动物占据了图片的主要面积，没有太多繁杂的背景或其它无关物品。



cat.12485.jpg



dog.4.jpg



cat.12489.jpg

可是，从训练集里面可以发现，某部分的图片里动物占据的面积不大，背景也是各式各样，有在屋子里、公园里、马路上、或纯色背景都有。除了环境背景，有些图片里面还有人脸、人手或其他人类身体部位（例如下面左图 - dog.1148.jpg）。其中比较趣的地方是有些图片只有狗的图案，不是真实的狗（例如下面右图 - dog.1259.jpg）。



如果有部分照片不是真实的猫或狗，在一定程度上会影响模型的表现，后面需要在训练模型之前进行数据清洗，把异常图片挑出来。

算法和技术

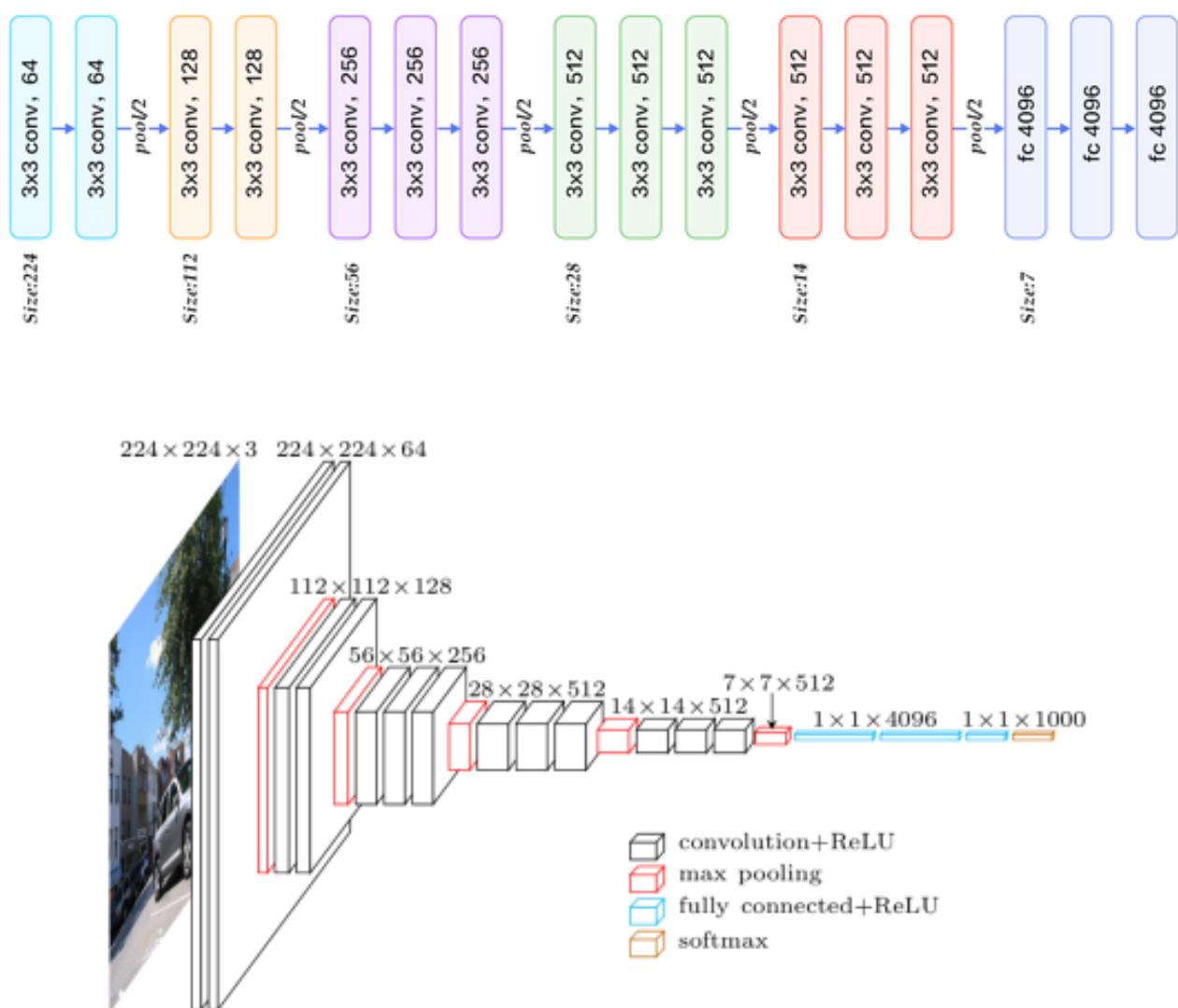
开发工具采用Keras，主要原因是容易上手，而且功能强大。算法方面，出于减低计算成本和提高准确率的考虑，直接从Keras自带的模型里面挑4个已经训练好的模型来作transfer learning：

- VGG16 [1]
- InceptionResNetV2 [2]
- Xception [3]
- ResNet50 [4]

下面是每个模型的一些简单介绍：

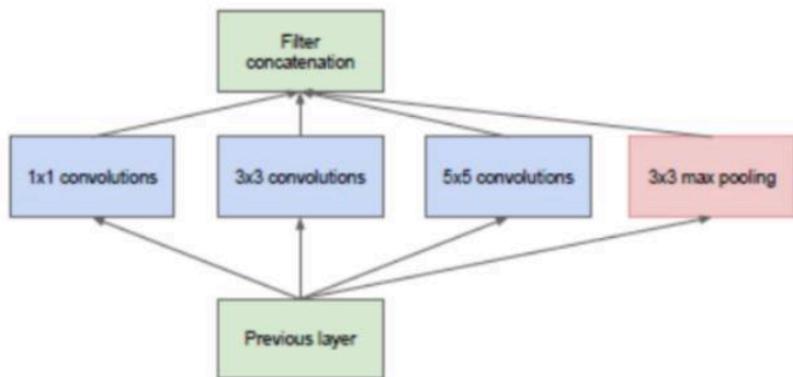
VGG16

VGG的模型概念在2014年的ImageNet Challenge中出现，由牛津可视化图形组（Visual Graphics Group）开发。VGG的结构很直接，就是直接把模型的depth加深。一般层数为16到19，VGG16代表的就是16层，另外有个叫VGG19的就是19层。从下图的第二部分可以看出，模型呈金字塔形，接近图片的那一端（底层）比较宽，而接近输出端（顶层）的那一端比较深。输入的大小同一为 224×224 ，卷积层是比较小的 3×3 。VGG16虽然简单易懂，可能也是比较过时的原因，相比现在主流的模型，VGG16的计算成本非常高，约有1亿多个参数（准确参数量：138,357,544），不过如果用迁移学习的手段，计算成本会大大减低（后文会详细说明）。



InceptionResNetV2

InceptionResNetV2 由 Google 创造并在 2016 年 8 月发布，该模型在 ILSVRC Challenge 图像分类基准测试中拿到了当时最好的成绩。InceptionResNetV2 是从早期的 InceptionV3 模型演化过来的，与其它 Inception 家族成员一样，主要使用了一种称为「Inception 模块」的概念，从下图可以看出，这个结构跟 VGG 的那种序列形架构有很大的不同，在一个层里面，出现了多种特征抽取器（feature extractor），让网络在训练的过程中可以在多个选项中进行选择，例如它可以选择与前一层的输出进行卷积，也可以选择直接池化，间接提升了网络的性能。

