I. 问题的定义

项目概述

人类对于世界的感知有 80%都来自于视觉，也就是我们的眼睛。眼睛捕捉到来自环境中物体的颜色、大小、形状、材质等信息，与大脑中对物体的认知产生关联，从而识别出环境中物体的类别。对于机器来说，要想对事件或者物体做出决策判断也不能缺少视觉信息。 尤其是近年来，随着 imagenet[5]大量有标签数据集的出现和计算机算力大幅提升，深度学习技术在视觉识别领域得到了迅速的发展，到今天，应用深度学习技术在 imagenet 对图像的识别分类准确率已经超过人类。

利用深度学习对图像进行分类，是在预先提供给机器大量的带有标签的图像的情况下，机器自发的提取图像中的特征并进行学习，进而对没有标签的、陌生的图像进行分类。比较著名的深度神经网络模型有 Alexnet[4]、VGG[1]、ResNet[2]、Inception[3]等模型对图像分类任务有巨大的推进作用，另外还有 tensorflow[12]、keras[11]、caffe 等开发工具和接口出现，提升了深度学习算法的开发和验证效率，使得今天我们能够比较方便的对深度学习在不同的应用场景下进行验证。

本项目的目的是利用预先提供的大量带有标签的猫和狗的图像，基于深度学习技术对没有标签的、未曾见过的猫狗图像进行分类，判断图像中包含的动物是猫还是狗。这也是 kaggle 比赛《Dogs vs. Cats Redux: Kernels Edition》所要实现的任务，因此数据集就来自于 kaggle；该数据集中共包含训练和测试两类数据，训练数据包含 25000 张猫和狗的图像，测试数据包含 12500 张无标签的猫和狗的图像。我们将基于 keras 搭建一个深度神经网络，利用训练数据对深度神经网络模型进行训练、优化，最后用优化过的模型对测试数据进行分类。

问题陈述

在本文中，我们的目标是基于 Kaggle dog vs cat 数据集上解决一个二分类问题。具体问题为利用带有标签的猫或者狗的图片，搭建深度神经网络模型并利用这些图片与标签进行训练得到优化的深度神经网络，最终利用优化的深度神经网络对未曾见过的没有标签的图片做出预测，判断出图片中包含猫还是狗从而达到分类的目的。

在本项目中，我们将利用成熟的深度神经网络模型例如 ResNet50 和其基于 imagenet 训练后的模型和权重，应用迁移学习构建深度神经网络，再通过训练不断调整模型的参数使得模型在猫狗分类中的表现更好。

其中我们利用的数据集是 Kaggle 提供的猫和狗的数据集。最终，我们期望优化过的深度神经网络在测试集上的得分表现可以在 Kaggle 前 10%，具体表现为 Kaggle logloss 比 90%的人要低。

评价指标

我们利用 Logloss 做为损失函数来评估模型的表现，Logloss 能够表现出模型的预测与真实值之间的差异大小；具体来说，当 Logloss 较小时，模型表现能力强，正确预测猫狗图片的能力强；当 Logloss 较大时，模型表现能力差，正确预测猫狗图片的能力弱。其中，Logloss 计算公式为：

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

其中 n 为测试集中的图片数量；当预测的图片的真实值为狗时y_i为 1，真实值为猫时y_i为 0；

$\hat{y}_i$代表的是预测图片为狗的概率。
最终，我们期望优化过的深度神经网络在测试集上的得分表现可以在 Kaggle 前 10%，即 logloss 低于 0.06127。

II. 分析

数据的探索

本次项目中使用的数据集是 Kaggle 比赛中提供的数据集。
这个数据集的训练数据一共包含 25000 张猫和狗的图片，其中包含猫的图片 12500 张，名字中包含‘cat’字段，另外可以看到图像中的猫在图像中所占比例比较大并且背景比较简单。

狗的图片 12500 张，名字中包含‘dog’字段，与猫的图像类似，这些图像中背景比较单一并且狗在图像中所占比例比较大

从训练集的数据可以发现，猫和狗在图像中所占比例比较大，虽然清晰度不一，图片大小不一，但猫和狗充满了图像的大部分区域，因此是个典型的分类问题。另外，训练集中猫和狗的图片数量相当，因此我们不需要对某一类样本进行补全以达到平衡样本的目的。

测试集中包含 12500 张图片，名字中没有包含任何暗视图像类别的字段，以数字命名，图像特点与训练集类似，背景单一且猫狗所占比例比较大。


13.jpg


14.jpg

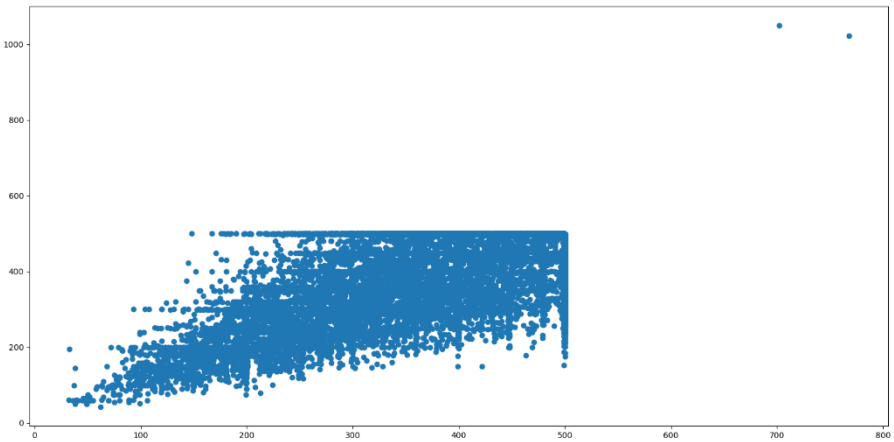

15.jpg


17.jpg

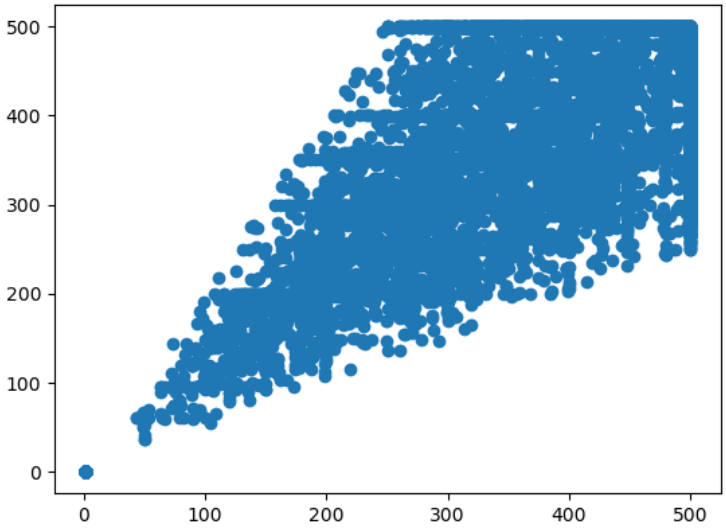

18.jpg


19.jpg

此外，我们对输入数据的图片尺寸分布进行了可视化，如下图。
可以看到训练集中大部分图片的尺寸都不超过（500，600），只有两张图片是离群值分别是 cat.835 和 dog.2317 如下图所示，而这两张并不是 outlier。



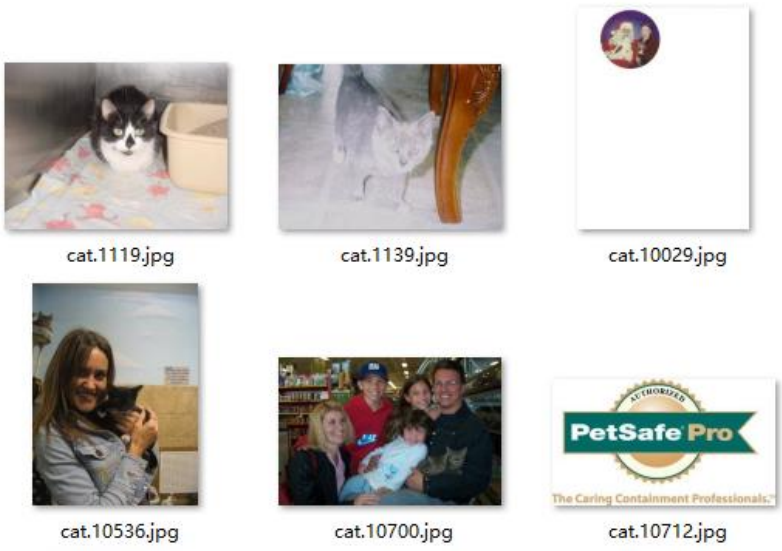
而测试集中的所有图片尺寸都在（500，500）范围内。



在此，我并没有做关于图片颜色分布的可视化，因为在本项目中我认为图片颜色分布并不是暗示图片是否是 outlier 的依据，而我们接下来将会利用著名网络在 imagenet 上的预测来进行 outlier 检测，在这个角度上图片颜色信息也已经被隐形的利用了，因此在此并不单独可视化。

