机器学习工程师纳米学位毕业项目

题目：猫狗大战

姓名：唐进民

日期：2017 年 6 月 18 日

目录

1、定义

1.1、 项目概述

计算机视觉是一门研究如何使机器“看”的科学，更进一步的说，就是指用摄影机和计算机代替人眼对目标进行识别、跟踪和测量等机器视觉，并进一步做图像处理，用计算机处理成更适合人眼观察或进行仪器检测的图像。

这个项目就是用到了机器视觉对猫狗图像进行识别，其实可以根据提供的数据集不同还可以用于其他事物的识别和判断，有很强的灵活性。

项目使用的是卷积神经网络算法，卷积神经网络早在1997年便被用来解决字符识别问题，然而它最近地广泛应用源于2012年ImageNet图片分类竞赛中，CNN以最高的分类准确率夺得头筹。

使用的项目数据集是从Kaggle上下载的训练和测试数据，训练数据都用相应的Label标记，标记了每张图片是猫还是狗。训练数据一共有25000张图片，测试数据是12500张。测试数据没有Label标记，当对模型训练完成后，用测试集拟合训练的模型，输出Label然后与正确的Label比对，来确定准确率。

1.2、 问题说明

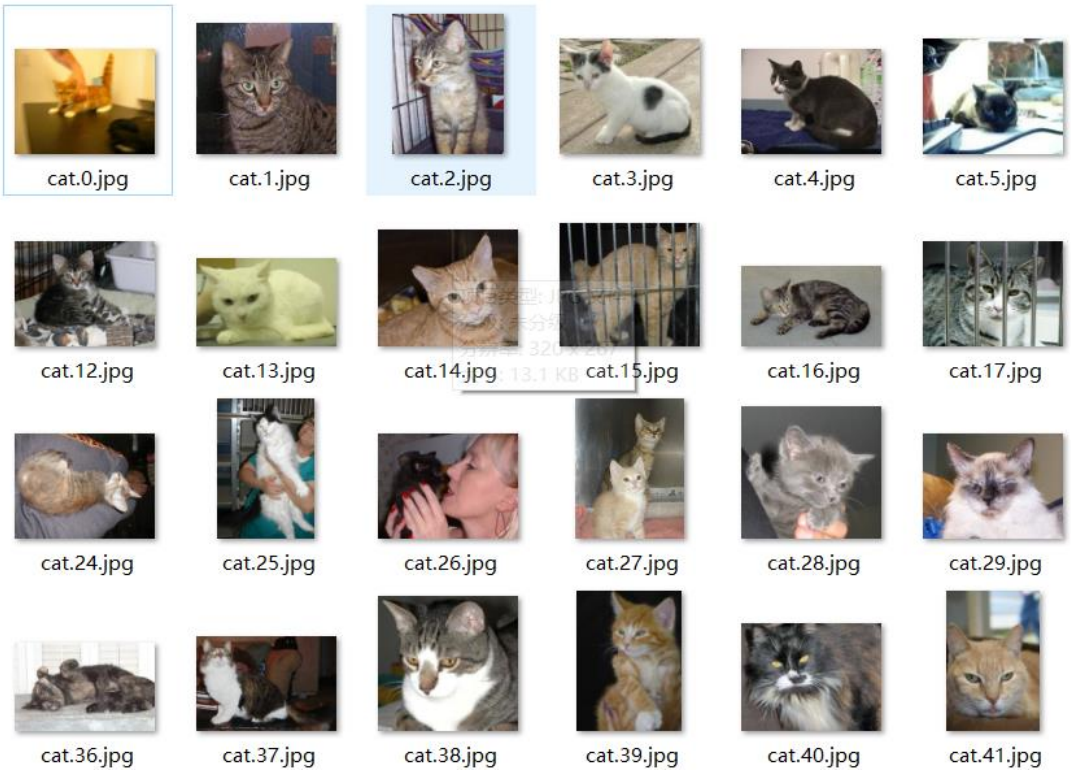
项目最终要解决的问题是在输入一张猫或者的狗的照片后，进行判断这张图片是猫的可能性是多少，是狗的可能性是多少。例如，输入的是狗的图片，模型判断是狗的可能性为 90%以上，然后给予明确的答复。这里我答复就是给输出的图片打上 Label，如果确认是够就标记为 1，确认是猫就标记为 0。

所以我题的困难主要将是对图片的预处理和训练模型的选择和参数的调优上。如果能够获得好的原始数据和优化的模型，那么对结果的预测准确率提升将是事半功倍的。

首先图片大部分是分辨率不同，所以大小也是不一样的。所以在输入到模型进行训练前，要对数据进行预处理，统一数据的输入格式。

其次在模型选择上，我使用的是卷积神经网络模型，通过输入大量带有标签的数据对模型进行训练，参数的调优要经过多次实验后才能明确确定。

输入训练数据的选择上，我使用的是每张带有猫或者狗的标记的图片，如下图所示:



当我完成这个模型训练后，就能够对新的无标签数据进行判断。之前也提到过，输出结果我已经明确了只有两种类型，所以在考虑输出结果是我可以将 0 和 1 作为图片是猫或者狗的标签。然后得到的结果再和标准结果进行比对，计算模型判断新数据的准确率。如果太低就再对模型进行优化，调节参数，改变图片输入类型等，最终得到一个理想的结果。

1.3、 评价指标

首先我们可以了解一下准确率。准确率简单的计算公式如下所示:

$$P(\text{准确率}) = \frac{A}{A+B}$$

其中 A 表示为满足条件的数据，B 为不满足条件的数据。总的数据量是 A+B。

了解了准确率后，我选定的评价是有效准确率。对于原始数据我将数据集分成训练集合和测试集合。训练集合对我的模型进行训练，测试集合用来验证结果的有效准确性在模型的训练过程中，会有一个自己的准确性评估，但是这个准确性足够高并不能代表模型的识别能力更强，因为有可能这个时候模型已经过拟合。所以我单独划分一个测试集合对模型的图片识别准确率进行评估更加科学、可靠。