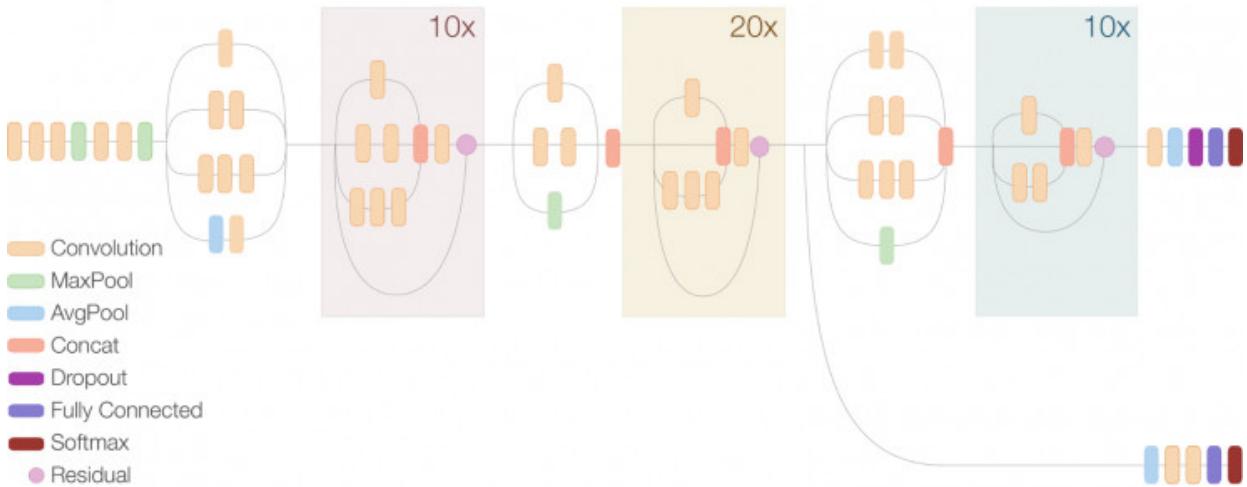


下图第一部分是整个模型的张开图，第二部分是让读者相对容易理解的压缩图。从图中可以看出，这模型的结构比 VGG 复杂多了，depth 也是非常深。不过，InceptionResNetV2 的主打卖点是用相对低的计算成本，约 5 千多万个参数（准确参数量：55,873,736）。来获得优秀的表现。简单来说，这个模型结构比较复杂难懂，不过非常高效。

Inception Resnet V2 Network

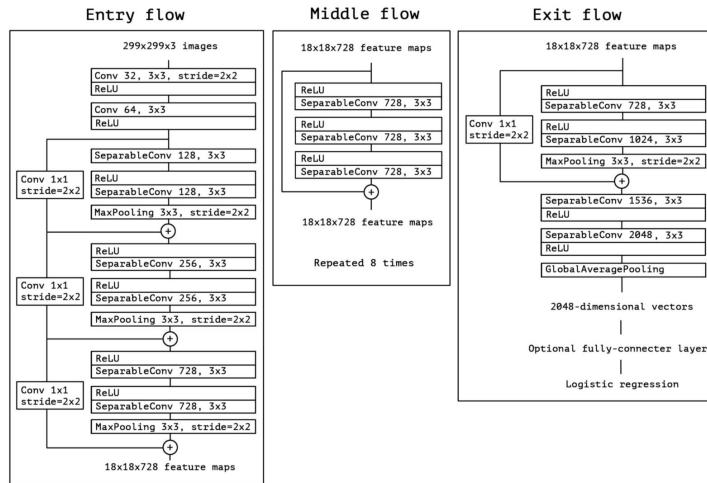


Compressed View



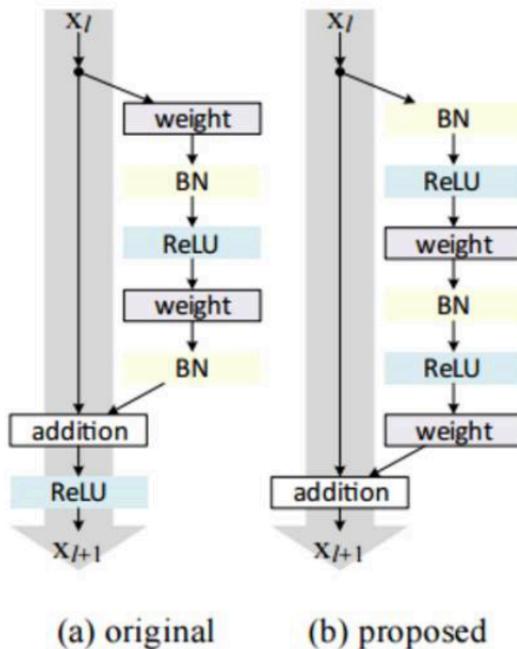
Xception

Xception最早发布于2016年10月，是Google对InceptionV3的另一个改进，Xception的命名代表了「extreme inception」，表示把Inception的概念发挥到极致。模型的改进主要体现在InceptionV3的基础上引进了depthwise separable convolution这个概念，就是下面结构图里的SeparableConv层。Xception的优点在于，相比InceptionV3，在几乎没有增加网络的结构复杂度下提升了模型的表现。相比前面提及的VGG16和InceptionResNetV2，Xception的计算成本更低，参数只有大概两千多万个（准确参数量：22,910,480）。一个比较有趣的事情是，Xception的作者也是Keras的作者Francois Chollet。



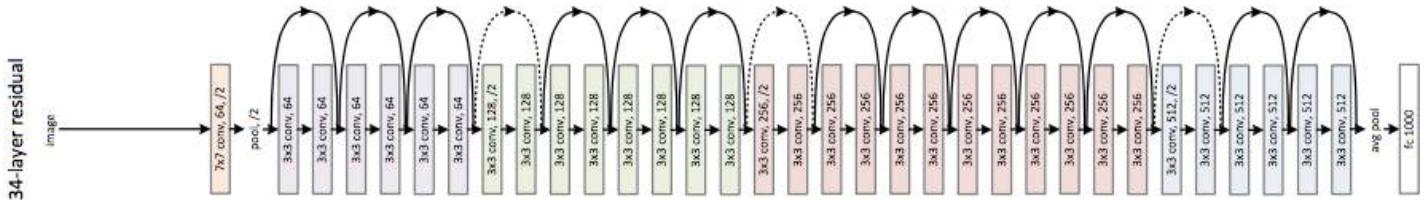
ResNet50

如果说Inception系列的模型是为了更宽，那ResNet就是为了更深，因为ResNet的出现就是为了解决一个问题：当网络的架构非常深的时候，因为梯度消失的问题（梯度信号会在反向传播到更底部的层时指数级地下降），表现反而变得更差。ResNet的架构主要基于残差模块这个概念，下面是示意图：