异常值检测

在数据正式输入模型进行训练之前，我们需要确保训练集中没有异常值存在。在本项目中，我们认为图像中不包含猫和狗的图片、或者包含猫狗但猫狗图像不明显的，都认为属于异常值。因为如果把不包含猫狗的图片输入到模型中训练如下图 cat.10029.jpg 和 cat.10712.jpg，会导致模型抽取出一系列跟真实猫狗无关的干扰特征，而影响模型的训练；对于图像中包含猫狗但不清晰的图片，我们也将其归类为异常值，例如下图中 cat.10536.jpg 这张图片，图片中有包含猫，但猫所占比例非常小而人占图片中的大部分区域，模型很可能抽取出的是人的特征而并不是猫的特征，影响模型的训练。

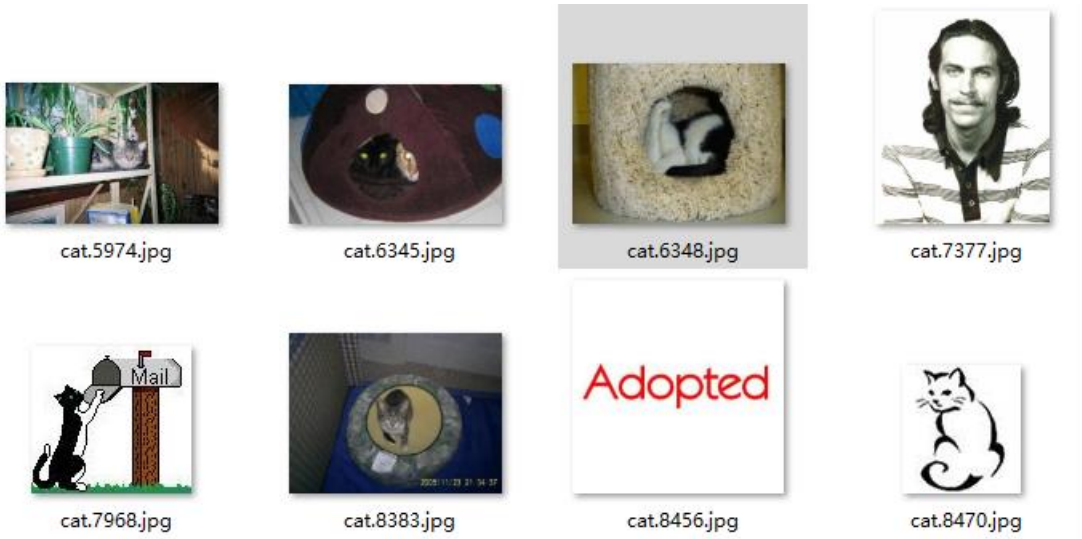


但由人肉眼观察 25000 张图片显然是不现实的，因此我们采用著名的 ResNet50 模型，通过

导入在 imagenet 上训练的权重来判断图片是否是 outlier。因为 imagenet 数据集包含 1000 个类别的图片其中包含猫和狗的图片，模型经过 imagenet 训练后能够提取出猫和狗的特征，因此我们利用训练好的模型对本次 kaggle 数据集的 outlier 进行筛选，最终由人来决定是否是 outlier，是的话将其从训练集移除。

如上图，利用的是模型对一张图片的预测，如果 Top20 类中没有包含猫和狗，模型就认为这张图片是 outlier。可以从图中看出，筛选出的 outlier 并不十分准确，例如 cat.1139.jpg 是一只猫而结果却被归类到 outlier 中。

经过试验，最终我们选择了 Top30 来检测 outlier，ResNet50 Top30 预测出的 outlier 为 83 张，经过人工筛选，确定其中的 63 张为 outlier，其中包含猫的图片 33 张，狗的图片 30 张；最终选择的猫的 outlier 如下所示：



狗的 outlier 如下：

去掉 63 张 outlier 就是最后训练所用的训练集，共 24936 张图片。

算法和技术

最近几年，大量带标签的数据集和优秀卷积神经网络如 VGG[1]、ResNet[2]、Inception[3]等的出现，卷积神经网络在图像领域得到了广泛的应用。本项目利用著名的卷积神经网络，基

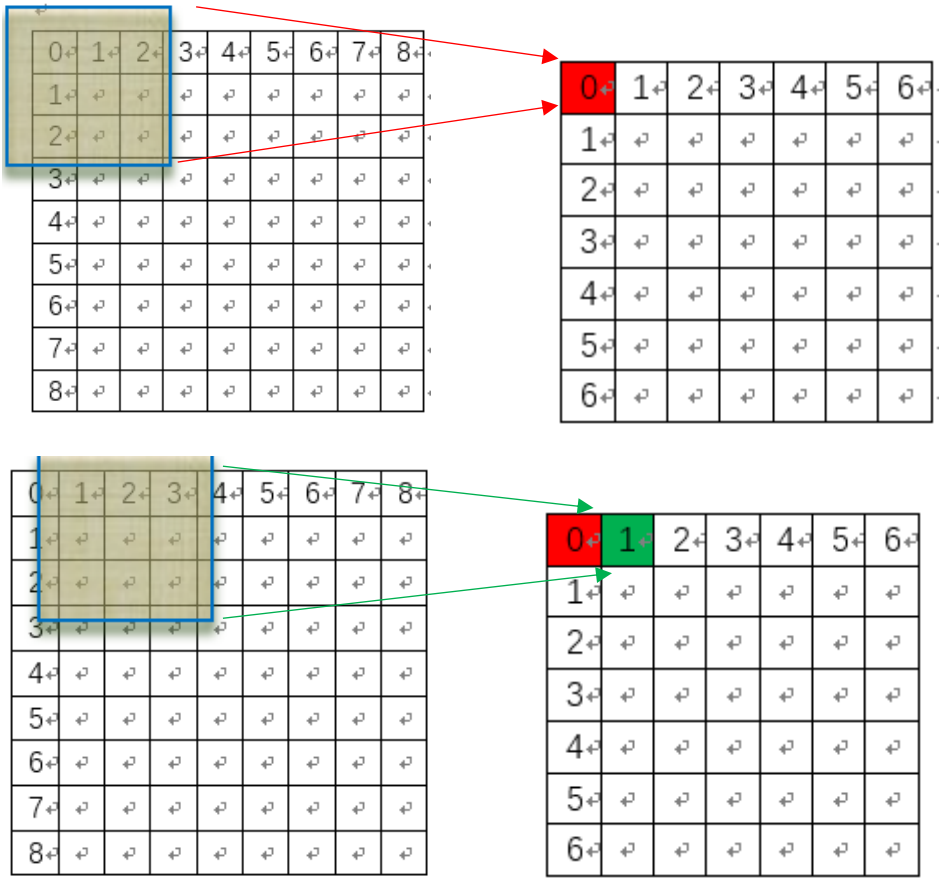
于 Keras 对网络进行迁移学习，使其适应猫狗图像识别的二分类问题，涉及到的算法如下。

卷积神经网络

卷积神经网络由很多层组成，具体包含：卷积层、激活函数、池化层、全连接层等。

卷积层：输入卷积层的图像是 2D 图像，选定一个 filter 做为滑动窗口。如下图，将 filter 从左上开始与图像中的对应部分进行卷积，得到一个输出；然后将 filter 窗口在水平和竖直方向上进行滑动，并且每滑动一次就与图像相对应的部分进行卷积，直到滑动到图像的边缘；滑动窗口的步长称为 stride。

如下图假设输入卷积层的图像为 9*9，filter 大小为 3*3，stride 为 1，经过卷积层，最终提取得到 7*7 的特征；另外，我们在 7*7 的外围 padding，使输出特征的大小与输入一致。



其中，一张图片进行卷积操作时，filter 中的权重不变，一幅图片中不同位置的相同目标，提取到的特征是相同的。权值共享在降低了网络模型复杂度的同时极大程度的减少了运算量，这也是与普通神经网络最根本的区别。

池化层：主要是针对卷积后提取的特征，对图像的局部区域求均值或最大值，用来代表其采样的区域。利用池化层这种下采样的方法，降低了输出参数量，并赋予模型对于轻度形变的容忍能力，提高了模型的泛化性能。

全连接层：两层之间所有的神经元都有连接，也就是 FC 层。

Dropout：训练时，使用 Dropout 随机忽略一部分神经元，不更新这部分神经元的权重值，以避免模型过拟合。

迁移学习

从以前的任务当中去学习知识（knowledge）或经验，并应用于新的任务当中。也就是说，迁移学习的目的是利用一个已经在某个特定领域的数据上优化过的模型，提取数据中的特