2、数据分析

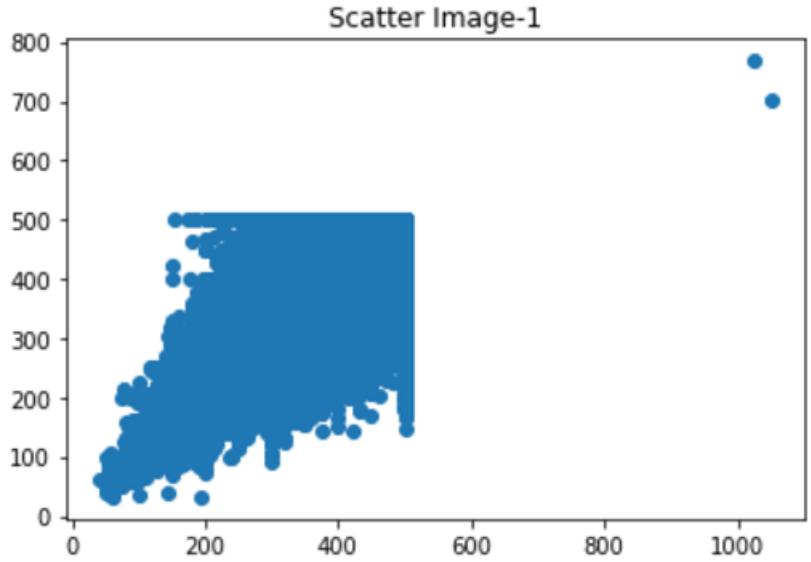
2.1、数据研究

从 Kaggle 上下载的数据集主要包含两个部分，一个测试集一个训练集。其中训练集包含了 25000 张带有猫或者狗标签的图片。测试集包含了 12500 张没有任何标签的猫或者狗的图片。

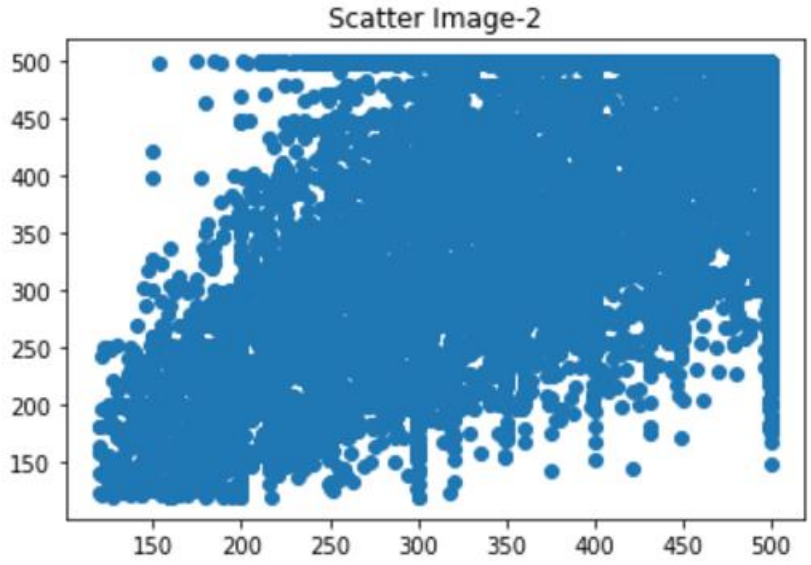
无论是测试集还是训练集，包含的图片格式尺寸都不是统一的，在进行研究前必须进行统一化。下图是随机选取的两张图片查看详细信息。

可以很明显的看出分辨率是不一样的。

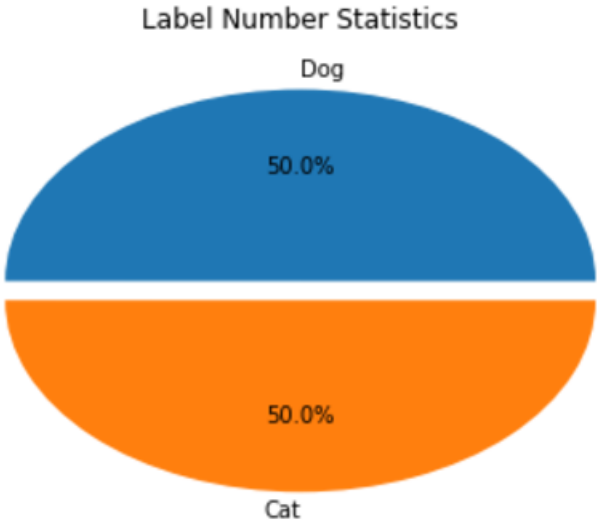
这里也引出了一个新的问题，这些数据在作为输入之前我必须对数据进行分辨率的转换。这里我将老的图片转换成到分辨率为 120X120 的新图片。在这个过程中，也必然会出现一部分异常值，和我不需要的分辨率图片。可视化我的原始数据集分布如下图:



可以很明显看到存在异常值，但是数量不多，对输入数据的影响较小。然后我将不满住需求的图片剔除，可视化如下图所示：



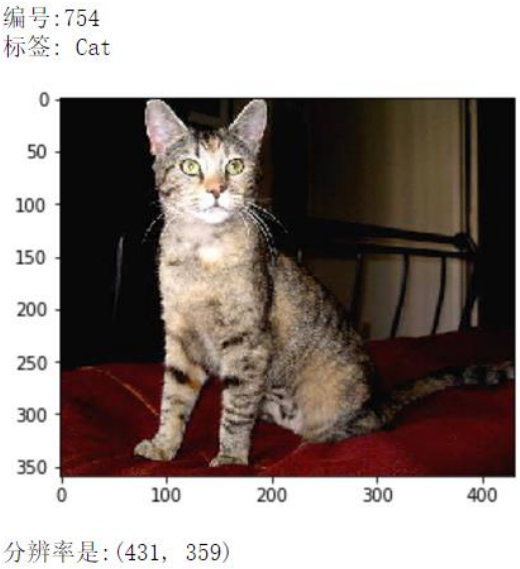
在不没有做图片剔除前，数据的总数为 25000。处理后图片的总数为 24586。有少量的减少。然后我继续了解一下数据集中猫狗图片分布占比是多少，可视化结果输出如下：



从图中可以看到，猫狗图片比率为各占 50%。

2.2、探索性可视化

现在我使用随机函数，随机抽取了一张图片来查看他的各种属性值，运行后图片的输出如下:



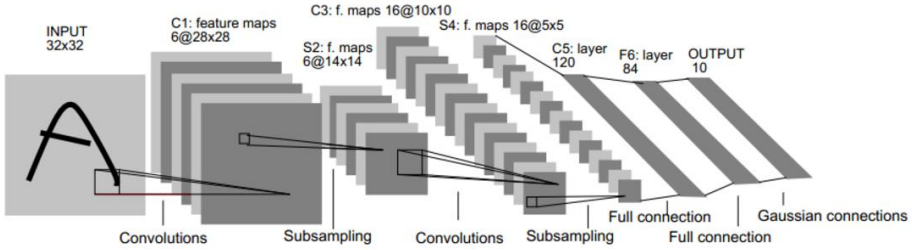
简单的可以看出，抽取图片的编号为 4355，这个图片的额标签是 Dog 和图片上的图画是吻合的。输出 Shape 值是 (304, 377)，可以知道这张图片的分辨率