简单来说，一个残差模块只会做两个选择，要么让输入的数据像VGG那样通过一系列的函数层，要么直接通过「捷径」跳过这个步骤。

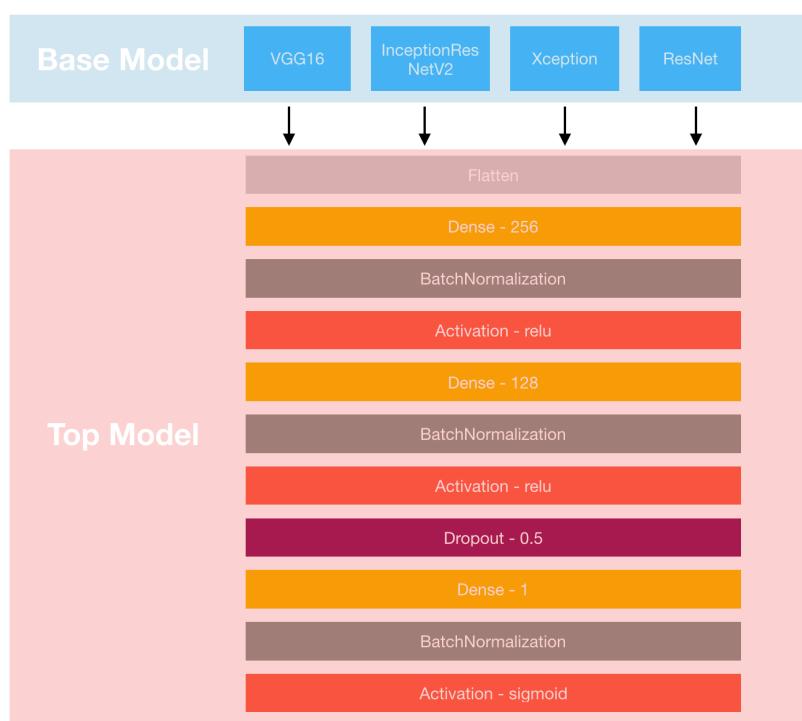
像VGG那样，把很多这样的残差模块连接在一起，就组成了一个完整的ResNet网络。由于网络实在太深，所以下面的结构图做了个90度的转动，方便阅读。



ResNet的优点是可以通过连接数百甚至数千个残差模块来构建一个非常深的网络，而且不会受到梯度消失问题的影响。ResNet50的参数量不算多，大概2千5百万个左右（准确参数量：25, 636, 712）。

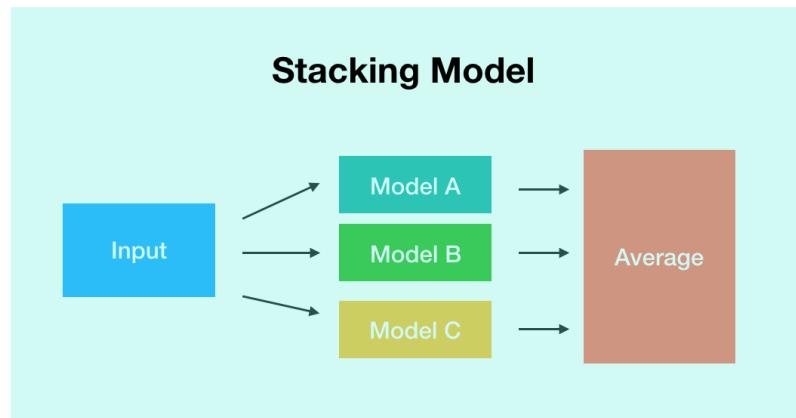
迁移学习

接着，通过Keras来分别把这4个模型的base model（从input layer到最后一个convolutional block）连同pre-trained weights一并导入，然后在上面建立top model。出于简化的原则，先使用统一的top model结构，看看模型表现如何，然后按需微调。具体模型结构请参考下面示意图，需要重新强调一下4个模型是分开训练的，不是同一个模型，但他们的top model部分是一样。下图是结构图：



融合模型

在实践中，每个模型对不同的图片一般会有不同的表现。有一个形象的比喻是，有两个小朋友，一个叫小明一个叫小红。小明对波斯猫比较了解，一看就知道波斯猫是猫的一种，而小红对波斯猫无感，让她看了还以为是只狗。反之，小红家里养了头松狮犬，也喜欢狗狗，所以她能识别不同狗狗的种类，而小明最不喜欢就是狗狗，更不要说让他区别什么是松狮犬了（虽然街上的小黄狗他还是能认得出来的）。我们的模型也一样，由于结构不一，加上其它未知因素，导致它们识别不同的图片都会有不同的表现。而这些差别，恰恰是可以利用的地方。简单来说，就是把这些模型组合起来，然后让它们一起去识别同一张图片，然后把各自的输出平均化，例如模型A的答案是0.9，模型B的答案是0.8，模型C的答案是0.7，那最终答案将是【 $(0.9 + 0.8 + 0.7) / 3 = 0.8$ 】。也可以想象为，三个不同的人，对同一张图片进行“投票”，然后综合他们的答案再来输出一个比较“中肯”的结果，这也算是传说中的三个臭皮匠胜过一个诸葛亮的典型例子。请看下面的结构图，根据简单化原则，选用最简单易懂的stacking 结构，用同一个input layer，然后平行连接不同的模型，把每个模型的输出通过Average layer平均化，得出最终output。



注意上图的模型只有三个，实际操作中数量可以变化。例如这个项目的策略就是创建5个不同的组合，然后提交5个不同的结果到Kaggle，最后选择表现最好的一个。

基准模型

本项目对标Kaggle的猫狗大战竞赛的public leaderboard，模型测试结果争取达到前15名 ($\text{loss} \leq 0.03994$)，最低要求达到leaderboard 前10% ($\text{loss} \leq 0.06149$)。

3. 方法

数据清理

前面提及过图片集里参杂了一些没有猫或狗的图片，所以需要针对这些图片做数据清洗的步骤。对于实现的方法，一般第一时间想到的方法就是肉眼一张张图片地去挑，虽然这方法的确很简单粗暴，

不过要从25000张图片里面一张张地去是一件比较可怕的事情，就算硬着头皮用人工选图的方式完成项目，同样的方法在其他项目不一定能用，因为很多其他项目的图片集可能达到几十万甚至过百万的图片，用人工方式筛选明显是错误和不理智的做法。所以，要用程序员的思维来完成这个工作。

Keras有现成调试好的分类器，并且提供了模型和权重文件。这些分类器是基于[ImageNet](#)，而ImageNet的分类涵盖了1000个物品，注意它的分类不是物种的分类，例如猫或狗，而是具体分类到哪一种狗或猫，例如1000个分类里有180个狗的品种，7个猫的品种。这些分类器可以用来检测图片是否有猫或狗在里面，从而发现异常图片。

首先，把ImageNet的1000个分类里面，把属于猫或狗的id都收集起来。ImageNet具体的分类标签可以在本项目的readme文件里找到网页链接，下面是部分截图：

text: imagenet 1000 class id to human readable labels (Fox, E., & Guestrin, C. (n.d.). Coursera Machine Learning Specialization.)

```
④ imagenet1000\_clsid\_to\_human.txt Raw
1 {0: 'tench, Tinca tinca',
2 1: 'goldfish, Carassius auratus',
3 2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
4 3: 'tiger shark, Galeocerdo cuvieri',
5 4: 'hammerhead, hammerhead shark',
6 5: 'electric ray, crampfish, numbfish, torpedo',
7 6: 'stingray',
8 7: 'cock',
9 8: 'hen',
10 9: 'ostrich, Struthio camelus',
11 10: 'brambling, Fringilla montifringilla',
```

这个文件包含了所有分类的id和标签，不过在里面找出180个狗狗品类的英文标签困难蛮大的，就算是中文的，估计也分不出这么多的狗狗品种来。幸好，有热心的同好把这些分类标签简化成物种品类，例如猫、狗、鸟、羊等等，而且还是中文的，具体的网页链接也在readme文件可以找到。收集好后，直接把id输入到两个数组里面了（cat 和 dog），下面是代码截图：

```
# Create two arrays that contains all the ImageNet codes of various
# kind of dogs and cats

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']
```

