# 机器学习纳米学位

毕业项目 沈振雷 优达学城

2018 年 4 月 22 日

## I. 问题的定义

### 项目概述

图像的识别与分类是人类的基本技能，一个三岁的孩子经过简单的学习就能分辨猫和狗，可以识别众多不同的人，可以避开障碍自由行走，然而这一问题对于计算机却并不简单，甚至长期以来被视为计算机的短板。

早期的图像识别研究主要手段是通过人工设计特征，然后利用传统的机器学习算法进行识别与分类[1]，然而设计好的特征是少数专家才能掌握的艺术，特征的好坏决定了模型表现的上限，特征的设计又费心费力，尽管人们在这一领域取得了不错的进展，然而与人类的能力相比还是相差甚远。

2006 年 Hilton 等人发表了一篇以深度信念网络算法为名的论文[2]，阐明可以用一种被称为"贪婪逐层预训练"的策略有效的训练神经网络来识别图像，此后深度学习迎来了研究的热潮。2014 年 Hilton 和他的学生 Alex Krizhevsky 等人设计的 AlexNet 深度学习模型一举在 ILSVRC 图像分类比赛中取得冠军[3]。AlexNet 采用卷积神经网络自动提取图像特

征，然后输入到全连接网络中进行学习，模型表现远超传统机器学习方法。

如今深度学习在许多图像识别与分类应用上已经超越了人类，广泛应用于自动驾驶、人脸识别，安防，停车场自动计费等等领域。

猫与狗是人类的两大宠物，尽管猫的品种千差万别，狗的形貌也相差甚远，然而人类可以轻易的识别出一个动物是猫还是狗。但是对于计算机来说，并非易事一方面对于猫或者狗，其类内差别很大，不同的品种可能形态、大小、毛色，纹路都相差巨大。另一方面，有些猫和狗有十分相像，计算机如何精确分类依然是一个挑战。

利用深度学习对猫狗进行识别分类属于有监督学习，需要大量已标记的图片。因此我采用 kaggle 上的一个竞赛的数据集：Dogs vs. Cats[4]。这一数据集中，训练集有 25000 张，猫狗各占一半，已经用文件名做了标签。测试集共有 12500 张，没有标定是猫还是狗。我可以通过对有标签的猫狗图片进行学习训练得到模型，然后对测试集进行识别预测验证模型。

## 问题陈述

我要解决的问题如下：

通过对已经标记是猫或者是狗的图片进行学习训练，得到一个模型，使模型可以成功识别测试集中未被标记的猫或者狗的图片，并输出图片是

狗的概率。这是一个典型的有监督学习二分类问题，可以通过卷积神经网络进行学习求解。

kaggle 这一比赛已经很多年，它有一个公开的排行榜展示大家提交的模型评分排行，我的目标是预测成绩排名在 kaggle leader board[5] 上占 Top 100，也就是 LogLoss 低于 0.05629。

### 评价指标

评测的指标我们就选择和 kaggle 平台一致的 LogLoss[6].， 公式如下：

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

$n$ 是测试样本中图像个数

$y_i'$是预测一个图片是狗的概率，1 代表 100%是狗， 0 代表 100%是猫。

$y_i$是一个图片的真实标签，1 代表是狗，0 代表是猫。

$\log(y_i')$ 是自然对数函数，以 e 为底。

## II. 分析

### 数据的探索

如前所述，训练集有 25000 张 jpg 格式的彩色图片，放在一个文件夹里，猫狗各占一半，样例图如图 1 图 2 所示。 可以看出图片是彩色图片，尺寸并不统一。每个图片都用文件名做了标记，如文件名为：cat.0.jpg 表明这是一只猫的图片。文件名为 dog.12499.jpg 表明这是一只狗的图