为 304x377 的。

2.3、算法与方法

我这里选择的卷积神经网络对模型进行训练，卷积神经网络是近年发展起来，并引起广泛重视的一种高效识别方法。20 世纪 60 年代，Hubel 和 Wiesel 在研究猫脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网络（Convolutional Neural Networks-简称 CNN）。现在，CNN 已经成为众多科学领域的研究热点之一，特别是在模式分类领域，由于该网络避免了对图像的复杂前期预处理，可以直接输入原始图像，因而得到了更为广泛的应用。 K.Fukushima 在 1980 年提出的新识别机是卷积神经网络的第一个实现网络。随后，更多的科研工作者对该网络进行了改进。其中，具有代表性的研究成果是 Alexander 和 Taylor 提出的“改进认知机”，该方法综合了各种改进方法的优点并避免了耗时的误差反向传播。

先看下典型的卷积神经网络模型，这里可以看下经典的额 LeNet5 的卷积模型如下图:



在这个经典的卷积神经网络模型中，我们可以看到卷积层（Convolutions），池化层（模型中是 Subsampling），全连接层（Full connection），和激活函数。

这里输入的是一张 32x32 大小的数据。首先经过第一层卷积模的 Patch 过滤得到 6 层的 28x28 的数据，然后经过池化层的过滤达到 6 层的 14x14 的数据，之后的操作类似。经过两层卷积和两层池化后，进行两次全连接操作，没次全连接完成后要经过激活函数来输出数据，激活函数的功能就是然满足条件的数据才能进行结果输出。最后经过高斯连接层进行结果的输出。

下图是几种常用的卷积模型:

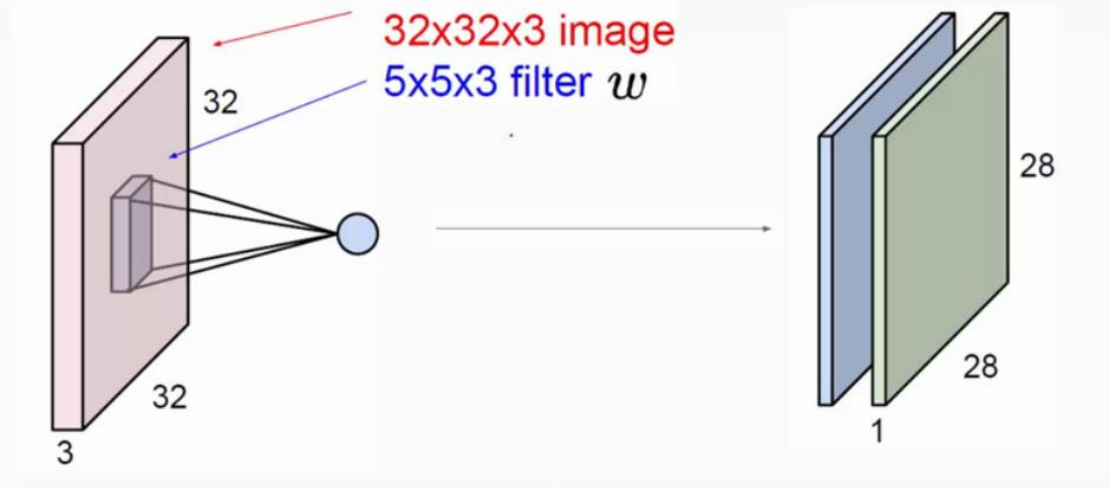
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

D 是我这个项目使用的 VGG16 模型。

在典型的架构中，它包含了 4 层卷积和 4 层池化，3 层全连接层，最后用 soft-max 作为结果的输出。soft-max 的函数如下图:

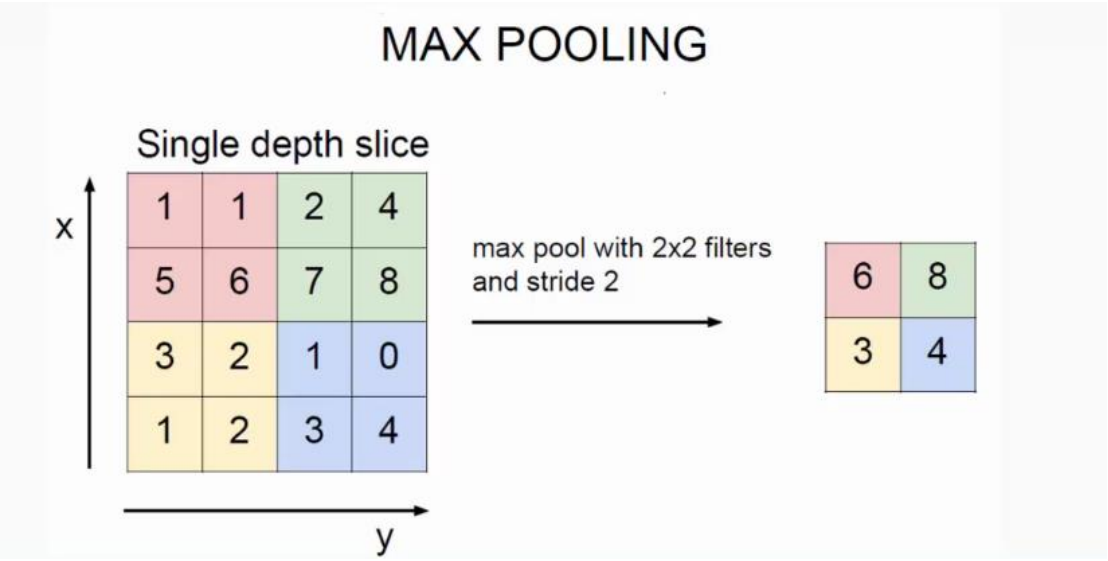
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

首先说的是卷积层，卷积成的主要作用就是对图像的特征提取，通过滤波器对原始输入图像进行过滤，得到一个新的特征图，原理如下所示:



但是就出现了一个新的问题，我们只是想对主要关心的特征进行提取训练，这里就映入了池化层。

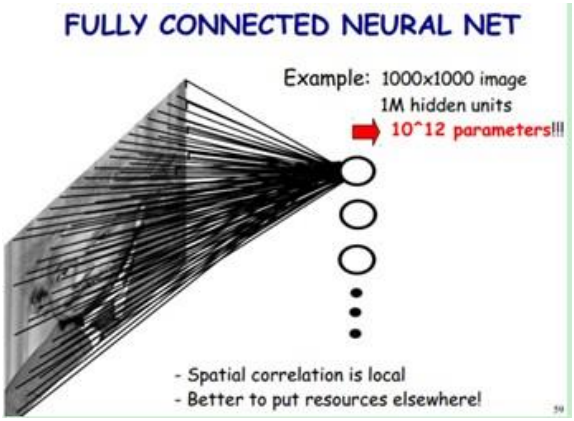
池化的主要作用是对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征。原理如下如所示：