然后，通过Keras提供的多个模型中选出三个模型来对异常图片进行检测。选多个模型的主要原因就是前部分提及到的一个臭皮匠胜过一个诸葛亮的原理，希望可以集合几个模型得出的结果，尽可能把更多异常的图片都找出来。

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

上面是来自Keras官方手册的模型数据，其中要着重注意的是Top-1 Accuracy 和 Top-5 Accuracy. 比如有一张猫猫图片，Top-1 Accuracy就是模型判断它最大概率是猫猫的准确率，而且Top-5 Accuracy就是，模型给出5个最大概率的可能分类，而这5个分类中包含正确分类（猫猫）的准确率。由于本项目任务的目的是尽可能找出有猫或狗的图片，所以应该选择Top-5 Accuracy偏高的模型：

1. Xception
2. InceptionResNetV2
3. Resnet50

选择好模型后，用下图的那个函数来把训练集里面的图片一张张地读出来，然后用模型来判断图片是否含有任何品种的猫猫或狗狗，如果没有，就把图片的路径跟文件名append到一个list里面去，直到文件夹的全部图片都检测过了，函数就返回这个list。

```
def get_wrong_pic_list(model, preprocess_input, decode_predictions, top_number=50):
    pic_list = []

    # loop through all the files and make predictions
    for root, dirs, files in os.walk("train/", topdown=True):

        for name in log_progress(files, name="Progress"):
            if name[0] == 'c' or name[0] == 'd':                                # using a progress bar to track the progress
                image_path = root + "/" + name                                    # exclude system hidden file like '.DS_Store'
                img = image.load_img(image_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_preds = decode_predictions(preds, top=top_number)[0]
                is_good_picture = False                                         # assume it is a wrong picture

                # loop through all predictions and
                # check if the prediction contains cats or dogs
                for i in decode_preds:
                    if not is_good_picture:
                        if i[0] in dogs or i[0] in cats:
                            is_good_picture = True
                    else:
                        # break the loop if cat or dog is found
                        break

                # if there is no dog or cat in the picture, then it is a bad picture
                if not is_good_picture:
                    pic_list.append(image_path)

    print(str(len(pic_list)) + " wrong pictures are found!")
    return pic_list
```

有了这个函数，就可以直接加载三个模型来分别判断那些图片是异常的。

```
model = Xception(weights='imagenet')
bad_picture_list = get_wrong_pic_list(model, preprocess_input, decode_predictions)

model = InceptionResNetV2(weights='imagenet')
bad_picture_list_2 = get_wrong_pic_list(model, preprocess_input, decode_predictions)

model = ResNet50(weights='imagenet')
bad_picture_list_3 = get_wrong_pic_list(model, preprocess_input, decode_predictions)
```

得到结果后，用 set() 的方法把三个list整合到一个list，这样可以保证list的元素不会重复。最后，得到了81个异常图片：



从图片列表可以看出，里面有“真正”异常的图片（没有猫或狗的图片），还有一些比较模糊的或者只看到动物一小部分的图片，这些也能算为异常的图片。可除了异常图片以外，有些图片是正常的。一般来说，如果检测出来的异常图片的数量太多（例如几百张），在不太影响测试集的数量下，可以秉持着“有杀错无放过”的理念，直接把图片全部删除。当是这次就只有81张，可以用人工的方式把正常的图片挑出来，只删掉正真的异常图片。下面是人工挑选出来认为不需要删除的图片：

```
good_image_list = [
'dog.6725.jpg', 'dog.7772.jpg', 'dog.7706.jpg', 'dog.6405.jpg',
'dog.1625.jpg', 'dog.3889.jpg', 'dog.4565.jpg', 'dog.11437.jpg',
'dog.4507.jpg', 'dog.3341.jpg', 'dog.2339.jpg', 'dog.11266.jpg',
'dog.10123.jpg', 'cat.2520.jpg', 'cat.9494.jpg', 'cat.11039.jpg',
'cat.5974.jpg', 'cat.4308.jpg', 'cat.712.jpg', 'cat.4842.jpg',
'cat.12476.jpg', 'cat.7599.jpg', 'cat.9391.jpg', 'cat.2337.jpg',
'cat.12203.jpg', 'cat.4520.jpg', 'cat.8921.jpg', 'cat.376.jpg',
'cat.1139.jpg', 'cat.9520.jpg', 'cat.10365.jpg', 'cat.3004.jpg',
'cat.3672.jpg', 'cat.4190.jpg']
```

人工筛选后，要删除的图片一共47张，下面是部分图片的截图。



数据分割

之后，通过Keras提供的ImageDataGenerator和flow_from_directory()来导入数据，把训练集的图片分成不同的文件夹里面去。文件夹的结构如下：

```
data/train/dogs/  
data/train/cats/  
data/validation/dogs/  
data/validation/cats/
```

由于前面删除了一些异常图片，导致数据集的数量少了47张图片，而这些图片里面，我们不清楚到底多少张是猫图，多少张是狗图，所以不太应该用人工的方法来分割图片，而且我是通过SSH来登陆AWS的，也用不了人工的方法。这个问题需要用程序员的思维解决，方法如下：

1. 先定义training set的数量为20000 (cat - 10000张, dog - 10000张)
2. 利用python的os.walk() 和 shutil.move() 来写个脚本，让计算机自动去把图片移动到对应的training set文件夹，例如cat的图片就移动到data/train/cats/。如果data/train/dogs/ 和 data/train/cats/ 各自都有10000张图片后，再把剩余的猫狗图片分别移动到validation set文件夹 (data/validation/dogs/ 和 data/validation/dogs/)。注意，这里图片的移动次序是随机的（不按图片id号），例如cat#1在train文件夹，而cat#5可能在validation文件夹。具体的代码实现可以参考代码文件code.ipynb，下面数据分割实现的主要逻辑代码：

```

# use os.walk to "walk through" all the pictures in train folder (downloaded from Kaggle)
for root, dirs, files in os.walk("train/", topdown=True):
    # get the name of each file
    for name in files:
        file_source = os.path.join(root, name)           # full path of the file

        if name[0] == "c":                             # check if picture name started with "c" -> cat
            if i < train_file_number_cat:
                shutil.move(file_source, os.path.join(train_cat_folder, name))
                i += 1
            else:
                shutil.move(file_source, os.path.join(validation_cat_folder, name))

        if name[0] == "d":                             # check if the picture name started with "d" -> dog
            if j < train_file_number_dog:
                shutil.move(file_source, os.path.join(train_dog_folder, name))
                j += 1
            else:
                shutil.move(file_source, os.path.join(validation_dog_folder, name))

```

数据分割后， 数据的最终分布如下：

```

File count:
Cat traning - 10000
Cat validation - 2477
Dog traning - 10000
Dog validation - 2476

```

猫狗图片的数量都差不多，为了数据的分布平均，和后面除以batch size的时候可以得出整数，再“修剪”一下，把cat和dog的validation set都统一弄成2475张图片。

```

# We will remove some pictures from validation folder so the samples is divisible by batch size
from random import randint

# remove 2 cat validation files randomly
for i in range(0, 2):
    os.remove(validation_cat_folder + cat_val_files[randint(0, len(cat_val_files)-1)])

# remove 1 dog validation files randomly
os.remove(validation_dog_folder + dog_val_files[randint(0, len(dog_val_files)-1)])

cat_val_files = os.walk(validation_cat_folder).__next__()[2]
dog_val_files = os.walk(validation_dog_folder).__next__()[2]
print("Cat validation - %s" % len(cat_val_files))
print("Dog validation - %s" % len(dog_val_files))

Cat validation - 2475
Dog validation - 2475