片。测试集有 12500 张彩色 jpg 图片，但是文件名没有带猫或者狗的信息，只是用标号命名，如 1.jpg, 2.jpg 等等。

图 1 训练集猫

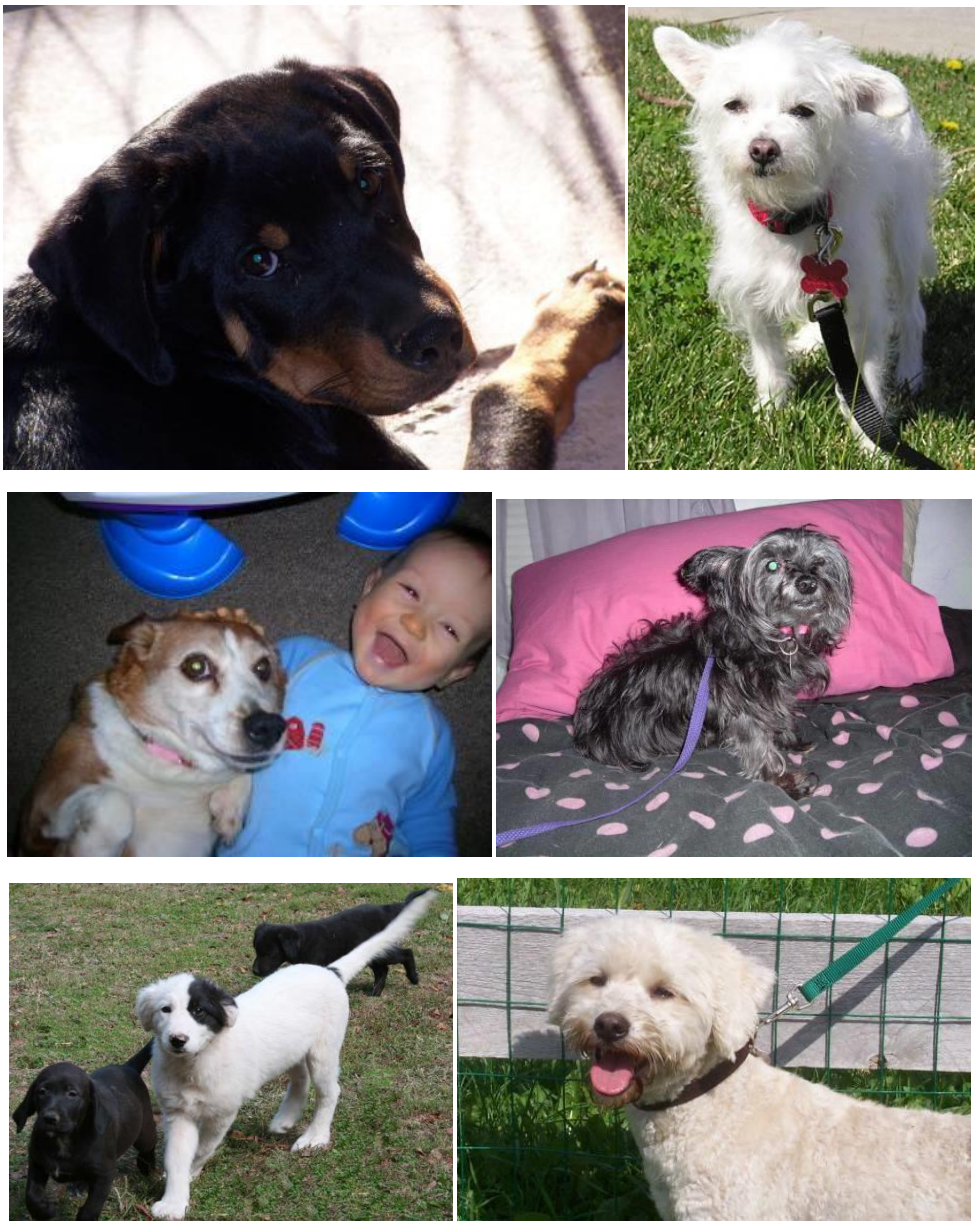

图 2 训练集狗

## 探索性可视化

对于分类问题一个需要关注的数据分布特性是训练集的类别比例，本项目数据集主要有猫狗两个分类，其比例为 1:1 如图 3 所示，这是对训练特别理想。

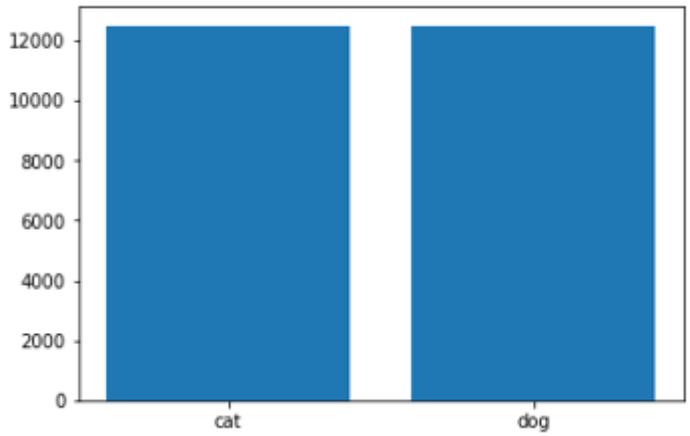


图 3 猫狗图片数量分布

对于数据集的探索，通常需要利用可视化技术发现异常值并处理，然而对于图片来说，我们很难通过常规可视化方法发现异常值，一种可能的方法是用眼睛一张一张图片看过去，但是显然对于 25000 张图片的数据集，眼睛看不太可行。然而，我们可以通过一些预训练模型进行异常图片识别。

这里，我门采用 ResNet50, Xception，和 VGG16 预训练模型。对图片进行预测，如果一个图片在三个模型预测的 Top30 分类中都不含有标签相应分类，就可以视为异常值。采用这一方法，我们从数据集中过滤了出了 65 张异常值图片，占整体数据集的 0.26%。图 4 图 5 展示了这些异常值的部分样例，可以看出，这些图片要么背景复杂，猫狗图片不显眼，要么图片中根本没有猫狗信息，这些图片存在于训练集中，势必对模型造成干扰。