这里展示的是最大池化层，通过滤波器选取每个块的最大参数进行重新组合然后的到新的核心特征图。

多层卷积就是卷积层加池化层的多层迭代最后提取出核心特征图。但是又有个新的问题，得到这些核心特征图后我们需要进行处理来预测结果，所以必须将这些特征图又重新进行归一化。这里就用到了全连接层。

全连接的作用就是连接所有的特征，并将输出值送给分类器。通过分类器预测结果值。如下图就是全连接层的原来图:



说了卷积层、池化层、全连接层之后，我们还要关注的一个核心问题就是模型中使用的优化方法。在之后会用的两个优化方法分别是 RMSprop 和 SGD。

SGD 全名 stochastic gradient descent， 即随机梯度下降。算法的每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本{x1,⋯,xm},以及相关的输y_i
2. 计算梯度和误差并更新参数:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta),y_i)$$
$$\theta \leftarrow \theta - \epsilon\hat{g}$$

这个优化算法的优点是训练速度快,对于很大的数据集,也能够以较快的速度收敛。不足之处由于是抽取,因此不可避免的,得到的梯度肯定有误差.因此学习速率需要逐渐减小.否则模型无法收敛,因为误差,所以每一次迭代的梯度受抽样的影响比较大,也就是说梯度含有比较大的噪声,不能很好的反映真实梯度。

RMSProp 优化算法通过引入一个衰减系数，让 r 每回合都衰减一定比例。算法的每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本{x1,⋯,xm},以及相关的输y_i

2. 计算梯度和误差,更新 r,再根据 r 和梯度计算参数更新量

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta),y_i)$$
$$r \leftarrow \rho r + (1-\rho)\hat{g}\odot\hat{g}$$
$$\triangle\theta = -\frac{\epsilon}{\delta+\sqrt{r}}\odot\hat{g}$$
$$\theta \leftarrow \theta + \triangle\theta$$

这种优化算法方法很好的解决了深度学习中过早结束的问题,适合处理非平稳目标，对于 RNN 效果很好。缺点是引入了新的超参，衰减系数 ρ 而且依赖于全局学习速率。

在这里选择 VGG16 模型作为此次训练的核心模型最主要的原因是，VGG16 在计算机视觉和图像识别上有很强的实用性。能够得到很好的预期结果。然而最大的问题是训练过程中会消耗非常多的计算机资源和时间。所以我们需要兼顾各方面，采用更加合理高效的方式完成这个项目。

2.4、基准测试

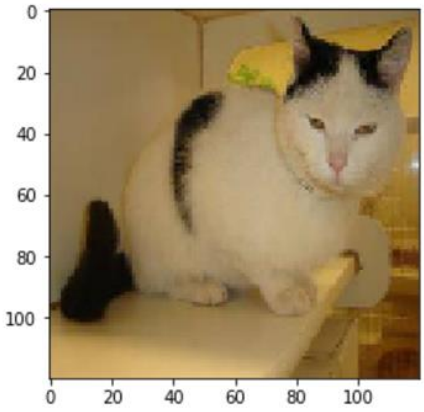
这里我确定基准测试的阀值不要求太高,因为按照神经网络的特性可以知道,在卷积神经网络的层数越多的时候,分类器的表现会趋于越好,但是层数的增多,模型复杂度增加，必然导致训练花费的时间大大增加。因为我的时间和计算资源都是有限的，所以综合考量，选择一个比较适中的基准阀值。

我们拟定基准测试的阀值为模型的有效准确率在94%以上就算合格。

3.1、数据预处理

之前说过输入的训练数据和测试数据中很多图片的 Shape 都是不一样的，所以必须进行统一的格式转换。我使用的是 PIL 库中自带的图片转换函数。如下图是已经转换成功的，120x120 的图片:

随机输出一张分辨率转换后的图片。



对所有训练集合的图片转换成功后作为新的输入数据集合注入到模型中去。并且训练集合我们将其进行切分，留出一部分在训练集合完成训练后进行准确率计算。

3.2、实施

数据处理完毕后，就可以用新的转换好统一格式的数据注入至卷积神经网络模型中了。我使用的是 VGG16 模型进行训练。在实施中发现当保证训练模型复杂度不变的情况下，编码过程层中输入数据的 Size 和模型的层数、训练中使用的区块大小都影响着模型训练的速度和最后的准确度。

模型建立完毕后实施中需要解决两个问题。一个是计算机计算资源不足，其次是防止模型的过拟合。

计算机资源不够的问题，我申请了亚马逊的 AWS 虚拟主机，配置为

1. 选择 AMI　2. 选择实例类型　3. 配置实例　4. 添加存储　5. Add Tags　6. 配置安全组　7. 审核

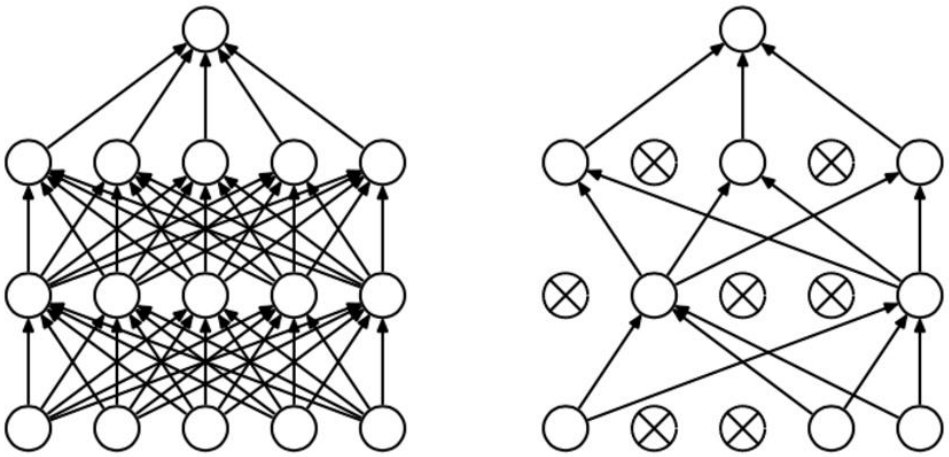
步骤 2: 选择一个实例类型

☐	计算优化	c3.8xlarge	32	60	2 x 320 (SSD)	-	10 Gb	是
☐	GPU 实例	g2.2xlarge	8	15	1 x 60 (SSD)	是	高	-
☐	GPU 实例	g2.8xlarge	32	60	2 x 120 (SSD)	-	10 Gb	-
■	GPU 计算	p2.xlarge	4	61	仅限于 EBS	是	高	-
☐	GPU 计算	p2.8xlarge	32	488	仅限于 EBS	是	10 Gb	-
☐	GPU 计算	p2.16xlarge	64	732	仅限于 EBS	是	20 Gb	-