```

执行过程

数据处理好后，就可以构建并训练模型了。由于前面【算法】部分的构思清晰，执行的过程就很轻松了。回顾一下，后面大致需要做的就是分别训练四个不同的模型，模型的base model是使用Keras提供的并且训练过的模型，top model是使用【算法】部分提及的结构。训练好后，再用这

四个模型组合成5个不同的ensemble models。然后用这5个ensemble models来判断Kaggle的最终的test set 测试集，最后把输出结果上载到Kaggle的官方平台来计算成绩。思路明确后，就可以开始了。

首先，先import所有需要的库、和设置图片集的文件路径。设置batch_size为50，epochs为50。其实 epochs 在实践中不需要50那么多，不过后面设置了一个叫EarlyStopping的callbacks，会自动停止模型训练，后面会再作说明。

```
import numpy as np
import pickle
import os
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense, Input, BatchNormalization, Activation
from keras.models import Model, Input
from keras import applications
from keras import optimizers
from keras.callbacks import EarlyStopping
%matplotlib inline

cat_train_files = os.walk("data/train/cats/").__next__()[2]
cat_val_files = os.walk("data/validation/cats/").__next__()[2]
dog_train_files = os.walk("data/train/dogs/").__next__()[2]
dog_val_files = os.walk("data/validation/dogs/").__next__()[2]

train_sample_size = len(cat_train_files) + len(dog_train_files)
validation_sample_size = len(cat_val_files) + len(dog_val_files)

batch_size = 50
epochs = 50
```

先来搭建VGG16的模型：

1. 创造一个输入大小为244x244的input layer
2. 加载Keras自带的VGG16模型，因为我们要做迁移学习，所以需要设置include_top参数为False。
3. 参考前文【算法】的top model结构，用Keras提供的各种layers来完成搭建
4. 把input layer, base model和top model连接起来，组合成一个model

```
# create the model
model_input = Input(shape=(224, 224, 3))
base_model = applications.VGG16(include_top=False)
# base_model = applications.VGG16(include_top=False, weights="imagenet", input_shape=(224, 224, 3))
x = base_model(model_input)
x = Flatten()(x)
x = Dense(256, use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Dense(128, use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Dropout(0.5)(x)
x = Dense(1, use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation("sigmoid")(x)

model = Model(inputs=model_input, outputs=x)
```

模型组建好后，就需要进行compile了。因为评估标准是log loss，所以loss设置为'binary_crossentropy'，optimizer是比较常用的SGD，learning_rate为0.001，decay为0.0002 (learning rate / epoch = 0.001 / 50)，momentum设置为0.9。

```
learning_rate = 0.001
decay_rate = learning_rate/epochs

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.SGD(lr=learning_rate, decay=decay_rate, momentum=0.9),
              metrics=['accuracy'])
```

模型的搭建基本就完了，之后就要处理数据的输入。对于数据的输入，Keras自带了一个非常简单而强大的ImageDataGenerator。在建立generator的时候，还可以做一个数据增强 (data augmentation)的步骤。数据增强，简单来说就是在generator把图片输入的时候，随机地改变图片的属性。以下面的代码为例，rotate_range = 180 就是图片会随机在0-180度的范围旋转，horizontal_flip=True就是图片会随机的进行平行翻转。由于原始图片集数量不多，为了避免过拟合 (over fitting)，可以把数据增强的力度调大点，一共设置了5个参数，而每个参数的数值都有意调大一点。需要说明一下，数据增强只应用在trainning set，validation只需要设置rescale=1./255就好（把每个pixel的数值转化成0-1之间）。

```
# Add data augmentation
train_data_generator = ImageDataGenerator(
    rescale=1./255,
    rotation_range=180,
    shear_range=0.3,
    zoom_range=0.4,
    horizontal_flip=True,
    vertical_flip=True)

val_data_generator = ImageDataGenerator(rescale=1./255)
```

然后，利用flow_from_directory()来输入文件夹里面的图片，这里可以设置target_size来resize每一张图片为统一尺寸 (224, 224)，至于class_mode就设置为binary，这样分类标签和数值的关系就会为{'cat':0, 'dog':1}，因为generator会默认按文件夹里面的子文件夹的排序来赋值，例如cat排在dog前面，所以cat为0，dog为1。

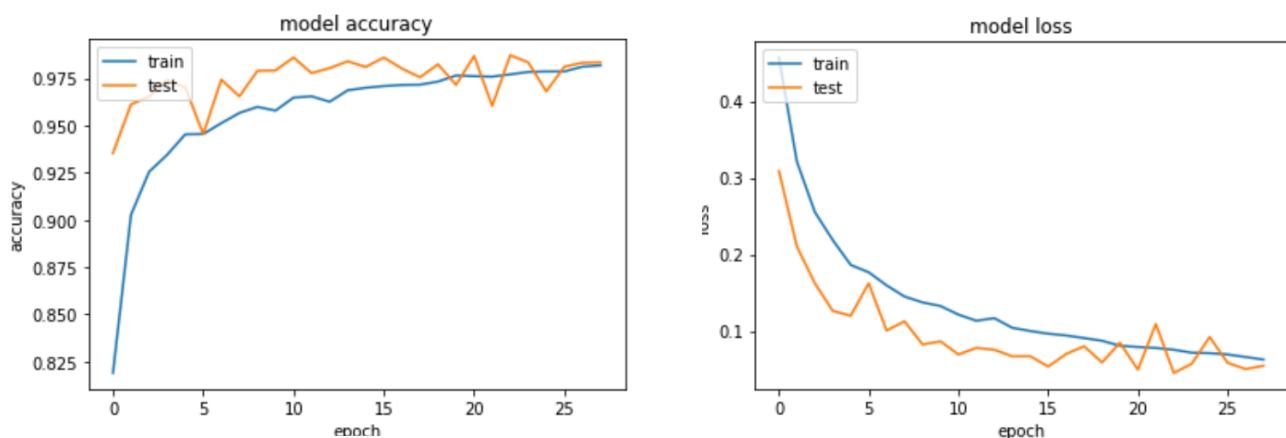
```
# read pictures from 'data/train' folder
train_data = train_data_generator.flow_from_directory(
    'data/train',
    target_size=(224,224),
    batch_size=batch_size,
    class_mode='binary')

# read pictures from 'data/validation' folder
validation_data = val_data_generator.flow_from_directory(
    'data/validation',
    target_size=(224,224),
    batch_size=batch_size,
    class_mode='binary')
```

最后，用`fit_generator()`来训练模型，值得注意的是`callbacks`参数设置了一个`EarlyStopping`，`monitor='val_loss'`和`patience=5`的意思是，如果`val_loss`在5个epochs内都没有减少，就自动停止模型的训练，避免过拟合和节省训练时间。