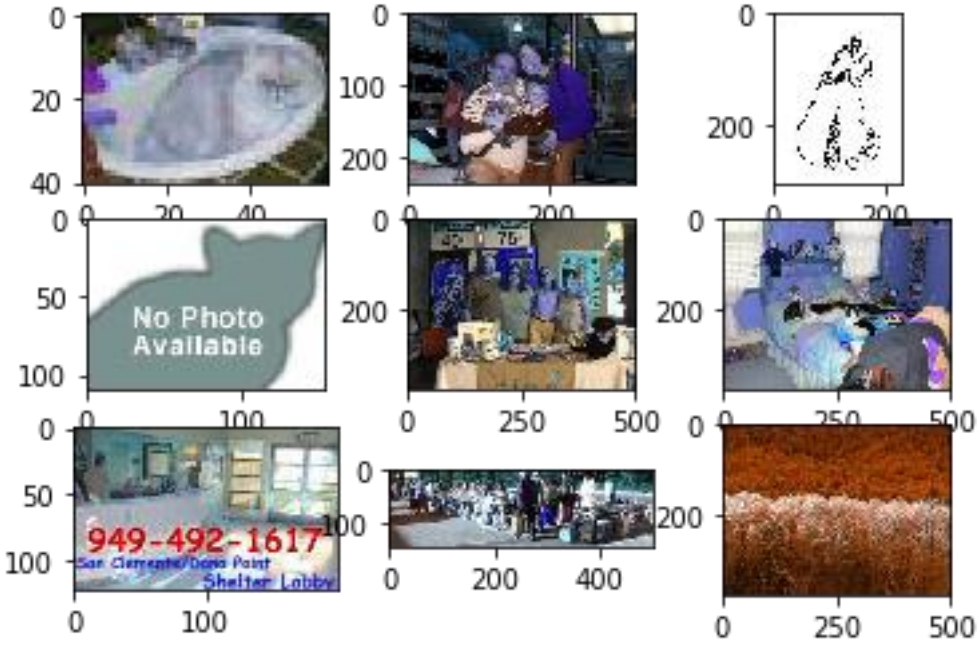


图 4 异常图片

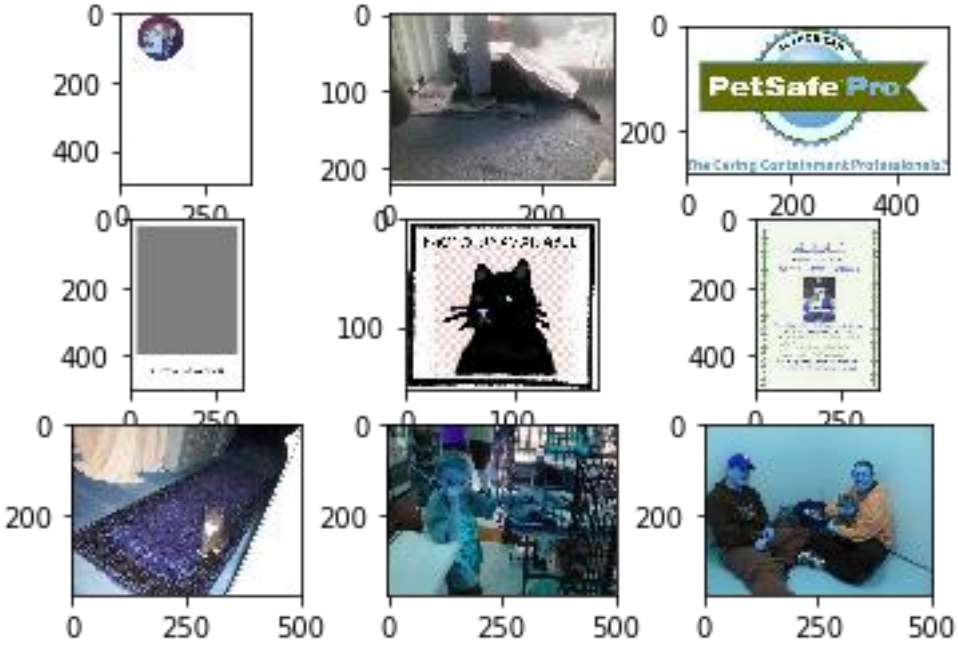


图 5 异常图片

## 算法和技术

这一项目特别适合用迁移学习，谷歌层针对 ImageNet 训练了几个模型，如 ResNet50[7]，InceptionV3[8]， Xception[9]等等，对上千种图片识别精度都极高，但这些都是多分类模型，倘若直接拿来预测猫狗，使用并不方便，且性能并不理想。但是这些模型提取的图像特征是非常有效的，我们可以利用这些模型提取特征，然后用这些特征重新训练神经网络识别猫狗。

ResNet50

在深度神经网络应用于图像识别领域时，深度显得特别重要，因为 CNN（卷积神经网路）能够提取特征并分层表示，层数越多，能够提取的不同 level 的特征越丰富。然而神经网络通常是通过梯度反向传播来训练的，当深度增加过多时，会导致梯度弥散。解决这一问题的方法是采用 Batch Normalization 来对输入层和中间层进行正则化，实践证明，这一方法可以成功的训练较深的网络，然而也就只能到几十层，继续增加层数虽然可以训练，但是训练集的准确率并没有明显提升。也就是说网络出现了退化问题， ResNet 通过残差网络成功解决了这一问题，网络可以有效训练数百层。ResNet 的思想是将网络层表示为残差函数，而实践证明残差网络更容易优化。假设我们的网络映射可以表示为 H(x), 那么从 H(5) =5.1 到 H(5)=5.2 输出变化只变化了越 2%，如果我们把网络表示为残差网络 H(x) = F(x)＋ｘ， 那么我们要学习的函数是，F(x) ＝ H(x)－ｘ，此时，Ｈ（ｘ）输出同样从 5.1 变化到 5.２，那么Ｆ（Ｘ）

的输出则是从 0.1 变化到 0.2 变化了 100%。很显然，这一变化对权重的调整作用更大，ResNet 的残差模块如图 6 所示。

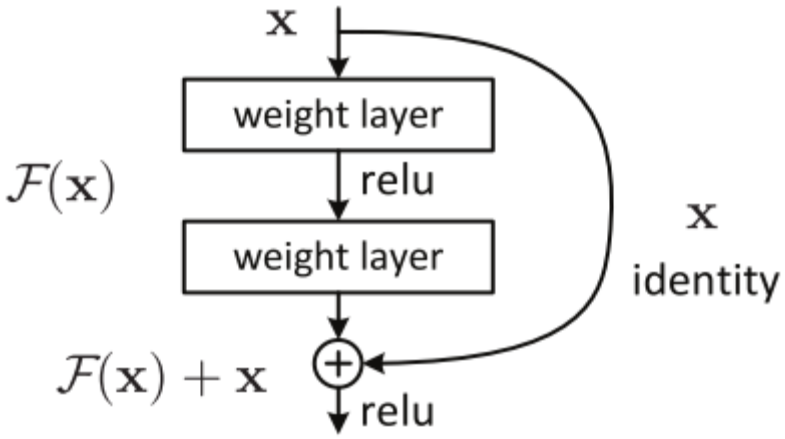


图 6 ResNet 中残差模块

ResNet 作者将 VGG19 设计出 Plain18 和 Plain34 （图 7），并与同等层数的残差网络 ResNet18 和 ResNet34 比较, Plain 网络展现了退化问题，而 ResNet34 表现明显优于 ResNet18 和 Plain34. 本文使用的 ResNet50 即为 50 层的残差网络。