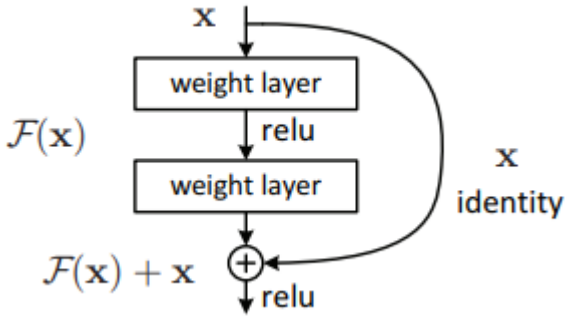
征，将其应用到类似领域中去。现在许多著名的模型如 ResNet50 基于 Imagenet 数据集都有很不错的表现，而猫狗二分类问题是 Imagenet 中的子类，因此利用其提取特征相似性，我们可以将其应用到猫狗分类当中。

迁移学习通常有两种应用方法：

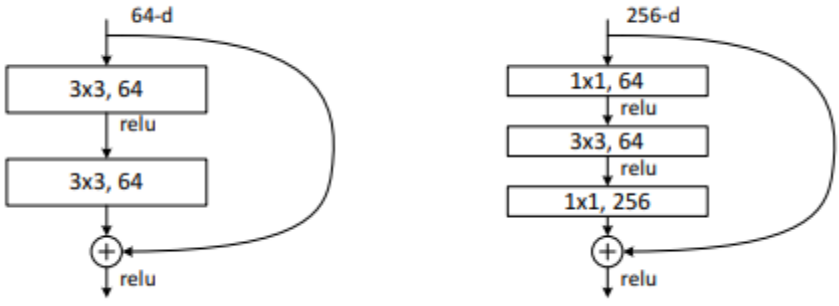
1. 利用迁移学习，基于预训练的网络提取特征；
2. Finetune：去掉预训练的网络 top 的全连接层，在之后另行设计分类器；利用猫狗数据集对网络进行训练时，只对最后几层的权值进行更新。

ResNet50

本项目利用 ResNet50 进行迁移学习，与以往模型的区别主要是提出了残差单元[2]，来解决随着深度增加梯度消失的问题。当残差为 0 时，相当于仅仅做了恒等映射，网络性能不会没有提升但也不会下降，实际上残差不会为 0，网络仍然能从输入的特征中学习到新的特征，从而拥有更好的性能。残差单元的结构如下图，称为 short connection。



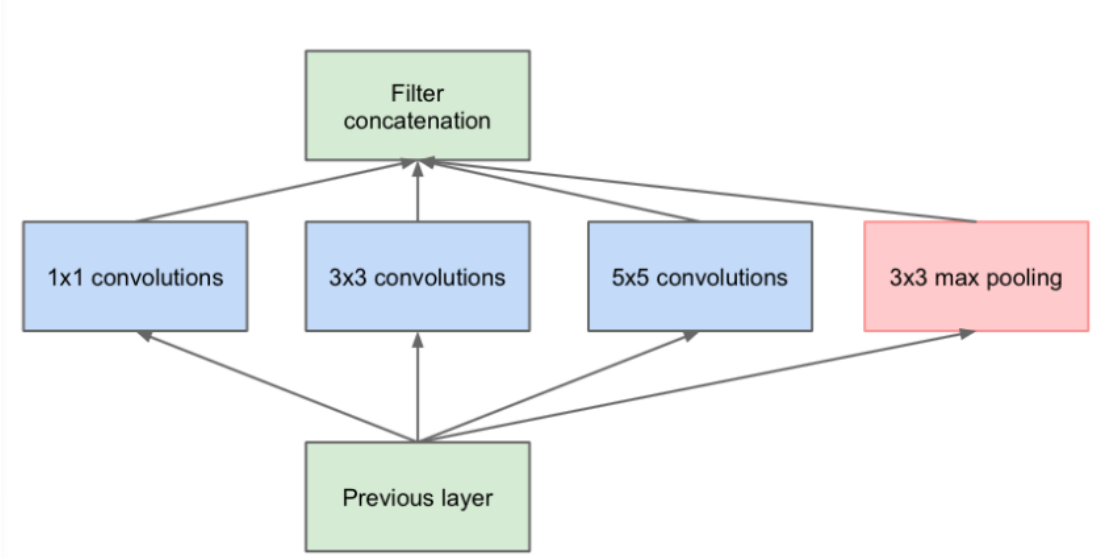
实际上，ResNet 使用两种残差单元，如下图。左图对应的是浅层网络，而右图对应的是深层网络，ResNet50/101/152 就是使用右边的残差单元。对于 short connection，当输入和输出维度一致时，可以直接将输入加到输出上。但是当维度不一致时（对应的是维度增加一倍），这就不能直接相加。有两种策略：（1）采用 zero-padding 增加维度，此时一般要先做一个 downsamp，可以采用 strde=2 的 pooling，这样不会增加参数；（2）采用新的映射（projection shortcut），一般采用 1x1 的卷积，这样会增加参数，也会增加计算量。短路连接除了直接使用恒等映射，当然都可以采用 projection shortcut。



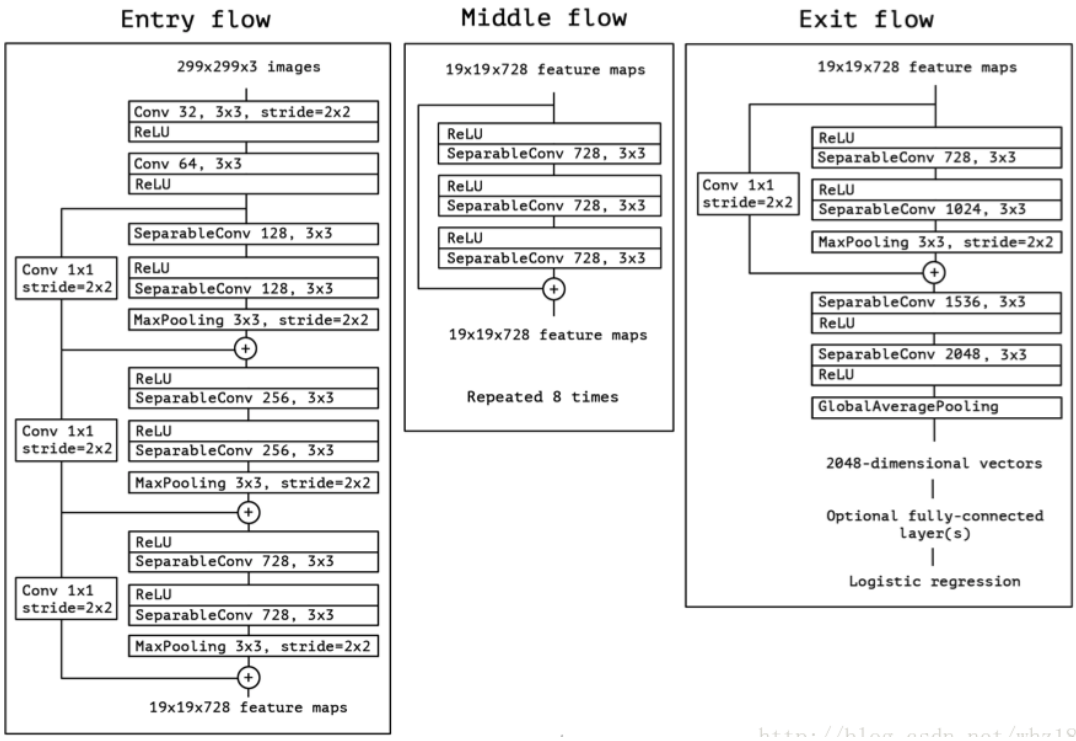
Inception 和 xception

通常来说卷积神经网络获得更高性能可以通过设计更深的网络（层数更多）或者更宽的网络（神经元更多）来实现。从 Alexnet、VGG 和 Resnet 看到，网络层数越来越多，表现也越来越好。而 Inception 的主要思想是利用更宽的网络来提升网络的性能，与之前的卷积神经网络最大区别是引入了 Inception 单元[6]，将 1*1，3*3，5*5 的 conv 和 3*3 的 pooling 组合在

一起。因为不同大小的卷积核代表在原图上不同大小的感受野，最后的拼接就是将不同尺度特征和 pooling 的融合一同输入网络中，进而由网络通过学习自动决定哪些 conv 是更有用的、是否需要 pooling 等，从而得到更优的网络结构。Inception 单元如下图：