虽然不是很高配置，考虑到性价比的问题也是已经够用了。

解决过拟合的问题，我们需要将神经网络模型中的部分单元进行丢弃。因为随着迭代次数的增加，我们将会发现测试数据的 loss 值和训练数据的 loss 存在着巨大的差距， 随着迭代次数增加，train loss 越来越好，但 test loss 的结果确越来越差，test loss 和 train loss 差距越来越大，模型开始过拟合。

Dropout 是指对于神经网络单元按照一定的概率将其暂时从网络中丢弃，从而解决过拟合问题



输入数据的 Size 最初选择的是 120x120x3，模型训练选择的划分区块为 32，迭代次数是 20。在 AWS 上训练完成所花费的训练时间为 40 分钟左右。

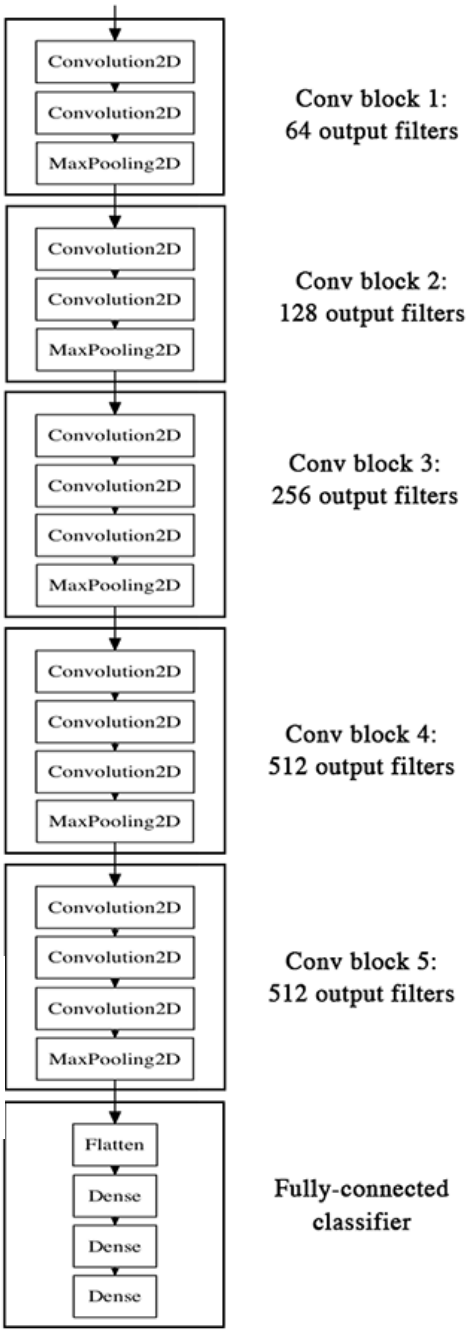
后 10 个 Epoch 的训练过程如下图所示:

```
Epoch 10/20
16128/16128 [==============================] - 122s - loss: 0.2732 - acc: 0.9062 - val_loss: 0.3621 - val_acc: 0.8715
Epoch 11/20
16128/16128 [==============================] - 122s - loss: 0.2614 - acc: 0.9095 - val_loss: 0.2917 - val_acc: 0.8740
Epoch 12/20
16128/16128 [==============================] - 122s - loss: 0.2486 - acc: 0.9159 - val_loss: 0.3650 - val_acc: 0.8844
Epoch 13/20
16128/16128 [==============================] - 122s - loss: 0.2719 - acc: 0.9123 - val_loss: 0.2990 - val_acc: 0.8760
Epoch 14/20
16128/16128 [==============================] - 122s - loss: 0.2665 - acc: 0.9153 - val_loss: 0.3123 - val_acc: 0.8757
Epoch 15/20
16128/16128 [==============================] - 122s - loss: 0.2670 - acc: 0.9147 - val_loss: 0.4473 - val_acc: 0.8879
Epoch 16/20
16128/16128 [==============================] - 122s - loss: 0.3360 - acc: 0.9110 - val_loss: 0.3841 - val_acc: 0.8755
Epoch 17/20
16128/16128 [==============================] - 122s - loss: 0.2903 - acc: 0.9116 - val_loss: 0.3008 - val_acc: 0.8790
Epoch 18/20
16128/16128 [==============================] - 122s - loss: 0.2938 - acc: 0.9117 - val_loss: 0.4868 - val_acc: 0.8584
Epoch 19/20
16128/16128 [==============================] - 122s - loss: 0.3200 - acc: 0.9095 - val_loss: 0.4207 - val_acc: 0.8891
Epoch 20/20
16128/16128 [==============================] - 122s - loss: 0.3327 - acc: 0.9093 - val_loss: 0.4153 - val_acc: 0.8886
```

从图中可以看出，训练得到的准确为已经达到 90%以上，但是有效准确率最高为 88.86%。可见效果不理想，需要对模型进行改进。

3.3、改进

首先我改变了模型的复杂度，因为之前使用的不是标准的 VGG16 模型架构，标准架构如下如所示:



做了更改后效果不理想,不仅结果有效准确率没有明显提升还大大增加了训练时间。而且过拟合。然后又对输入图的分辨率进行放大，使用 240X240X3 分辨率的输入数据集，有效准确率能够达到 90%左右，但是消耗大量的训练时间，不能很好的对模型进行调整。

最后使用了原始分辨率图片，然后在 BatchSize 上做了改造。先使用小的 BatchSize 对模型进行训练，这样在前期训练可以防止过拟合的发生。然后使用大的 BatchSize 对模型进行训练，这样可以提升模型的准确性，防止发生局部最