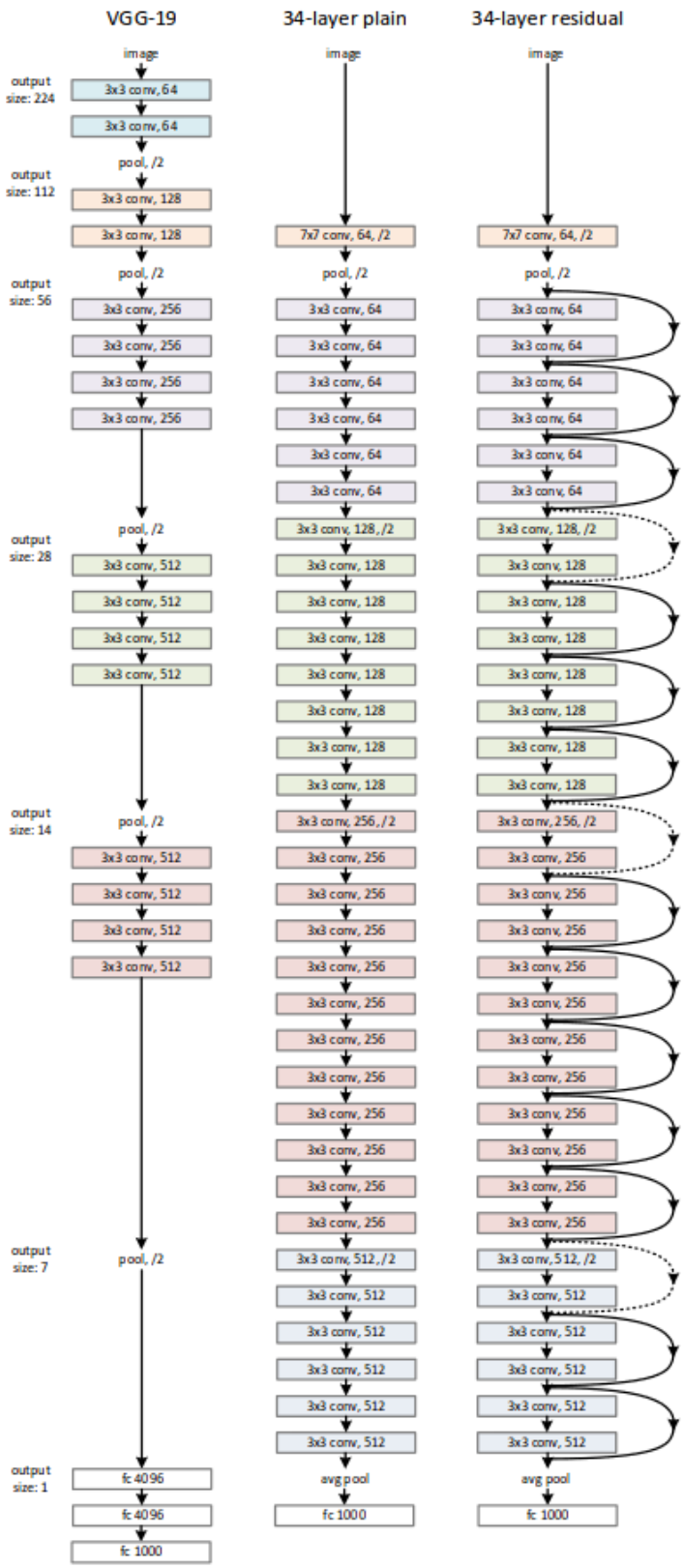


图 7 Plain 网络和残差网络结构对比

InceptionV3

一般的卷积神经网络都是着眼于提高深度，而单层上的卷积网络却只有一种，这样单层网络所能表示特征的能力就受到了限制。那么能不能通过增加单层网络的丰富性来提高模型表现呢，Inception 网络证明这是可行的。Inception 网络通过构造 Inception 单元是成为多级特征提取器。这种单元使用 1X1 卷积， 3X3 卷积、5X5 卷积和一个 3X3 的下采样组合提取特征，这样就产生了 InceptionV1，其结构如图 8 所示。

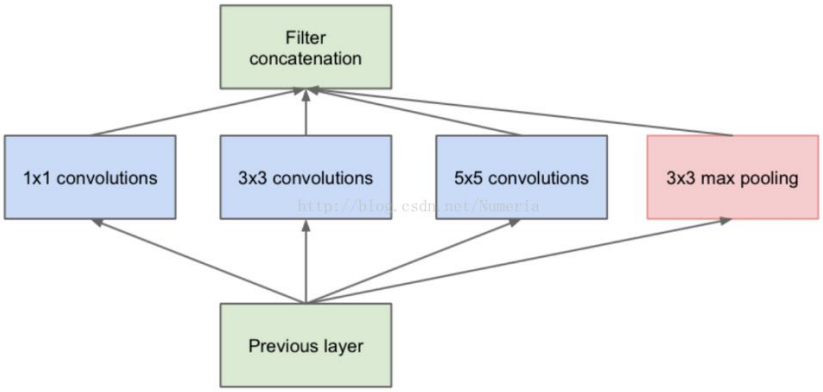


图 8 InceptionV1 单元网络微结构

这种模型确实提高了单层网络的特征提取能力，然而带来的问题是计算量的显著增大，为了解决这一问题，出现了 InceptionV2. InceptionV2 一方面将 5X5 的卷积替换成 2 层 3X3 的卷积，另一方面通过将上一层的输入通过 1X1 卷积进行降维。这样既能大大提升卷积提取特征的能力，又不会使计算量提升太多，其结构单元如图 9 所示。

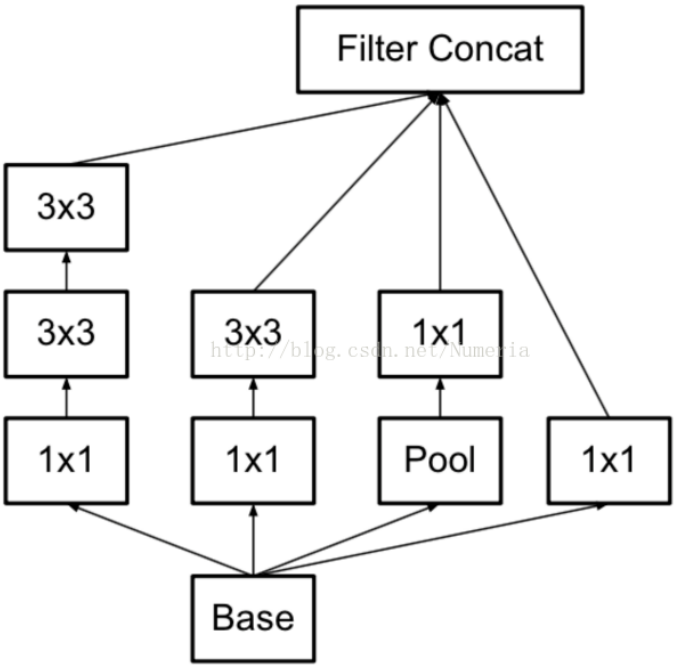


图 9 InceptionV2 单元网络微结构

Inception v3 网络，主要在 v2 的基础上，提出了卷积分解（Factorization），代表作是 Inceptionv3 版本的 GoogleNet。

Inception v3 将 7*7 分解成两个一维的卷积（1*7,7*1），3*3 也是一样（1*3, 3*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。

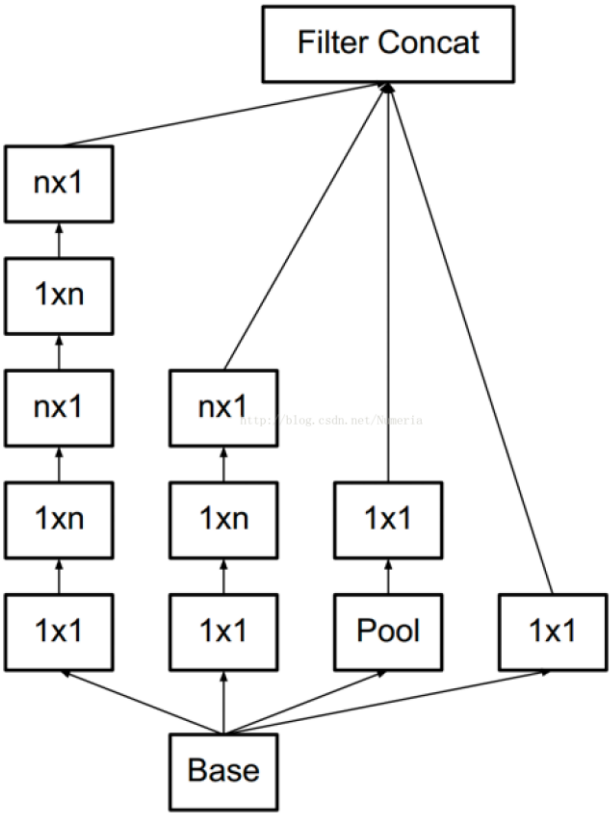


图 10 InceptionV3 模块结构

Xception

Xception 是由 keras 的作者 Francois Chollet 设计的，可以看做是 Inception 的改进版本。其主要思想是实现通道相关性和空间相关性的分离。Inception 主要是通过增加同一层的卷积核数量和多样性来增强特征提取能力，如果我们将这一思想简化，同一层的卷积核全部使用 3X3 卷积，然后再增加卷积核的数量，使其与通道个数相同，那么，每个通道对应着一个卷积，此时我们得到了一个极致的 Inception 单元，实现了深度卷积和空间卷积的分离，如图 11 左所示。1X1 卷积只作用于一个像素的所有通道上，实现了空间分离的深度卷积，而 3X3 卷积只作用于一个通道上，实现了深度分离的空间卷积。Xception 单元由此