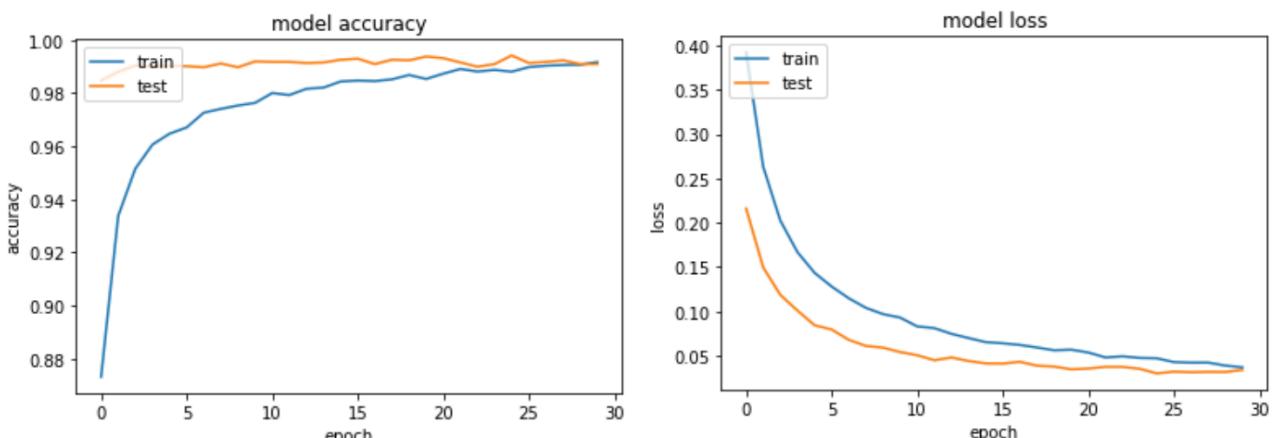
```
history = model.fit_generator(  
    train_data,  
    steps_per_epoch=train_sample_size // batch_size,  
    epochs=epochs,  
    callbacks=[EarlyStopping(monitor='val_loss', patience=5)],  
    validation_data=validation_data,  
    validation_steps=validation_sample_size // batch_size)
```

下图是训练的结果。模型在第28个epoch的时候就自动停下来了，`validation set`上的表现是 {`val_loss: 0.0544, val_acc: 0.9834`}，从下面的曲线图来看，模型应该在过拟合之前就自动停下来了，证明了这个`EarlyStopping`的参数是多么了不起的东西！

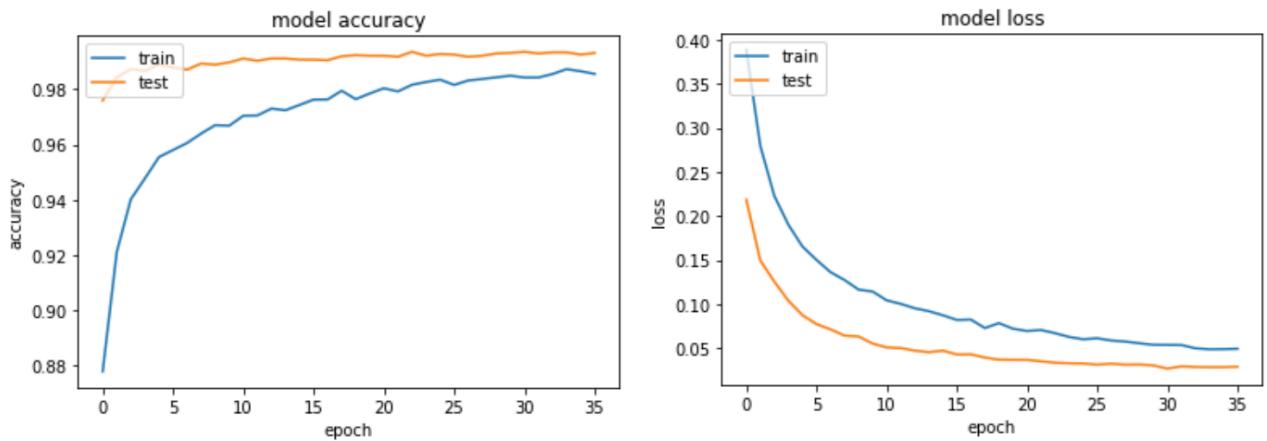


VGG16的训练就完了，后面可以用一摸一样的方法，来训练其他的三个模型，以下是训练结果：

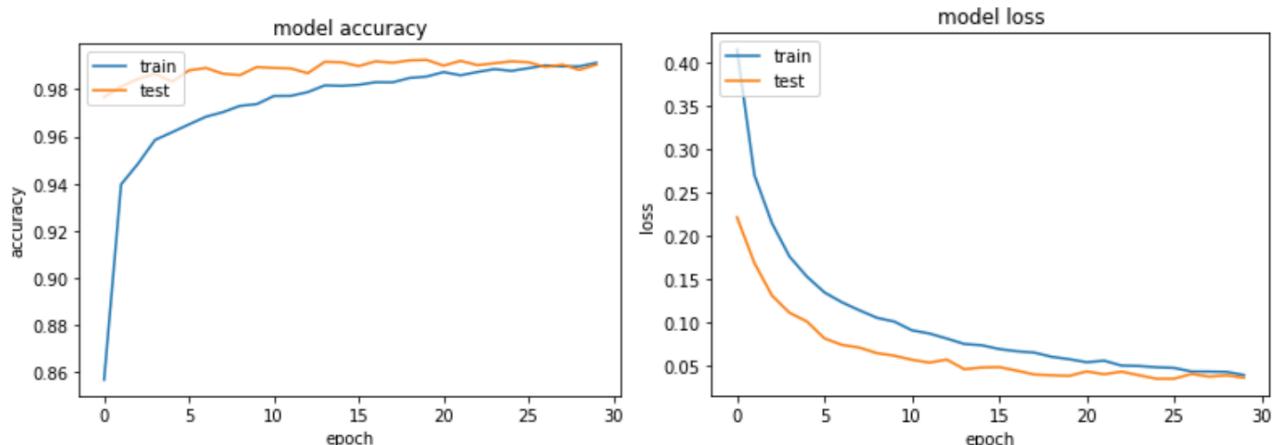
InceptionResNetV2 - { val_loss: 0.0340, val_acc: 0.9909 }



Xception - { val_loss: 0.0290, val_acc: 0.9931 }



ResNet50 - { val_loss: 0.0370, val_acc: 0.9905 }



综合来说，在不多的训练数据和简单的模型结构来说，这4个模型的效果都不错，最低的一个模型 val_loss 可以达到 0.029 (Xception)。如果模型在 test set 上能有这个表现，可以拿到 leadership board 的第一名了，当然，这是不可能的，因为 test set 不会跟 validation set 一摸一样，最终效果还是看模型 generalize 的能力。

完善

之前提到的融合模型 (ensemble model)，可以作为一个简单直接的完善方案。把模型做成 5 个不同的组合，看看能否综合每个模型的能力，来增强模型的 generalize 的能力，并提高模型的测试表现。

首先，把之前保存好的模型加载后，再用数组把它们组合起来。

```

# load all the models saved previously

print("loading vgg16...")
model_vgg16 = load_model('model_vgg.h5')                      # VGG16
print("loading irnv2...")
model_irnv2 = load_model('model_irnv2.h5')                     # InceptionResNetV2
print("loading xception...")
model_xception = load_model('model_xception.h5')              # Xception
print("loading rn50...")
model_rn50 = load_model('model_rn50.h5')                       # ResNet50

```

```

# create model combinations
model_combo_a = [model_vgg16, model_irnv2, model_xception, model_rn50]
model_combo_b = [model_irnv2, model_xception, model_rn50]
model_combo_c = [model_xception, model_rn50]
model_combo_d = [model_irnv2, model_xception]
model_combo_e = [model_irnv2, model_rn50]