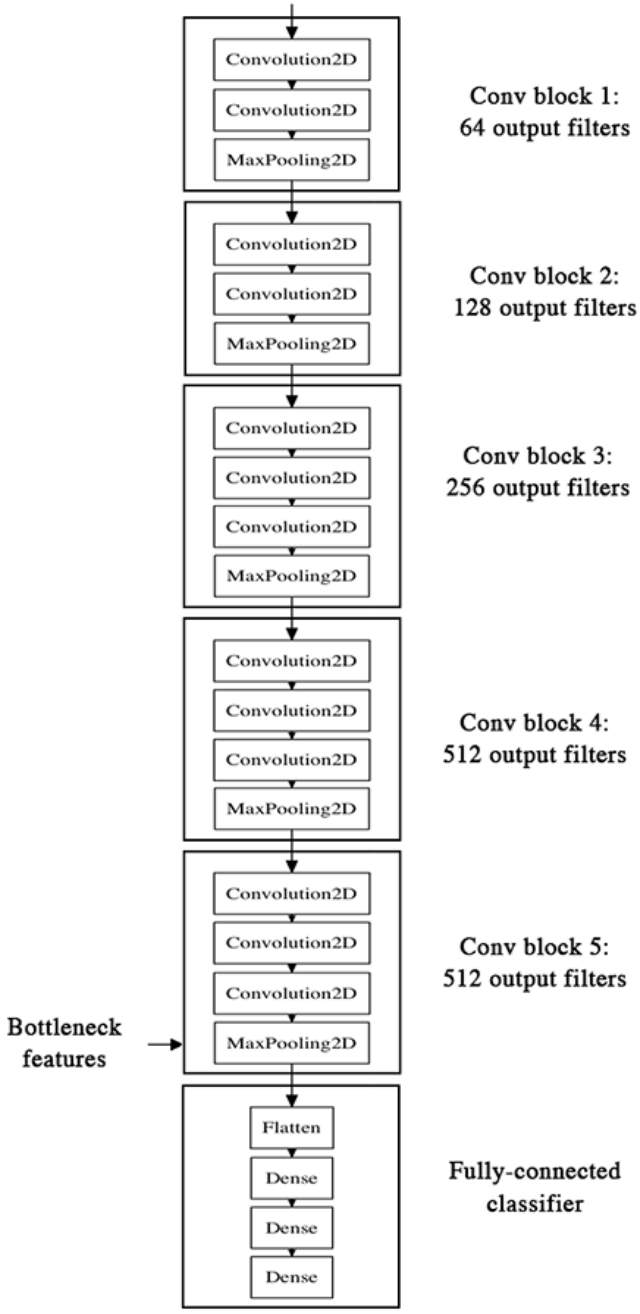
小，训练的后 10 个 Epoch 的如下图所示:

```
Epoch 5/15
15734/15734 [==============================] - 82s - loss: 0.1603 - acc: 0.9399 - val_loss: 0.3595 - val_acc: 0.9118
Epoch 6/15
15734/15734 [==============================] - 82s - loss: 0.1433 - acc: 0.9464 - val_loss: 0.2536 - val_acc: 0.9146
Epoch 7/15
15734/15734 [==============================] - 82s - loss: 0.1395 - acc: 0.9479 - val_loss: 0.2837 - val_acc: 0.9070
Epoch 8/15
15734/15734 [==============================] - 82s - loss: 0.1252 - acc: 0.9532 - val_loss: 0.3062 - val_acc: 0.9065
Epoch 9/15
15734/15734 [==============================] - 82s - loss: 0.1211 - acc: 0.9548 - val_loss: 0.3475 - val_acc: 0.9067
Epoch 10/15
15734/15734 [==============================] - 82s - loss: 0.1181 - acc: 0.9586 - val_loss: 0.3116 - val_acc: 0.8704
Epoch 11/15
15734/15734 [==============================] - 82s - loss: 0.1080 - acc: 0.9604 - val_loss: 0.5338 - val_acc: 0.8889
Epoch 12/15
15734/15734 [==============================] - 82s - loss: 0.1070 - acc: 0.9608 - val_loss: 0.3238 - val_acc: 0.9141
Epoch 13/15
15734/15734 [==============================] - 82s - loss: 0.1026 - acc: 0.9620 - val_loss: 0.3312 - val_acc: 0.9115
Epoch 14/15
15734/15734 [==============================] - 82s - loss: 0.0926 - acc: 0.9645 - val_loss: 0.2908 - val_acc: 0.8910
Epoch 15/15
15734/15734 [==============================] - 82s - loss: 0.0830 - acc: 0.9691 - val_loss: 0.3353 - val_acc: 0.9105
```

从上图可以看到，模型最大的有效准确率已经可以达到 91.41%了。但是还是和基准阀值有所差距。

再次对模型进行优化，使用的方法是 Bottleneck features 和 Fine-tune 的方法。

VGG16 的 Bottleneck features 的网络架构如下图所示:



如上如所示，为了方便对模型进行快速的训练，这里做了两个改进。首先，使用的是部分训练数据和测试数据。选取了 1000 张狗和 1000 张的猫的图片组成新的容量为 2000 张图片的训练集合，选取了 400 张狗和 400 张的猫的图片组成新的容量为 800 张的测试集合。其次，因为建立的是 VGG16 标准模型，为了减少在卷积层花费大量的训练时间，导入了 VGG16_Weight.h5 文件来设置卷积层的权

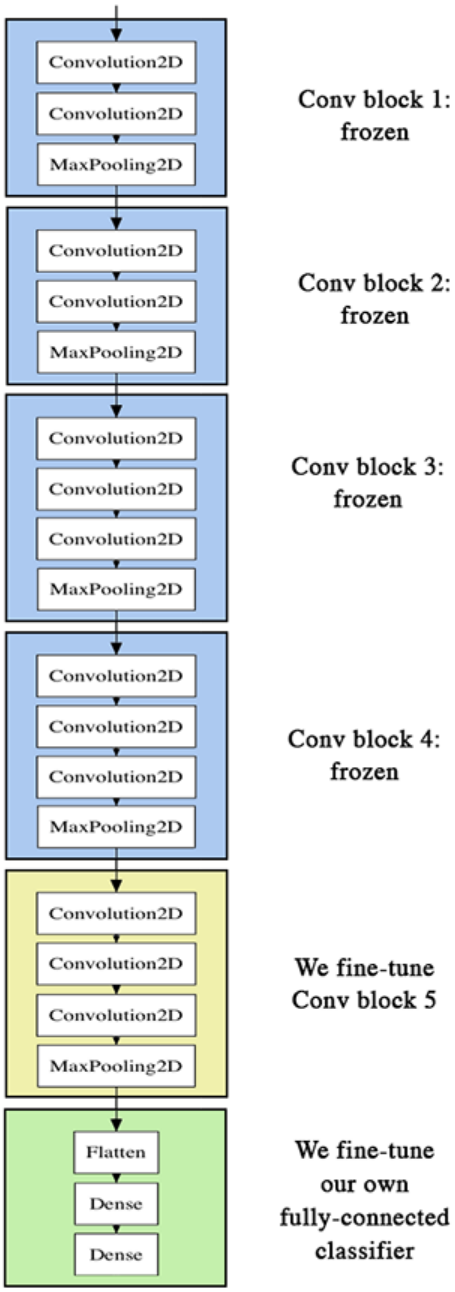
重。然后只是对全连接层之后的部分进行训练，得到权重保存为 h5 文件供后面的 Fine-tune 使用。