启发而生，但是稍有改动。Xception 做空间卷积核深度卷积的顺序有所不同，Xception 先对每一层进行通道分离的空间卷积，然后再做 1X1 卷积，并且在做通道分离的空间卷积时，去掉 Relu 操作。Xception 单元结构如图 11（右）所示。Xception 与 Inception 相比，在不增加网络复杂度的情况下，提升了模型性能。

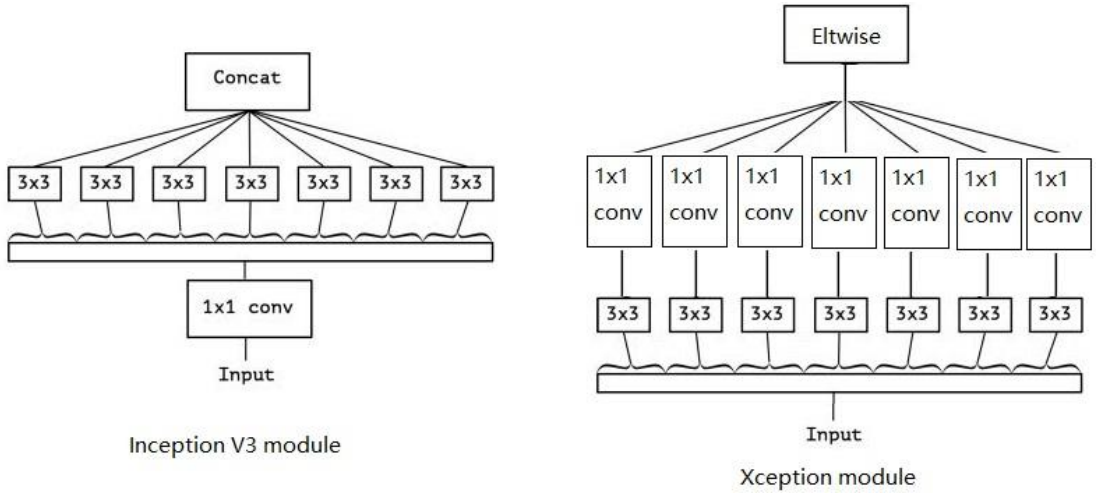


图 11 Xception 深度分离卷积网络结构

以上介绍了 keras 中较为优秀的几个预训练模型，其 Top1 准确率和 Top5 准确率如下表所示：

表 1 预训练模型准确率

| 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 |
| ResNet50 | 99MB | 0.759 | 0.929 | 25,636,712 | 168 |
| InceptionV3 | 92MB | 0.788 | 0.944 | 23,851,784 | 159 |

我们可以利用这些预训练模型，进行迁移学习，设计步骤如下：

1. 将图片输入 ResNet50 得到卷积层最后输出，做全局平均池化后作为图片的第一组特征。

2. 将图片输入 InceptionV3 得到卷积层最后输出，做全局平均池化后作为图片的第二组特征。

3. 将图片输入 Xception 模型得到卷积层最后输出，做全局平均池化后作为图片的第三组特征。

4. 将三组特征输入到神经网络，训练并对测试集进行预测，输出是狗的概率。

5. 神经网络只有输入输出两层，在输入层后添加 Dropout 以减少过拟合。

6. 在输出层采用 sigmoid 函数将输出值映射到 0-1 之间。

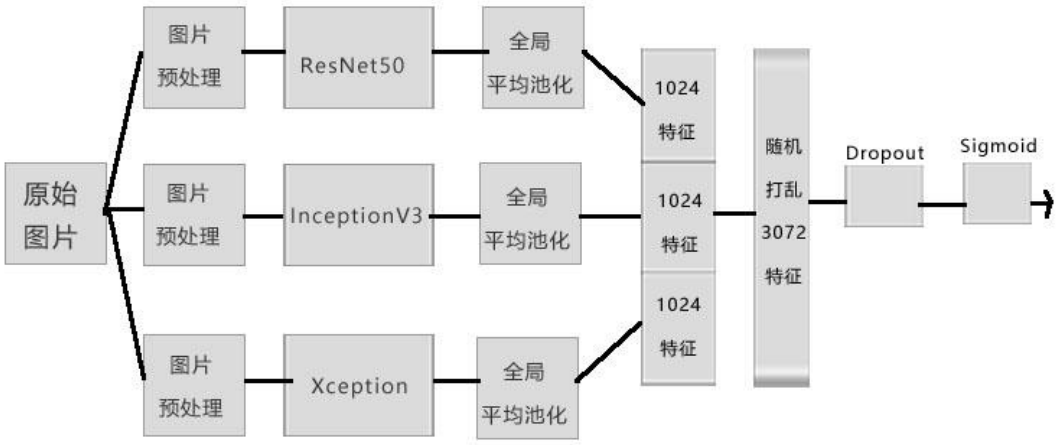


图 12 模型结构

### 基准模型

由于本项目源自一个 Kaggle 比赛，kaggle 上有一个比赛的排行榜，我们的目标是进入前 TOP100，因此我们可以选择第 100 名的模型作为基准模型，第 100 名的 LogLoss 成绩是 0.05629。

## III. 方法

### 数据预处理

我们的数据集主要需要两方面预处理，第一个是图片需要统一尺寸，由于我们采用迁移学习，图片会作为预训练模型的数据，而预训练模型都有自己不同的默认图片大小，为了获得较好的性能。我们选择将模型预处理成每个模型默认大小一致的大小，如 VGG19, ResNet50 默认输入