```

然后，利用下面的ensemble() 函数来创建ensemble models。这个函数的作用是把输入的参数models分别于输入的参数model_input连接起来，然后把所有models的输出用Average来平均化，最后把组合好的model返回。

```

# define a reusable function to create ensemble model
def ensemble(models, model_input):
    # collect outputs of models in a list
    yModels = [model(model_input) for model in models]

    # averaging outputs
    yAvg = Average()(yModels)

    # build model from same input and avg output
    model = Model(model_input, yAvg, name='ensemble_model')

    return model

```

有了这个ensemble() 函数，就可以很轻松地把5个组合模型都创建起来。注意之前的4个单个模型的输入shape都是(224, 224, 3)，所以创建的model_input都必须一致是(224, 224, 3)。

```

# create model input shared by all models
model_input = Input(shape=(224, 224, 3))

# create ensemble models
ensemble_model_a = ensemble(model_combo_a, model_input)
ensemble_model_b = ensemble(model_combo_b, model_input)
ensemble_model_c = ensemble(model_combo_c, model_input)
ensemble_model_d = ensemble(model_combo_d, model_input)
ensemble_model_e = ensemble(model_combo_e, model_input)

```

评估模型的表现前，需要把模型compile一下。需要说明下，由于后面只需要做evaluate而不是trainning，所以这里的optimizer应该是多余的，可是根据Keras的设计，还是要随便放一个进去，这估计是Keras的维护者没有考虑到的一个小情景。

```
# compile all ensemble models
all_models = [ensemble_model_a,
              ensemble_model_b,
              ensemble_model_c,
              ensemble_model_d,
              ensemble_model_e]

for model in all_models:
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.SGD(lr=0.001),
                  metrics=['accuracy'])
```

下面的每个模型的评估成绩：

```
model A -> loss: 0.031714165272812046, accuracy: 0.9949494991639648
model B -> loss: 0.028225761603074846, accuracy: 0.9957575785993326
model C -> loss: 0.02865432636492481, accuracy: 0.9933333372828936
model D -> loss: 0.028052365435569576, accuracy: 0.995353537978548
model E -> loss: 0.03068202854406954, accuracy: 0.995555559192041
```

最好的成绩是Model D，loss可以去到0.02805，虽然感觉比之前的独立模型Xception的0.029不是低非常多得多，可是我们不要忘记这只是validation set的成绩，最终成绩取决于Kaggle平台计算的test set成绩。按道理来说，ensemble model的generalize的能力应该比单个模型强。

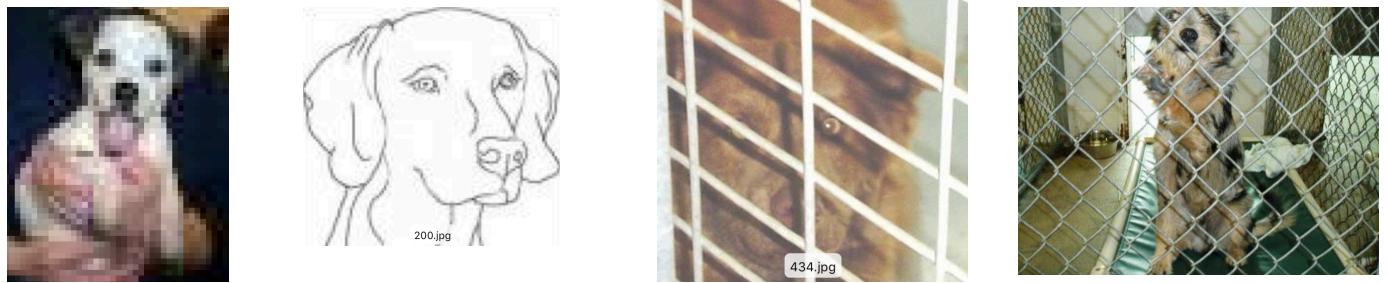
4. 结果

最终模型选择的依据很简单，就是把5个组合模型在test set上得出的结果（csv格式）上传到Kaggle，然后得到最好成绩的那个就是winner了！下面是Kaggle的成绩。

Submission and Description	Public Score	Use for Final Score
pred_b.csv 10 days ago by Patrick Zhong add submission details	0.05185	<input type="checkbox"/>
pred_d-3.csv 10 days ago by Patrick Zhong add submission details	0.05194	<input type="checkbox"/>

果然，test set的成绩跟validation set差一点，最好成绩的两个模型是Model B 和 Model D，所以最终选择稍微胜出的Model B为最终模型，虽然它们没有相差太远。综合来看，这个最终模型是合理的，而且与期待结果一致，因为单个模型在trainning set，test set 和validation的表现虽然有一点

小浮动，可是也差不远，这是正常的情况，代表模型没有出现overfit或underfit的状况，所以，模型的出的结果绝对可信。至于鲁棒性方面，貌似输入数据的一些微笑的变化有点影响模型的结果，例如validation set和test set的表现差异，5个模型在validation set上的表现都优于test set的。其中一个造成这个差异的原因可能是训练集数据有进行过数据（异常值）处理，而测试集没有。检查测试集后返现，的确有些异常的或者太模糊的图片存在里面（请看下图）。虽然影响不大，不过在以后改善模型的时候，怎样提高鲁棒性和generalize的能力是一个值得思考的切入点。



之前提及的基准模型是争取log loss少于0.03994，最低要求也要达到leaderboard 前10% ($\text{loss} \leq 0.06149$)。最终模型在Kaggle的成绩是0.05185。虽然达不到最理想的成绩，可是也达到了leaderboard的前6%，高于项目要求的10%。至于准确率 (Accuracy)，模型也达到了99%以上，所以这个模型用在产品或真实场景上，也能有效地识别猫猫和狗狗，解决问题。

5. 项目结论

项目的主要目的是训练一个深度学习的模型来让计算机去识别猫或狗，训练和测试的图片直接用Kaggle平台的猫狗大战比赛提供的官方数据，而开发工具主要用比较主流的Keras，然后利用Keras提供的4种模型 (VGG16, InceptionResNetV2, Xception, ResNet50) 作为base model，再搭上自己设计的top model，改良成为专门为本项目解决问题的分类器（从分类1000种物品变成只分类猫或狗的模型），最后，把训练好的4个模型组合成5个不同的ensemble model来对测试集进行判断后，把结果以csv格式上传到Kaggle平台来检验分数，最终拿到前6%的成绩，高于项目要求的10%。

其中较有意思和难度的不是建立和训练模型部分，反而是数据清理。在过往的学习中，一般数据都是“干净”的，可是这毕业项目所用的Kaggle数据中，却有很多“异常”的图片，例如下面没有猫狗的图片。



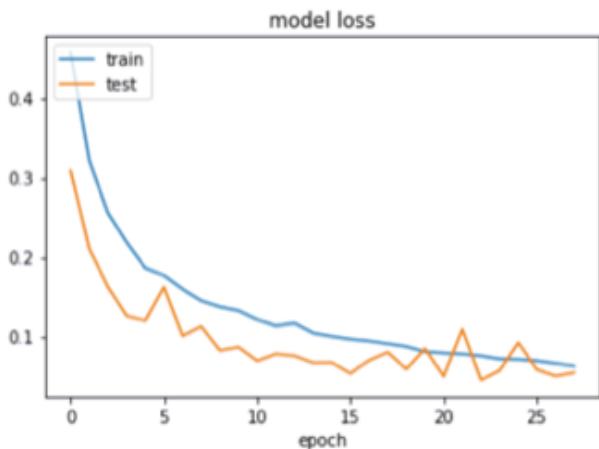
刚开始的时候会让人有点不知所措，因为图片跟纯数字的数据不一样，不容易用统计学的方法来找出异常值。但这也是毕业项目想表现的实践精神，因为在真实世界中，往往连如何有效收集优质数据都是个难题，更不要抱着收集后的数据都是干干净净的美好幻想了，所以要有随机应变的做事心态。