后来，Inception 继续进化在 conv 和 pooling 之后加入了 1x1 卷积核来进行降维，大大减少了计算量；加入 BN（Batch Normalization）层[7]，与 ResNet 相结合，对 inception 单元中的卷积核进行不同的变换等产生了 Inception V1、InceptionV2、InceptionV3、InceptionV4 等网络，一直到 2017 年 google 基于 depthwise separable convolution 结构重新设计了 Inception 模块，即为 Xception[9]，进一步在减少模型参数的前提下增加了模型的表现能力。
depthwise separable convolution 就是将传统的卷积操作分成两步，假设原来是 3*3 的卷积，那么 depthwise separable convolution 就是先用 M 个 3*3 卷积核一对一卷积输入的 M 个 feature map，不求和，生成 M 个结果；然后用 N 个 1*1 的卷积核正常卷积前面生成的 M 个结果，求和，最后生成 N 个结果。图中 sparsableConv 就是 depthwise separable convolution。另外，每个小块的连接采用的是 residule connection（图中的加号），而不是原 Inception 中的简单拼接。



基准模型

我们利用 Logloss 做为损失函数来评估模型的表现，当 Logloss 较小时，模型表现能力强，正确预测猫狗图片的能力强；当 Logloss 较大时，模型表现能力差，正确预测猫狗图片的能力弱。其中，Logloss 计算公式为：

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

其中 n 为测试集中的图片数量；当预测的图片的真实值为狗时y_i为 1，真实值为猫时y_i为 0；$\hat{y_i}$代表的是预测图片为狗的概率。

最终，我们期望优化过的深度神经网络在测试集上的得分表现可以在 Kaggle 前 10%，即 logloss 低于 0.06127。

III. 方法

数据预处理

由于 imagenet 的存在，并且许多著名的网络模型都已经针对 imagenet 做过模型的调优和训练，因此可以很好的对 imagenet 上的物体种类进行分类，如下表。恰好 imagenet 包含了几种猫和上百个不同品种的狗的分类，因此我们可以方便的利用已经存在的著名模型来对猫狗中的图片进行 outlier 分析。

Documentation for individual models

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

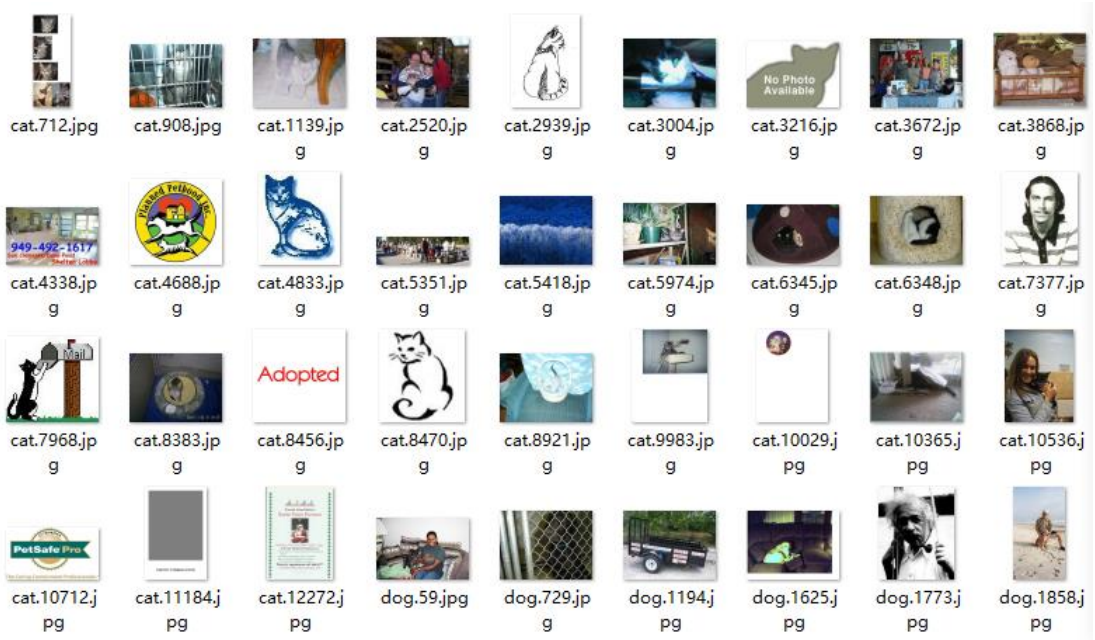
The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

从表中可以看到，从上到下各种网络从 Xception 到最后的 DenseNet201 对 imagenet 中的图片进行预测时，Top-1 Accuracy 对概率最大的一类图片进行预测的准确率依次降低从 79%到 77%，而如果预测时给出前 5 个可能的分类，5 个可能的分类中包含正确的图片分类的概率即为 Top-5 Accuray，准确率都在 93%以上，这已经是一个比较高的概率了。

由于我们此处是为了找到数据集中的 outlier，希望 Top-N accuracy 越高，说明我们找 outlier 的准确率就越高；根据 Top-N accuracy 的原理，我们有理由认为 Top-30 比 Top-5 的准确率要高很多，因为 Top-30 包含了更多的可能性。

我们利用 ResNet50 在 imagenet 上的预训练模型对 top30 类物体进行预测；由于 imagenet 数据集中包含 118 个不同狗的品种和 7 个猫的品种，因此当预测 top30 类别中不包含 118+7=125 个猫狗的种类时，模型认为这张图片是 outlier。

在实验中，输出结果中有 83 张图片 top30 类别中即不包含狗也不包含猫。经过人工检查，剔除掉预测错误的图片，最终有 63 张图片被认为是真正的 outlier 如下：



可以看到 outlier 是模糊的图片、不是真实的猫狗图片、或者非猫非狗的图片，将这些删除就是我们最终使用的训练数据集。

数据集分割

在剔除数据集中的 outlier 之后，我们从中随机挑选出 2500 张猫的图片和 2500 张狗的图片做为验证集，将其放如/valid 文件夹中。

```
for root, dirs, files in os.walk(scr):
    for d in dirs:
        for i in range(0, 2500):
            rand = random.randint(1, 12470)
            tmp = rand
            fileNm = d + '.' + str(rand) + '.jpg'

            scr_path = os.path.join(scr, d, fileNm)
            #print(scr_path)
            dst_path = os.path.join(dst, d)
            #print(dst_path)

            while not os.path.exists(scr_path):
                tmp = tmp - 1
                fileNm = d + '.' + str(tmp) + '.jpg'
                scr_path = os.path.join(scr, d, fileNm)

            shutil.copy(scr_path, dst_path)
            os.remove(scr_path)

            print(fileNm)
```

读入数据时进行图像增强

利用 Keras 中的 ImageDataGenerator 对输入的图像进行预处理。在本项目中，对图片的预处理包括：

我们选用的预训练模型 ResNet50 自带的预处理函数 preprocess_input 对对图像 BGR 三通道值减去 imagenet 均值、将图像大小统一归一化成大小为 (224, 224)以适应模型的输入，

对图像在[0,40]度中随机进行旋转、对高宽分别进行平移、放大图像、随机对图片进行反转[10]等处理，以获得更加丰富的训练数据。
而如果采用 Inception V3 的话，需要将 RGB 三通道值缩放至【-1，1】之间，并且将图像大小 resize 成（299，299）以适应模型的输入，再利用 keras 中的图像增强对图像进行旋转、平移、放大等处理，获得更加多样化的训练数据。

```
# define the training set and validation set
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(DATASET_PATH + '/train',
                                                  target_size=IMAGE_SIZE,
                                                  interpolation='bicubic',
                                                  class_mode='categorical',
                                                  shuffle=True,
                                                  batch_size=BATCH_SIZE)
```

另外，就像评审中给出的建议一样，过于激进的数据增强可能会导致训练集和验证集分布不一致最终造成过拟合问题，因此，在利用 xcetion 模型时，我们会做关于是否进行数据增强的对比试验，即分别进行数据增强和不增强，最后利用输出的模型对测试集进行预测，进而比较此次数据增强是否会导致模型的过拟合。

模型构建

我们利用 keras 载入在 imagenet 上预训练的模型，去掉 top layer 之后连接上 dropout layer 和输出层，得到我们训练的模型。

训练过程

卷积神经网络在低层次结构中提取的是通用的特征，例如边缘、形状等信息，与猫狗中的低层次结构是类似的，因此我们在训练中只需要训练高层次 layer 就可以将预训练的模型使用在猫狗分类的任务中。

在本项目中，我们将训练步骤分为两步：

1. 首先保持原先的 ResNet50 中的参数不变，训练模型最新增加的三层，将新增加层的参数训练调整到最优状态。
2. 再放开 ResNet50 的最后两个 block，再加上新增加的三层同时进行训练，得到 fine-tuned 的模型。

在整个训练过程，我们分别调用了 callbacks 的 ModelCheckpoint、ReduceLROnPlateau 和 EarlyStopping 来对训练过程进行记录和观察，其中[11]