这里的 Epoch 设置为 50，将最后 10 次的输入如下图所示:

```
Epoch 40/50
2000/2000 [==============================] - 0s - loss: 0.0150 - acc: 0.9940 - val_loss: 0.8967 - val_acc: 0.9012
Epoch 41/50
2000/2000 [==============================] - 0s - loss: 0.0214 - acc: 0.9940 - val_loss: 0.9510 - val_acc: 0.8962
Epoch 42/50
2000/2000 [==============================] - 0s - loss: 0.0136 - acc: 0.9930 - val_loss: 0.9581 - val_acc: 0.8938
Epoch 43/50
2000/2000 [==============================] - 0s - loss: 0.0230 - acc: 0.9940 - val_loss: 0.9318 - val_acc: 0.8962
Epoch 44/50
2000/2000 [==============================] - 0s - loss: 0.0139 - acc: 0.9945 - val_loss: 0.9270 - val_acc: 0.8950
Epoch 45/50
2000/2000 [==============================] - 0s - loss: 0.0060 - acc: 0.9980 - val_loss: 0.9286 - val_acc: 0.8962
Epoch 46/50
2000/2000 [==============================] - 0s - loss: 0.0134 - acc: 0.9955 - val_loss: 0.9363 - val_acc: 0.9000
Epoch 47/50
2000/2000 [==============================] - 0s - loss: 0.0194 - acc: 0.9950 - val_loss: 0.9239 - val_acc: 0.8938
Epoch 48/50
2000/2000 [==============================] - 0s - loss: 0.0096 - acc: 0.9955 - val_loss: 0.9804 - val_acc: 0.9012
Epoch 49/50
2000/2000 [==============================] - 0s - loss: 0.0199 - acc: 0.9940 - val_loss: 0.9265 - val_acc: 0.8988
Epoch 50/50
2000/2000 [==============================] - 0s - loss: 0.0089 - acc: 0.9965 - val_loss: 1.0324 - val_acc: 0.8950
```

可以看到这里的最高有效准确率已经突破了 90%，虽然比之前的差一点，但是可以看到这里的训练时间平均不足 1 秒，而且我使用的还只是部分训练集。因为 VGG16 的训练是非常耗费计算资源的，这样极大的提升了计算效率，因为我们应该将重点工作关注在接下来的 Fine-Tune 上。

Fine-Tune 的网络架构模型图如下所示:



从上图可以看出，我需要做的是冻结除最后一个卷积层以外的所有卷积层的权重更新。然后拿最后一个卷积层和全连接层后部分进行 Fine-Tune。可以导入 VGG16_Weight.h5 文件和之前训练得到的全连接层部分的 h5 文件。

Epoch 设置为 10 次，输入训练结果，如下:

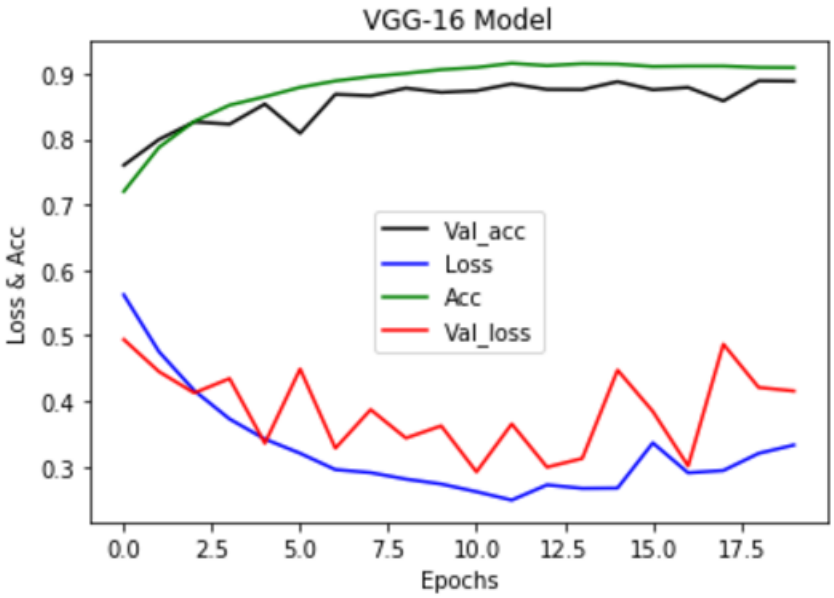
```
Epoch 1/10
2000/2000 [==============================] - 514s - loss: 3.0635e-04 - acc: 0.9998 - val_loss: 0.5396 - val_acc: 0.9334
Epoch 2/10
2000/2000 [==============================] - 513s - loss: 1.2537e-04 - acc: 1.0000 - val_loss: 0.5341 - val_acc: 0.9400
Epoch 3/10
2000/2000 [==============================] - 515s - loss: 6.8741e-04 - acc: 0.9998 - val_loss: 0.5451 - val_acc: 0.9287
Epoch 4/10
2000/2000 [==============================] - 514s - loss: 5.9468e-04 - acc: 0.9997 - val_loss: 0.4978 - val_acc: 0.9404
Epoch 5/10
2000/2000 [==============================] - 515s - loss: 0.0011 - acc: 0.9998 - val_loss: 0.4324 - val_acc: 0.9386
Epoch 6/10
2000/2000 [==============================] - 515s - loss: 0.0020 - acc: 0.9996 - val_loss: 0.4647 - val_acc: 0.9400
Epoch 7/10
2000/2000 [==============================] - 514s - loss: 3.4394e-04 - acc: 0.9999 - val_loss: 0.5133 - val_acc: 0.9437
Epoch 8/10
2000/2000 [==============================] - 514s - loss: 6.3384e-04 - acc: 0.9998 - val_loss: 0.5308 - val_acc: 0.9381
Epoch 9/10
2000/2000 [==============================] - 515s - loss: 0.0020 - acc: 0.9995 - val_loss: 0.5004 - val_acc: 0.9339
Epoch 10/10
2000/2000 [==============================] - 515s - loss: 3.2302e-04 - acc: 0.9999 - val_loss: 0.5087 - val_acc: 0.9424
```