大小是(224,224)。InceptionV3,  Xception 默认输入图片大小是(299,299)。

另一个需要预处理的是将图片进行归一化，图片默认值是 0-255 的整形值，而预训练模型输入的是-1~1 的浮点值，因此需要对图片进行归一化。

此外我们对数据集的目录结构进行了微调，调整前数据的目录结构为

```
input -------train------- dog.0.jpg
        |          |------- cat.1.jpg
        |          |-------...
        |
        ----test -------- 0.jpg
                  |--------1.jpg
                  ---------...
```

为了方便使用 `ImageDataGenerator`，我们将训练数据中，猫狗图片分别移动到不同的分类文件夹。 同时 `test` 目录结构做同样的目录深度调整，调整后的目录结构为：

```
input -------train------dog------- dog.0.jpg
        |          |          |------- dog.3.jpg
        |          |          |------- ...
        |          |
        |          |
                   --------cat------- cat.1.jpg
        |                   |------- cat.5.jpg
        |                   |------- ...
        |
        -------- test -------test ------ 0.jpg
                              |--------1.jpg
                              ---------...
```

## 执行过程

1、解决内存问题

执行过程中遇到的第一个主要问题是内存问题，我尝试将数据通过 OpenCV 读入内存，然后对其进行归一化预处理。由于图片量很大，在训练时即使很小的 batch 也会内存错误。随后我修改了图片读取方式，采用 Generator， 只读取当前训练所需要的数据进入内存进行训练。

2、解决重复生成特征问题

我尝试了 4 中预训练模型生成特征，包括 VGG19，ResNet50，InceptionV3， Xception。由于模型较大，且图片较多，每次生成特征保存在内存中需要几十分钟。但是如果发生意外情况，这些导出的特征可能丢失，需要重新从模型生成，这样特别耗时。因此我采用将生成的特征保存到硬盘的方式，这样每次意外或者电脑重启后，只需要从硬盘读取已经生成的特征就可以了。

3、earlyStop

当模型训练到一定程度，再继续训练验证集的 Loss 就不再下降，这可能有两个原因，一个原因是学习速率过大，导致震荡，如图 13。另一种可能原因是已经开始过拟合，如图 14，无论哪种情况，继续训练下去都不应该继续训练，在过拟合的情况下，如果继续训练，模型表现甚至会变得更差。

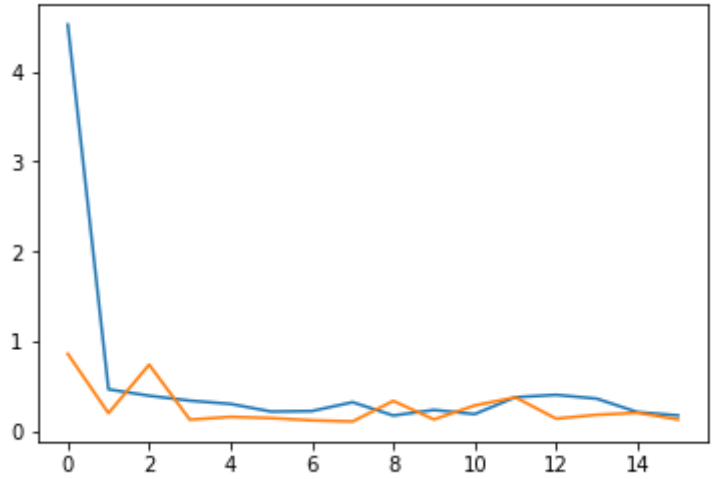


图 13 Adam loss 学习曲线 蓝色：训练集，橙色：验证集
lr=1

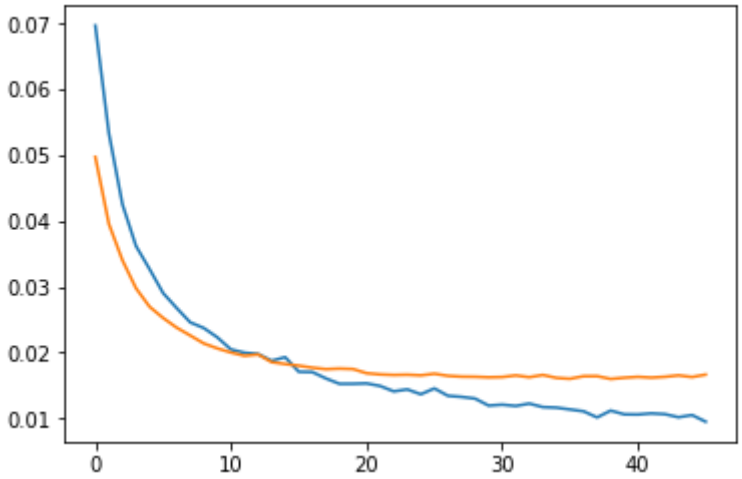


图 14 Adam loss 学习曲线 蓝色：训练集，橙色：验证集
lr=0.00007

## 完善

最开始我对 keras 中提供的主流预训练模型[10]，包括 VGG16[11]，ResNet50, InceptionV3, Xception 等都进行了特征提取并用全连接网络进行模型融合，得到的结果如图 15 所示，采用的是 Adam[12]优化器，优化器参数是 Keras 默认参数，从图中可以看出，学习速率过大，导致结果不收敛。

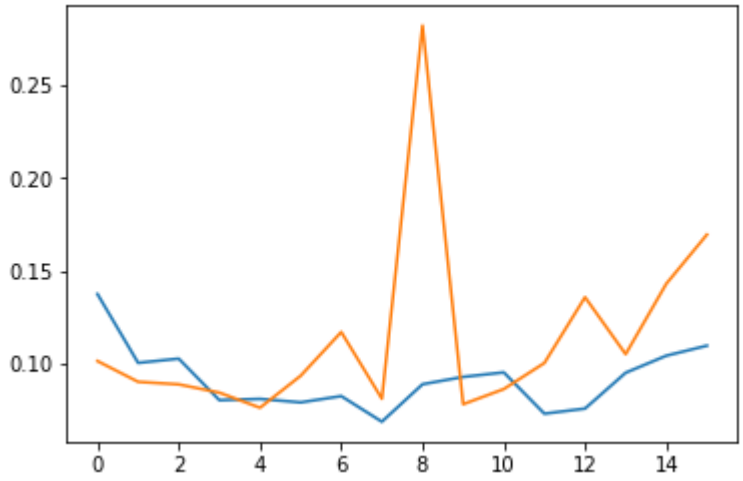


图 15 Adam loss 学习曲线 蓝色：训练集，橙色：验证集

通过降低学习速率，当学习速率在 0.000001 时，网络收敛的较好，此时需要的训练周期也相应较长。学习曲线如图 16 所示。最终验证集的正确率稳定在 0.9942