ModelCheckpoint：我们定义当本次训练的 epoch 后，模型在验证集上的准确率最高时，保存本次 epoch 后的模型

ReduceLROnPlateau：当评价指标不在提升时，降低学习率

EarlyStopping：当 val_loss 经过 6 个训练轮数不在优化时停止训练（即 patience 设置为 6）。

在整个训练过程中，我们先后采用了 3 个预训练模型，他们分别是 ResNet50[2]、InceptionV3[8]和 Xception[9]。

训练 ResNet50 得到初步的结果：
首先训练顶端的 Flatten、dropout 和输出分类层，经过 10 个 epochs 训练停止，历时 2609s；但仔细观察输出结果，其实在经过 4 个 epochs 的训练，模型在验证集上的准确率达到最高点。因此我们选择 epochs4 输出的网络参数做为下一轮训练的起始值。

```
Epoch 1/50
623/623 [==============================] - 261s 419ms/step - loss: 0.1000 - acc: 0.9624 - val_loss: 0.0425 - val_acc: 0.9849
Epoch 2/50
623/623 [==============================] - 257s 413ms/step - loss: 0.0933 - acc: 0.9655 - val_loss: 0.0313 - val_acc: 0.9871
Epoch 3/50
623/623 [==============================] - 258s 413ms/step - loss: 0.0845 - acc: 0.9685 - val_loss: 0.0349 - val_acc: 0.9869
Epoch 4/50
623/623 [==============================] - 257s 412ms/step - loss: 0.0830 - acc: 0.9678 - val_loss: 0.0274 - val_acc: 0.9895
Epoch 5/50
623/623 [==============================] - 258s 414ms/step - loss: 0.0765 - acc: 0.9705 - val_loss: 0.0389 - val_acc: 0.9845
Epoch 6/50
623/623 [==============================] - 258s 415ms/step - loss: 0.0780 - acc: 0.9697 - val_loss: 0.0424 - val_acc: 0.9835
Epoch 7/50
623/623 [==============================] - 259s 415ms/step - loss: 0.0705 - acc: 0.9726 - val_loss: 0.0364 - val_acc: 0.9863
Epoch 8/50
623/623 [==============================] - 256s 411ms/step - loss: 0.0696 - acc: 0.9737 - val_loss: 0.0362 - val_acc: 0.9865
Epoch 9/50
623/623 [==============================] - 257s 412ms/step - loss: 0.0696 - acc: 0.9736 - val_loss: 0.0367 - val_acc: 0.9855
Epoch 10/50
623/623 [==============================] - 257s 413ms/step - loss: 0.0667 - acc: 0.9744 - val_loss: 0.0356 - val_acc: 0.9871
time consumed: 2609.5891664028168
```

放开 ResNet50 顶部的两层继续训练：经过 10 个 epochs 训练停止，历时 2615s；可以看到在 epochs4 之后在验证集上的准确率达到最高点 0.9901.

```
Epoch 1/50
623/623 [==============================] - 264s 424ms/step - loss: 0.0871 - acc: 0.9700 - val_loss: 0.0269 - val_acc: 0.9897
Epoch 2/50
623/623 [==============================] - 259s 416ms/step - loss: 0.0782 - acc: 0.9708 - val_loss: 0.0363 - val_acc: 0.9881
Epoch 3/50
623/623 [==============================] - 260s 417ms/step - loss: 0.0795 - acc: 0.9705 - val_loss: 0.0293 - val_acc: 0.9883
Epoch 4/50
623/623 [==============================] - 259s 415ms/step - loss: 0.0730 - acc: 0.9715 - val_loss: 0.0269 - val_acc: 0.9901
Epoch 5/50
623/623 [==============================] - 258s 415ms/step - loss: 0.0679 - acc: 0.9746 - val_loss: 0.0291 - val_acc: 0.9883
Epoch 6/50
623/623 [==============================] - 257s 413ms/step - loss: 0.0676 - acc: 0.9742 - val_loss: 0.0297 - val_acc: 0.9881
Epoch 7/50
623/623 [==============================] - 259s 416ms/step - loss: 0.0681 - acc: 0.9732 - val_loss: 0.0292 - val_acc: 0.9879
Epoch 8/50
623/623 [==============================] - 258s 414ms/step - loss: 0.0642 - acc: 0.9746 - val_loss: 0.0283 - val_acc: 0.9883
Epoch 9/50
623/623 [==============================] - 258s 415ms/step - loss: 0.0649 - acc: 0.9749 - val_loss: 0.0288 - val_acc: 0.9883
Epoch 10/50
623/623 [==============================] - 260s 417ms/step - loss: 0.0673 - acc: 0.9754 - val_loss: 0.0312 - val_acc: 0.9877
time consumed: 2615.46914935112
```

将模型保存下来，对测试集中的图片进行分类，提交到 kaggle 上得分为 0.08432。
可以看到这个结果并没有达到我们预先设定的基准，因此接下来我们有两个 option 来对结果进行改进：

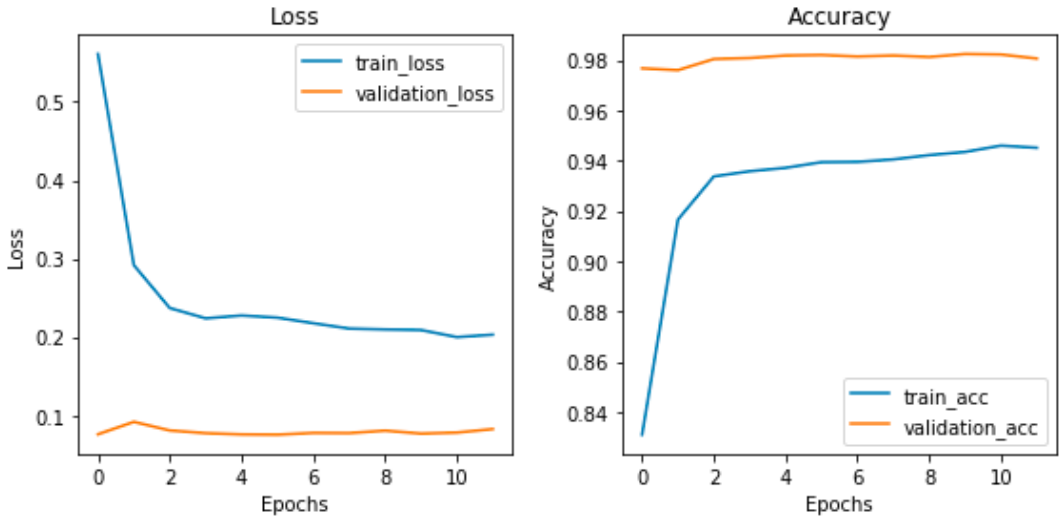
1. 调整超参数例如 learning rate，调整新添加的 layer 数目以及参数
2. 利用更强大的预训练模型例如 inception、xception 等作为基础模型

首先来看 option 1：
在这个步骤中，我们不仅继续使用比较小的 learning rate，还添加了 ReduceLROnPlateau 回调函数，当验证集上的准确率不再下降时，将 learning rate 的值降低到原来的 1/10. 经过训练，我们发现这种方法对提升模型的表现不明显，可能的原因是 learning rate 原本就是比较小的值 1e-5，不存在 step 太大以至于越过最优值，因此在这种情况下调整 learning rate 收效甚微。
除此之外，还试验了调整 dropout 的概率、将 flatten、dropout 层替换为 GlobalAveragePooling，对性能的提升都不明显不能达到要求，因此采用 option2 对网络进行修改。

Option 2:

预训练模型采用 InceptionV3[8]：首先训练顶端的 Flatten、dropout 和输出分类层.经过 10 个 epochs 训练停止，其中在经过 4 个 epochs 后验证集上的准确率已经达到最高点。