可见预测效果有了很大的提升，结果中最大准确率已经超过 94%，结果达到了基准阀值。

4.1、自由形态的可视化

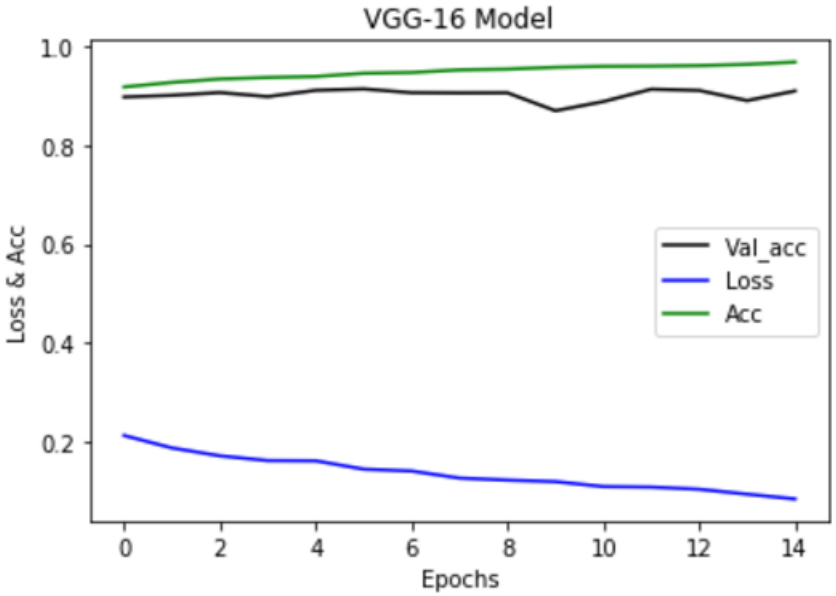
现在将三次总要训练结果进行可视化输出。

在未做改进的模型训练中，我们选择了 20 次迭代训练。这是迭代 20 次的训练 Loss、Accuracy、Vlidate_Loss 和 Vlidate_Accuracy 结果可视化输出:

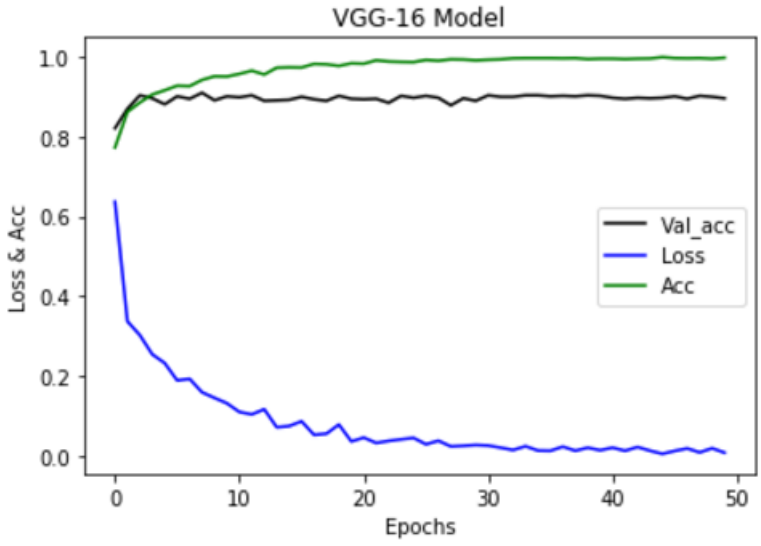


进行了第一次模型的重要改进。我们选择了 20 次迭代训练。这是迭代 20

次的训练 Loss、Accuracy 和 Vlidate_Accuracy 结果可视化输出:

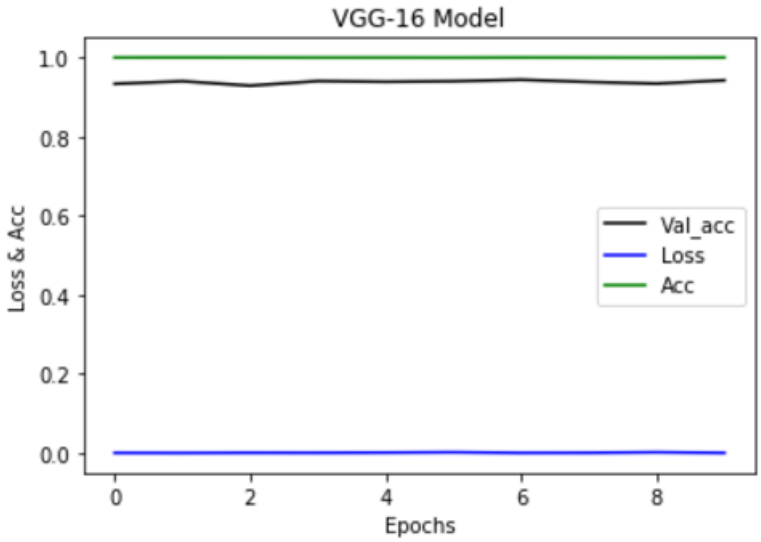


进行了第二次模型的重要改进。我们选择了 50 次迭代训练。这是进行 Fine-Tune 前迭代 50 次的训练 Loss、Accuracy 和 Vlidate_Accuracy 结果可视化输出:



进行了第二次模型的重要改进。我们选择了 10 次迭代训练。这是进行

Fine-Tune 后迭代 10 次的训练 Loss、Accuracy 和 Vlidate_Accuracy 结果可视化输出:



4.2、思考

这个项目中因为计算机资源的不足导致使用更加复杂的模型会增加大量的训练时间、消耗更多的计算资源。必然导致每次调节参数需要大量的时间进行重新训练。这个问题的解决方法需要通过提升硬件才能得到根本解决。

当计算能力得到解决后我们才能够进行更加多的尝试。但是其实图像的识别难点并不在于此，如果想要得到更好的预测结果，且是对图像周围环境的抗干扰性也要考虑在内，还有多种猫或者狗出现时的干扰性等等。

4.3、改进

我们现在能够进行的改进是首先增加模型架构的复杂度，提升计算机的计算能力。这是训练出更好模型的重要条件。

其次，如果不能很好的满足硬件上的条件。还可以通过下面的方法使我现在的模型突破 95%以上的有效正确率：1、适当提升输入的数据量，2、适当提升 Dropout 的值，3、使用 fine-tune 微调更多的卷积块。但是这些尝试都会消耗大量的时间，今后还需探索更多的高效、高准确率的方式实现目的。

参考文献

1、https://github.com/nd009/capstone/tree/master/dog_vs_cat
2、http://www.jianshu.com/p/64172378a178
3、VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION
4、
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
5、http://www.cnblogs.com/zf-blog/p/6075286.html
6、http://blog.csdn.net/u014595019/article/details/52989301