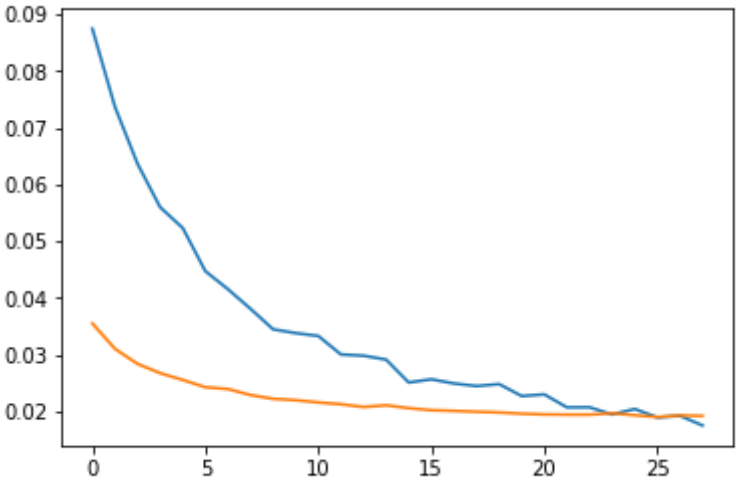


图 16 Adam loss 学习曲线 蓝色：训练集，橙色：验证集

虽然得到的成绩不错，但是学习训练较费时。我们在提取特征时，最后卷积的输出使用了全局平均池化，而全局平均池化可以作为全连接网络的替代。因此可以去掉全连接网络，直接让特征输入与输出相连接。修

改后重新调整学习速率使网络收敛（学习速率 0.00002），训练结果与全连接网络模型表现十分接近，但是训练时间显著减少。

学习曲线和正确率曲线如图 17 图 18 所示。此时正确率依然收敛在 0.9942

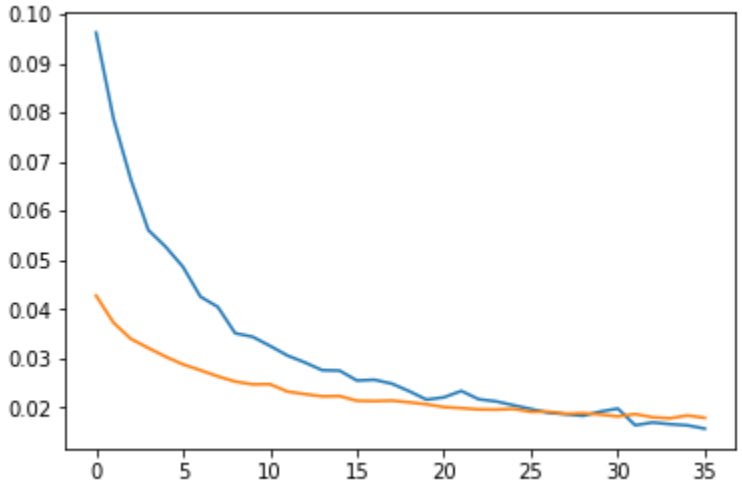


图 17 去除全连接后的 Adam loss 学习曲线 蓝色：训练集，橙色：验证集

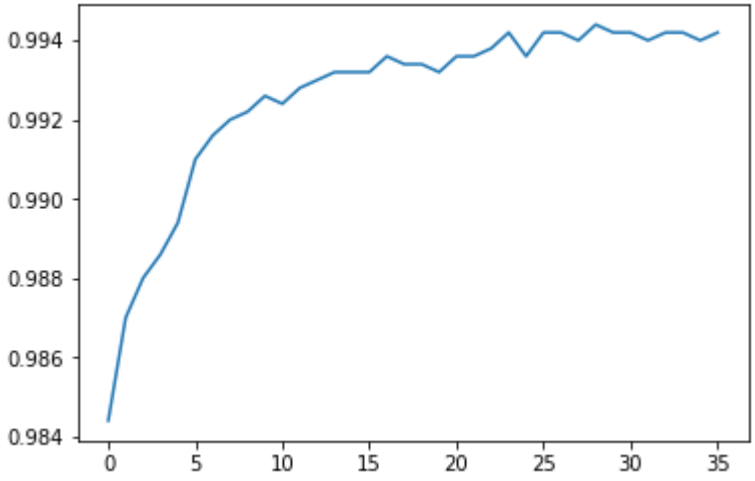


图 18 去除全连接后的验证集正确率曲线

我们使用了四个模型组合，然而试验中发现，四个模型组合模型表现并不是最好的，于是我们进行了模型组合测试。首先我们测试了单模型迁移学习的模型表现。

表 1 单个模型正确率

| | VGG16 | ResNet50 | Xception | InceptionV3 |
|---|---|---|---|---|
| 正确率 | 0.9814 | 0.9868 | 0.9942 | 0.9932 |

可以看出 Xception 单个模型表现最好，与之前我们四个模型的组合性能接近。四个模型组合性能没有超越 Xception，这可能是由于 VGG16 和 ResNet50 模型正确率太低，干扰了模型组合表现。

随后我们也测试了四个模型两辆组合的表现。

表 2-1 Xception 与其他模型组合正确率

| | VGG16 | ResNet50 | InceptionV3 |
|---|---|---|---|
| Xception | 0.9946 | 0.9938 | 0.9940 |

表 2-2 VGG16 与其他模型组合正确率

| | Xception | ResNet50 | InceptionV3 |
|---|---|---|---|
| VGG16 | 0.9946 | 0.9874 | 0.9944 |

表 2-3 VGG16 与其他模型组合正确率

| | Xception | VGG16 | InceptionV3 |
|---|---|---|---|
| ResNet50 | 0.9938 | 0.9874 | 0.9930 |

通过上述组合模型我们发现一个有趣的现象，VGG16 单模型表现最差，然而它和最好的模型融合表现却是最好的。这里一个可能的原因是，这个模型和 Xception 差异比较大，因此提取的特征互补性更强。

表 3 三模型融合正确率

| InceptionV3，VGG16，ResNet50 | VGG16，ResNet50，Xception | VGG16，InceptionV3，Xception | InceptionV3，Xception，ResNet50 |
|---|---|---|---|
| 0.9928 | 0.9930 | 0.9940 | 0.9950 |

最终我们选择 InceptionV3，Xception，ResNet50 三个模型进行组合，其模型融合正确率最高。

模型组合确定后，我们再调整了一下学习速率最后模型训练结果如下：

Train on 19973 samples, validate on 4994 samples

Epoch 1/64　loss: 0.1958 - acc: 0.9293 - val_loss: 0.0494 - val_acc: 0.9914

Epoch 2/64　loss: 0.0453 - acc: 0.9903 - val_loss: 0.0300 - val_acc: 0.9920

Epoch 3/64　loss: 0.0313 - acc: 0.9926 - val_loss: 0.0235 - val_acc: 0.9948

中间迭代省略...