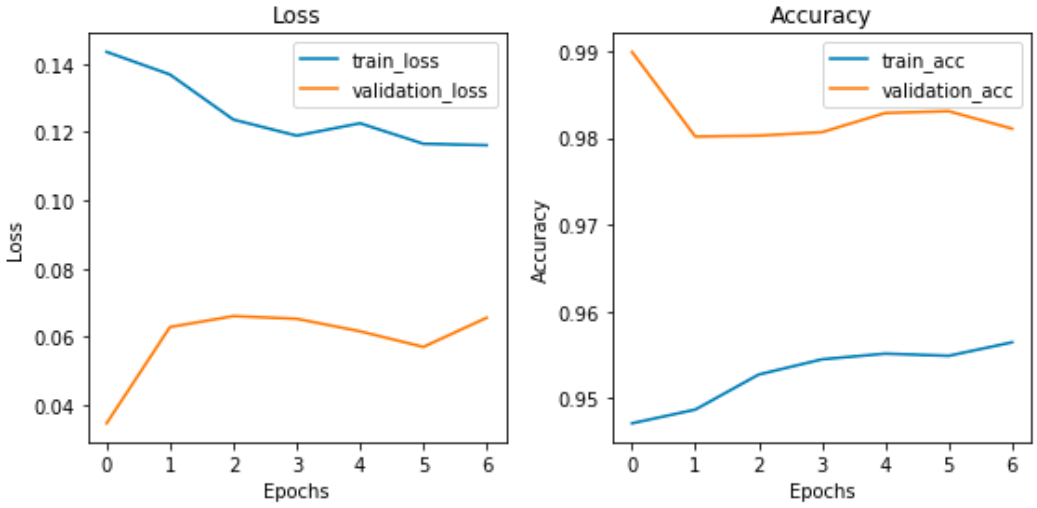
```
Epoch 1/50
623/623 [==============================] - 440s 707ms/step - loss: 0.3017 - acc: 0.8702 - val_loss: 0.0791 - val_acc: 0.9702
Epoch 2/50
623/623 [==============================] - 428s 687ms/step - loss: 0.1757 - acc: 0.9317 - val_loss: 0.0514 - val_acc: 0.9823
Epoch 3/50
623/623 [==============================] - 427s 685ms/step - loss: 0.1623 - acc: 0.9365 - val_loss: 0.0640 - val_acc: 0.9779
Epoch 4/50
623/623 [==============================] - 428s 686ms/step - loss: 0.1504 - acc: 0.9426 - val_loss: 0.0514 - val_acc: 0.9833
Epoch 5/50
623/623 [==============================] - 428s 686ms/step - loss: 0.1376 - acc: 0.9472 - val_loss: 0.0539 - val_acc: 0.9825
Epoch 6/50
623/623 [==============================] - 429s 688ms/step - loss: 0.1386 - acc: 0.9486 - val_loss: 0.0567 - val_acc: 0.9813
Epoch 7/50
623/623 [==============================] - 428s 688ms/step - loss: 0.1300 - acc: 0.9513 - val_loss: 0.0584 - val_acc: 0.9811
Epoch 8/50
623/623 [==============================] - 429s 688ms/step - loss: 0.1324 - acc: 0.9499 - val_loss: 0.0536 - val_acc: 0.9823
Epoch 9/50
623/623 [==============================] - 430s 690ms/step - loss: 0.1314 - acc: 0.9488 - val_loss: 0.0601 - val_acc: 0.9805
Epoch 10/50
623/623 [==============================] - 429s 688ms/step - loss: 0.1330 - acc: 0.9497 - val_loss: 0.0589 - val_acc: 0.9811
time consumed: 4327.041462182999
```



再放开 InceptionV3 的最上面两层参数进行训练：经过 7 个 epochs 训练停止，其中在经过 1 个 epochs 后验证集上的准确率已经达到最高点。

可以看到 InceptionV3 由于层数比 ResNet50 多，训练 10 个 epochs 的时间从 2600s 左右上升到 4300s。

```
Epoch 1/50
623/623 [==============================] - 438s 703ms/step - loss: 0.1439 - acc: 0.9472 - val_loss: 0.0347 - val_acc: 0.9899
Epoch 2/50
623/623 [==============================] - 431s 692ms/step - loss: 0.1369 - acc: 0.9487 - val_loss: 0.0629 - val_acc: 0.9802
Epoch 3/50
623/623 [==============================] - 430s 690ms/step - loss: 0.1237 - acc: 0.9528 - val_loss: 0.0661 - val_acc: 0.9803
Epoch 4/50
623/623 [==============================] - 430s 690ms/step - loss: 0.1194 - acc: 0.9546 - val_loss: 0.0653 - val_acc: 0.9807
Epoch 5/50
623/623 [==============================] - 430s 690ms/step - loss: 0.1225 - acc: 0.9552 - val_loss: 0.0616 - val_acc: 0.9829
Epoch 6/50
623/623 [==============================] - 430s 691ms/step - loss: 0.1171 - acc: 0.9550 - val_loss: 0.0571 - val_acc: 0.9831
Epoch 7/50
623/623 [==============================] - 431s 692ms/step - loss: 0.1162 - acc: 0.9565 - val_loss: 0.0656 - val_acc: 0.9811
time consumed: 3038.370771884918
```

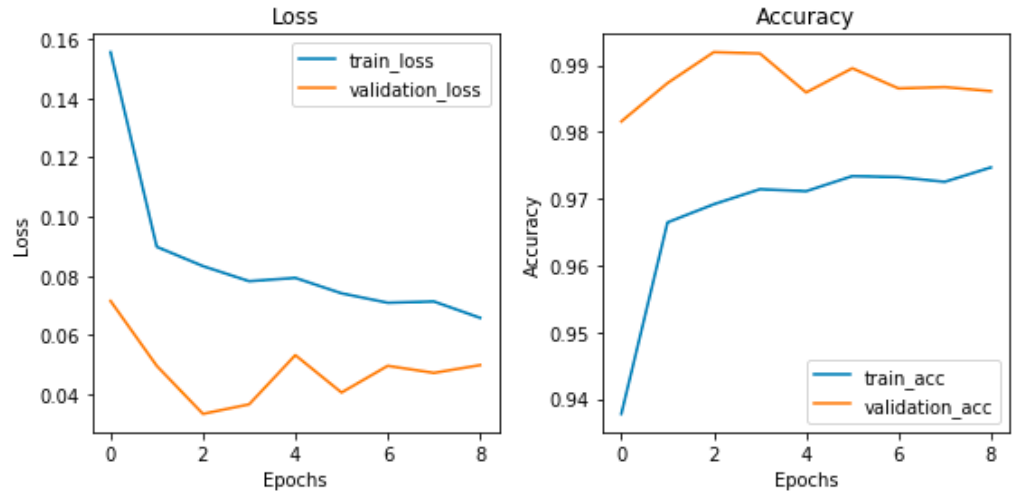


经过 InceptionV3, 我们将预测结果提交到 kaggle 上发现, 得分比 ResNet 有所提高到 0.0709, 但还不足以达到要求。因此我们将预训练模型设置为 xception。

预训练模型采用 xception[9]:

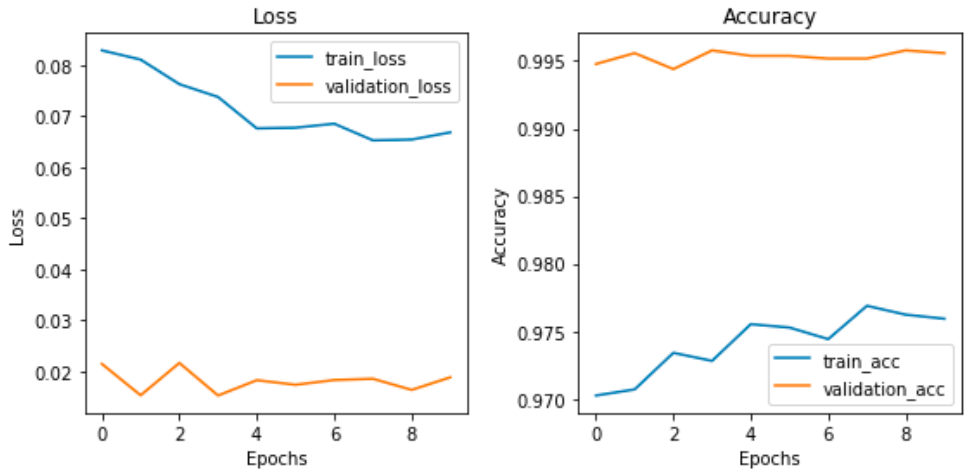
首先训练顶端的 Flatten、dropout 和输出分类层.经过 9 个 epochs 训练停止，其中在经过 3 个 epochs 后验证集上的准确率已经达到最高点。

```
Epoch 1/50
623/623 [==============================] - 461s 740ms/step - loss: 0.1555 - acc: 0.9378 - val_loss: 0.0716 - val_acc: 0.9816
Epoch 2/50
623/623 [==============================] - 452s 726ms/step - loss: 0.0905 - acc: 0.9665 - val_loss: 0.0497 - val_acc: 0.9873
Epoch 3/50
623/623 [==============================] - 453s 727ms/step - loss: 0.0835 - acc: 0.9692 - val_loss: 0.0335 - val_acc: 0.9919
Epoch 4/50
623/623 [==============================] - 453s 727ms/step - loss: 0.0791 - acc: 0.9715 - val_loss: 0.0367 - val_acc: 0.9917
Epoch 5/50
623/623 [==============================] - 453s 727ms/step - loss: 0.0800 - acc: 0.9712 - val_loss: 0.0533 - val_acc: 0.9859
Epoch 6/50
623/623 [==============================] - 453s 728ms/step - loss: 0.0753 - acc: 0.9719 - val_loss: 0.0407 - val_acc: 0.9895
Epoch 7/50
623/623 [==============================] - 454s 728ms/step - loss: 0.0719 - acc: 0.9733 - val_loss: 0.0497 - val_acc: 0.9865
Epoch 8/50
623/623 [==============================] - 452s 726ms/step - loss: 0.0728 - acc: 0.9710 - val_loss: 0.0473 - val_acc: 0.9867
Epoch 9/50
623/623 [==============================] - 451s 724ms/step - loss: 0.0658 - acc: 0.9747 - val_loss: 0.0499 - val_acc: 0.9861
time consumed: 4155.016465425491
```