除了要自己清理数据意外，整个项目基本是从零走到一。过往的课程项目除了有该阶段的学习资料，也提供了很多课外资料作为参考。而且，之前项目的notebook文件都是已经搭好的模版，有详细的标注提示甚至提供了部分现成的代码。而在这个毕业项目中，我们需要在基本没有资料和任何代码提供的情况下，自己去摸索怎样使用AWS来搭建环境、学习使用Keras（之前连Keras是啥都不知道）、研究流行的各种模型如ResNet、和编写helper function来处理数据的转移等等。过程虽然相对漫长和痛苦，可是学到的东西是实实在在的。

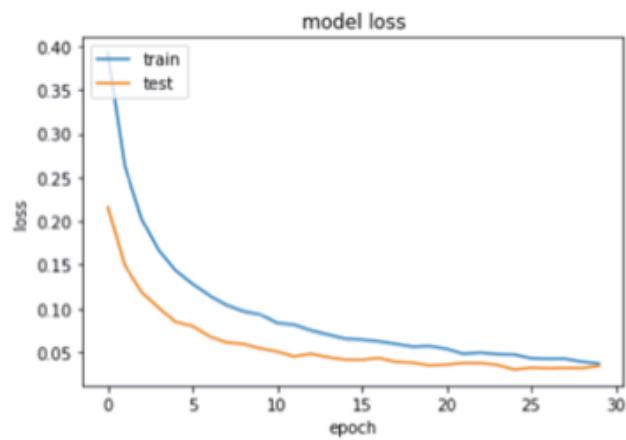
比较诧异的是，模型的构建和训练比想象中的简单多了。本来以为，要进入Kaggle比赛前10%会是一个蛮有难度的事情，例如当时的预想是要用tensorflow来重新搭建一个自己设计的庞大的模型，然后还要经过漫长而重复的各种尝试才能完成。幸运的是事实并不是如此，通过使用Keras和迁移学习（Transfer Learning），很容易就拿到理想的成绩，从中我们也学习到避免重造轮子、学会站在巨人肩膀上的重要性。

每个模型的特征和表现可以参考下图，综合来说，虽然Xception的训练速度比较慢（比较多的epochs），可是达到的成绩是最好的。InceptionResNetV2和ResNet50无论在速度和表现都差不多。而VGG16是最糟糕的模型，为什么呢？因为除了val_loss是最高的，不要忘记了它的参数数量也是最高的，一共有138357544个参数，是其他三模型的参数量的5到6倍之多，具体数值可以参考前文【数据清理】的对比表。所以我们以下结论，VGG16，至少在本项目中，不是一个理想的模型结构，

VGG16 - { val_loss: 0.0544 }

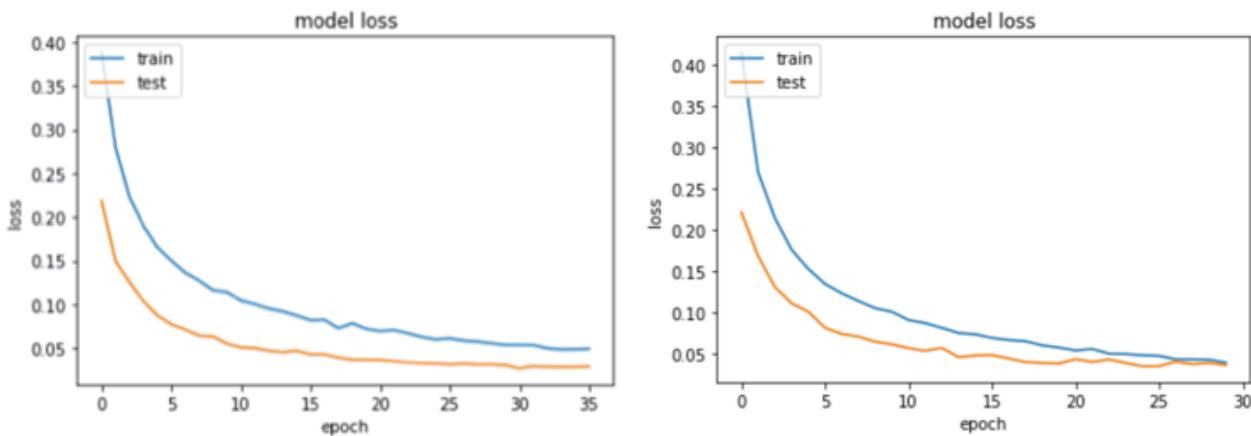


InceptionResNetV2 - { val_loss: 0.0340 }



Xception - { val_loss: 0.0290 }

ResNet50 - { val_loss: 0.0370 }



最后，成绩虽然没有冲到理想中的前15名，可我们使用了一个简单、易懂、和高效的方法（或pipeline）来达到目的，把成绩冲到了前6%，高于项目要求的10%，符合了前文【问题陈述】提及的简单化原则和项目的期望。当然，模型还有可以继续改善空间，可是基于完成比完美重要的原则，本项目就暂时按这个结果来画上句号。因为，机器学习技术的价值是如何应用在真实场景中来帮助人类提升效率和解决真实问题，而不是在同一个数据集上，为了一分几毛的准确率去花大量时间在不断地调整模型。例如，在真实的场景中，需要区别猫猫狗狗的应用应该不多，甚至没有，但我们可以思考如何使用本项目所学到的技术来应用在其他的通用场景，例如区别咖啡豆中的优质豆和瑕疵豆，释放人们的劳动力的同时还能提高优品率，而且咖啡豆的图片采集也相对容易得多，这是个题外话。当然，我们也是可以在不久的未来，针对本项目的最终模型来进行改善，从而达到学术性的目的，也是一件美好的事。以下是模型改进的一些思考，

1. 训练集数据

- 从其它渠道收集更多的猫狗图片，增加训练集的数量。
- 加大数据增强的力度，例如调大参数的range数值。
- 尝试把训练集分成k-fold来训练模型

2. 数据清理

- 改进算法、把更多的“异常图片找出来，让训练数据更加干净。

3. 改进模型结构

- 调整参数，例如把drop rate调大，来减低over fitting。
- 修改top model的结构，例如增加或减少Dense层数。
- 尝试不同的optimizer和相应的参数。
- 增加或减少EarlyStopping callbacks的参数值。

需要说明的是，本项目所用的整体方案基本是通过自己的构思所创建的，出于独立思考原则，所以在整个过程中没有参考任何其他关于猫狗大战的代码资料。其实，网络上有很多类似【如何冲到猫狗大战前1%】的文章，我们以后可以拜读一下，从前人的知识果实中，找到可以改进自己模型的启发。

参考资料

1. Karen Simonyan, A.Z., Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV], 2014.
2. Christian Szegedy, S.I., Vincent Vanhoucke, Alex Alemi, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261v2, 2016.
3. Chollet, F., Xception: Deep Learning with Depthwise Separable Convolutions. arXiv: 1610.02357 [cs.CV], 2016.
4. Kaiming He, X.Z., Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs.CV], 2015.