再放开 xception 的最上面两层参数进行训练：经过 10 个 epochs 训练停止，其中在经过 1 个 epochs 后验证集上的准确率已经达到最高点 0.9958。

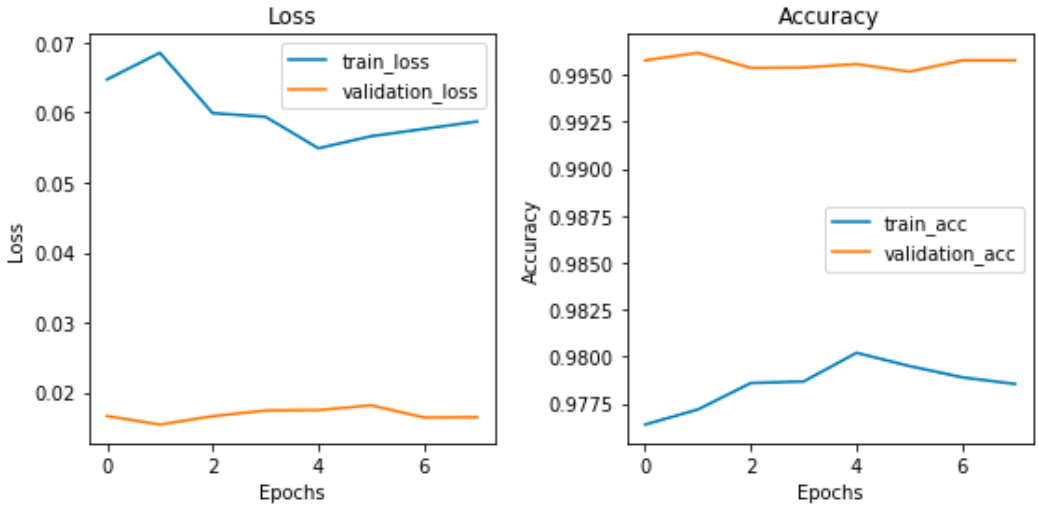
```
Epoch 1/50
623/623 [==============================] - 525s 842ms/step - loss: 0.0830 - acc: 0.9703 - val_loss: 0.0214 - val_acc: 0.9948
Epoch 2/50
623/623 [==============================] - 490s 787ms/step - loss: 0.0844 - acc: 0.9692 - val_loss: 0.0153 - val_acc: 0.9956
Epoch 3/50
623/623 [==============================] - 483s 776ms/step - loss: 0.0762 - acc: 0.9734 - val_loss: 0.0216 - val_acc: 0.9944
Epoch 4/50
623/623 [==============================] - 454s 729ms/step - loss: 0.0747 - acc: 0.9713 - val_loss: 0.0152 - val_acc: 0.9958
Epoch 5/50
623/623 [==============================] - 454s 729ms/step - loss: 0.0677 - acc: 0.9756 - val_loss: 0.0182 - val_acc: 0.9954
Epoch 6/50
623/623 [==============================] - 453s 727ms/step - loss: 0.0680 - acc: 0.9753 - val_loss: 0.0174 - val_acc: 0.9954
Epoch 7/50
623/623 [==============================] - 458s 734ms/step - loss: 0.0684 - acc: 0.9745 - val_loss: 0.0183 - val_acc: 0.9952
Epoch 8/50
623/623 [==============================] - 455s 730ms/step - loss: 0.0673 - acc: 0.9754 - val_loss: 0.0185 - val_acc: 0.9952
Epoch 9/50
623/623 [==============================] - 454s 728ms/step - loss: 0.0653 - acc: 0.9763 - val_loss: 0.0163 - val_acc: 0.9958
Epoch 10/50
623/623 [==============================] - 454s 729ms/step - loss: 0.0671 - acc: 0.9760 - val_loss: 0.0188 - val_acc: 0.9956
time consumed: 4734.597938299179
```



在此之后，我们试着把 xception 的最上面 4 层放开进行训练，试图比较训练更多层是否会对最终的结果产生比较大的影响。

经过训练，我们看到 8 个 epochs 训练停止，其中只经过 2 个 epochs 后验证集上的准确率已经达到最高点 0.9962，这比上一次训练的 0.9958 高出 0.0004，考虑到我们的验证集有 5000 张图片，这意味这经过最后一轮的训练，模型多预测正确了 2 张图片，但并不能说明模型的性能有提升。因此我们认为最后一轮训练不是必须的。

```
Epoch 1/50
623/623 [==============================] - 462s 742ms/step - loss: 0.0665 - acc: 0.9749 - val_loss: 0.0166 - val_acc: 0.9958
Epoch 2/50
623/623 [==============================] - 455s 730ms/step - loss: 0.0687 - acc: 0.9772 - val_loss: 0.0154 - val_acc: 0.9962
Epoch 3/50
623/623 [==============================] - 455s 730ms/step - loss: 0.0599 - acc: 0.9786 - val_loss: 0.0166 - val_acc: 0.9954
Epoch 4/50
623/623 [==============================] - 455s 730ms/step - loss: 0.0594 - acc: 0.9787 - val_loss: 0.0174 - val_acc: 0.9954
Epoch 5/50
623/623 [==============================] - 460s 738ms/step - loss: 0.0559 - acc: 0.9787 - val_loss: 0.0175 - val_acc: 0.9956
Epoch 6/50
623/623 [==============================] - 469s 753ms/step - loss: 0.0593 - acc: 0.9780 - val_loss: 0.0182 - val_acc: 0.9952
Epoch 7/50
623/623 [==============================] - 469s 752ms/step - loss: 0.0578 - acc: 0.9789 - val_loss: 0.0164 - val_acc: 0.9958
Epoch 8/50
623/623 [==============================] - 469s 752ms/step - loss: 0.0611 - acc: 0.9770 - val_loss: 0.0165 - val_acc: 0.9958
time consumed: 3747.995441675186
```



最终我们选择了三次训练的模型来对测试集的图片进行预测，在 kaggle 上的得分为 0.05468，达到最初的要求。

Submission and Description	Public Score	Use for Final Score
submission.csv 3 minutes ago by jhwang submit1102_xception	0.05468	☐
submission.csv a day ago by jhwang submit 1101_inception_updated	0.07090	☐
submission.csv 3 days ago by jhwang submit_1030_finetuned_more_epochs	0.08432	☐
submission.csv 3 days ago by jhwang 1030_best_top	0.08603	☐
submission.csv 3 days ago by jhwang submit_1030_remove_outlier	0.08756	☐

除此之外，针对过分的数据增强是否会导致训练集与测试集数据分布偏差的问题，我们对 xception 的训练过程进行了对比测试。也就是说，在将训练数据注入模型训练时，不进行数据增强（只进行简单的翻转，而不进行平移、旋转等操作），将训练好的模型对测试集中的图片进行预测，最后利用 kaggle 得分来判断是否存在数据增强导致训练数据与测试数据分布偏差的问题。

首先对训练过程进行观察：

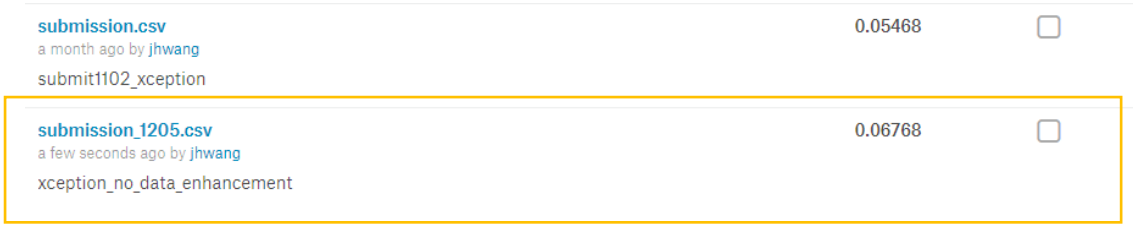
首先训练顶端的 Flatten、dropout 和输出分类层.经过 8 个 epochs 训练停止，其中在经过 2 个 epochs 后验证集上的准确率已经达到最高点 0.9867。

```
Epoch 1/50
623/623 [==============================] - 188s 301ms/step - loss: 0.1045 - acc: 0.9626 - val_loss: 0.0815 - val_acc: 0.9756
Epoch 2/50
623/623 [==============================] - 160s 257ms/step - loss: 0.0475 - acc: 0.9839 - val_loss: 0.0492 - val_acc: 0.9867
Epoch 3/50
623/623 [==============================] - 160s 256ms/step - loss: 0.0431 - acc: 0.9843 - val_loss: 0.0651 - val_acc: 0.9807
Epoch 4/50
623/623 [==============================] - 160s 256ms/step - loss: 0.0376 - acc: 0.9860 - val_loss: 0.0707 - val_acc: 0.9791
Epoch 5/50
623/623 [==============================] - 160s 256ms/step - loss: 0.0327 - acc: 0.9898 - val_loss: 0.0627 - val_acc: 0.9819
Epoch 6/50
623/623 [==============================] - 160s 256ms/step - loss: 0.0294 - acc: 0.9904 - val_loss: 0.0565 - val_acc: 0.9829
Epoch 7/50
623/623 [==============================] - 160s 257ms/step - loss: 0.0309 - acc: 0.9899 - val_loss: 0.0606 - val_acc: 0.9821
Epoch 8/50
623/623 [==============================] - 160s 257ms/step - loss: 0.0308 - acc: 0.9903 - val_loss: 0.0564 - val_acc: 0.9831
time consumed: 1318.531171321869
```

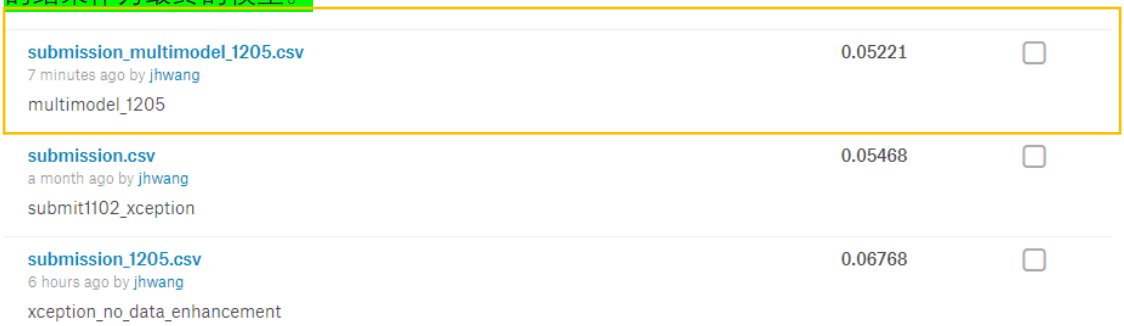
再放开 xception 的最上面两层参数进行训练：经过 5 个 epochs 训练停止，其中在经过 2 个 epochs 后验证集上的准确率已经达到最高点 0.9948。

```
Epoch 1/50
623/623 [==============================] - 161s 259ms/step - loss: 0.0310 - acc: 0.9898 - val_loss: 0.0261 - val_acc: 0.9932
Epoch 2/50
623/623 [==============================] - 161s 258ms/step - loss: 0.0282 - acc: 0.9908 - val_loss: 0.0224 - val_acc: 0.9948
Epoch 3/50
623/623 [==============================] - 161s 259ms/step - loss: 0.0269 - acc: 0.9915 - val_loss: 0.0340 - val_acc: 0.9911
Epoch 4/50
623/623 [==============================] - 162s 259ms/step - loss: 0.0228 - acc: 0.9924 - val_loss: 0.0261 - val_acc: 0.9940
Epoch 5/50
623/623 [==============================] - 161s 259ms/step - loss: 0.0219 - acc: 0.9920 - val_loss: 0.0257 - val_acc: 0.9936
time consumed: 816.0497772693634
```

最后我们将该模型输入测试集中对测试集中的图片进行预测，kaggle 得分为 0.06768；相比较用经过数据增强的数据集，经过相同的训练步骤，对 xception 网络进行训练得到的结果为 0.05468；可以得出结论数据增强确实能够提高模型的表现，也没有导致训练数据与测试数据分布不一样的问题，因此本项目中的数据增强是有效的。



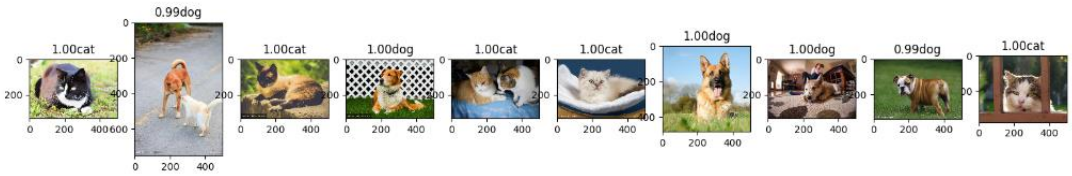
最后基于三个训练好的模型（Resnet50、Inception V3、Xception）进行融合，我们将三个模型预测的结果分别按照 0.2、0.3、0.5 的权重进行相加，得到对测试集中图片的预测结果，提交 kaggle 得分 0.05221，可以看到比三个单独的模型表型都要好，我们将三个模型融合的结果作为最终的模型。



IV. 结果

模型的评价与验证

我们选择三个模型的融合模型为最终结果，kaggle 得分 0.05221。为了测试模型的鲁棒性，我们从网上随机下载了 10 张猫和狗的图像，根据模型的预测如下图：



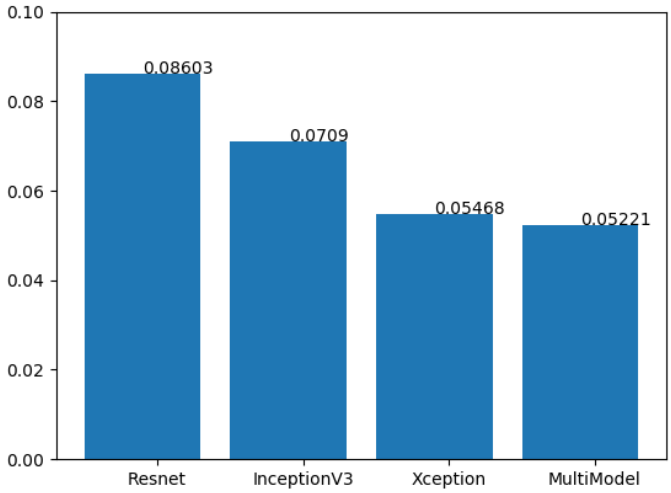
可以看到，所有的图像都预测正确，说明模型至少能够适应微小的变化，具有很好的泛化能力。

合理性分析

我们选择的最终模型在验证集上的准确率为 0.9962，在 kaggle 上得分 0.05221；对网络上随机下载的 10 张图片都能够正确预测，说明我们的模型能够对猫和狗进行分类，达到了项目的目的。

V. 项目结论

在本项目中，在对数据进行清洗剔除掉 outlier 之后，通过 3 种不同的模型作为预训练模型，利用迁移学习，分别对三个模型进行训练得到最优模型。利用三种模型分别对测试集数据图片进行预测，得到的 kaggle 如下图，从 Resnet50 到 Xception 依次降低，模型的表现越来越好。最终，我们将三种模型的结果进行融合，将三个模型的结果进行加权平均得到融合模型的 kaggle 得分 0.05221，达到了项目的预期。因此将三种模型的融合模型作为本项目的最终模型。



在整个过程中，学习到了很多内容，例如在载入数据集中的过程，最初打算生成 pickle 文件来直接载入，最终发现数据集过大导致的生成和载入非常慢，甚至电脑无法承受，到利用 keras 从文件夹下载入数据，即不占用内存，也不受数据多少的限制。

另外，keras[11]在本项目中起到了非常重要的作用，可以说 keras 大大加快了开发的速度，但在项目中学习到的只是一部分内容，还有很大一部分需要今后不断学习。

需要作出的改进

在本项目中，训练数据和验证数据的载入是依靠 keras 从两个不同的文件夹中读入，这就造成了训练数据和验证数据必须在事先人为的分开，也就是说真正用于训练的数据会减少，不利于模型的训练。尽管我们用这种方式也达到了项目的目标，但如果能够利用交叉验证的方式载入数据集，就能更充分的利用所有的训练数据，对模型的训练有更大的帮助。

参考

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

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015

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

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. 2014.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of The 32nd International Conference on Machine Learning, pages 448–456, 2015

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015

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

[10] https://github.com/jkjung-avt/keras-cats-dogs-tutorial

[11] https://keras.io/zh/callbacks/

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane,´ R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker,
V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. War- den, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org