Epoch 23/64　loss: 0.0096 - acc: 0.9972 - val_loss: 0.0130 - val_acc: 0.9964

Epoch 24/64　loss: 0.0090 - acc: 0.9969 - val_loss: 0.0132 - val_acc: 0.9964

Epoch 25/64　loss: 0.0081 - acc: 0.9976 - val_loss: 0.0132 - val_acc: 0.9964

Epoch 26/64　loss: 0.0087 - acc: 0.9972 - val_loss: 0.0132 - val_acc: 0.9962

模型在第 26 个 epoch 由于验证集三次没有下降而提前停止。

# IV. 结果

## 模型的评价与验证

针对比赛数据，对模型表现最好的评价方式莫过于提交到比赛看成绩。将最终模型对测试集进行预测，预测结果提交到 kaggle 比赛平台，最终成绩为 0.03857，排名 12 名，达到了我们预设的目标 Top 100。

## 合理性分析

回顾我们的基准模型 LogLoss 是 0.05629。而我们最终模型提交后得到的成绩是 0.03857. 比基准模型 LogLoss 下降了 32%。在 kaggle 排名可以拍到第 12 名，证明已经完成最初的目标。

# V. 项目结论

## 结果可视化

对最终模型学习过程进行可视化，Loss 收敛曲线和正确率曲线如图 19、图 20 所示。为了让图片后期细节更清楚，我省略了前面 4 个 epoch 的数据。

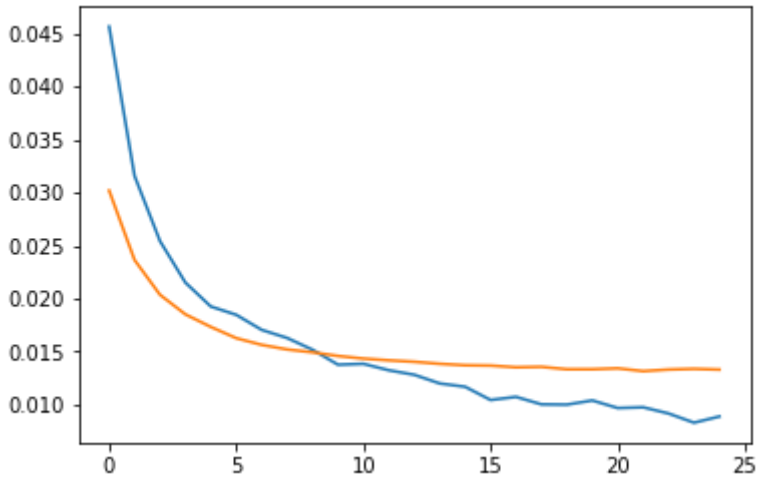


图 19 最终模型 Loss 收敛曲线

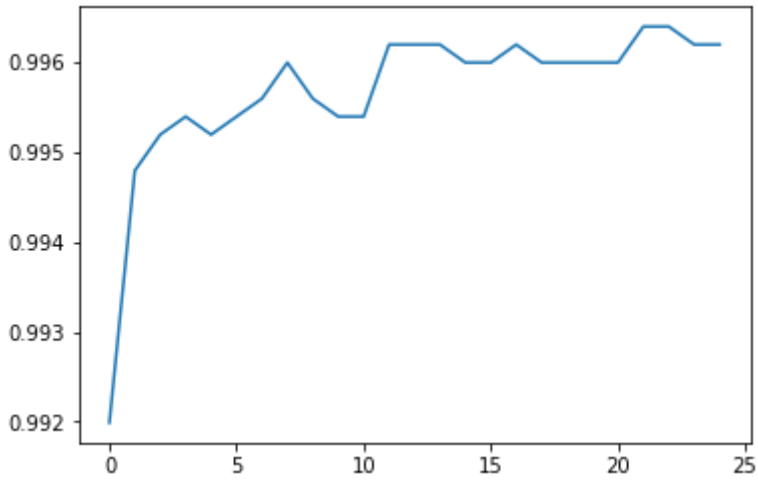


图 20 最终模型正确率上升曲线

从图 11 可以看出验证集 Loss 已经收敛平稳，训练集交叉熵已经比验证集明显低，表明已经有过拟合，但是由于验证集的交叉熵并没有增加，因此这种轻度的过拟合对模型是无害的。我在 epoch 20 停止训练，得到的模型在测试集上的评分明显差于目前最终的模型，这也证明了前述解读的正确性。

从图 20 可以看出模型正确率最终收敛在 0.996 上方小幅震荡。

## 对项目的思考

本文选取了 keras 几个针对 ImageNet 的预训练模型进行迁移学习，按照预训练模型要求对图片进行了归一化，然后，我们将图片输入不同的预训练模型导出特征。我们将特征全局平均池化后输入两层的神经网络，最后得到单个输出，代表图片是狗的概率。

在模型组合上我们选择了四个预训练模型，并对其单模型表现，两两组合表现、三模型组合表现，已及四个模型全组合的表现进行测试，最终选取了 InceptionV3，Xception，ResNet50 三个模型进行融合，得到了可以在 kaggle 排行榜上排到 11 名的成绩，这个成绩超过了我最初的期望。

再模型组合测试时我发现，模型两两组合时，单模型表现较差的并不一定和其他模型组合后也会较差，VGG16 单模型表现较差，但是它和 Xception 组合后却是两模型组合里表现最好的。 然而三模型组合时，单模型表现较好的三个模型组合起来效果就更好。

整个过程中最耗时的是调参，其实这里主要调的是优化器和学习速率，因为优化器不同所需学习速率也不相同，所以针对每个优化器都有一个学习速率的调参过程。在寻找最佳学习速率时，我是通过先给予较大的学习速率，然后逐渐降低学习速率的数量级，找到合适的学习速率数量级，最后再细化最佳学习速率，一个好的学习速率，可以使学习快速收敛，并且最后震荡幅度在可接受范围内。

## 需要作出的改进

关于本项目，如果想要得到更好的模型，还有两个可能改进的方向，本文尚未做尝试。

1. 模型训练完毕后，可以对整个训练数据集进行预测，将预测错误的图像单独取出来，做数据增强，然后将数据增强后的数据添加到原始数据集，再进行训练得到第二个模型。然后重复上述步骤得到第三个模型，最后用三个模型 一起投票得出最终是狗的概率。

2. 本文是直接使用的预训练模型的网络结构和权重，由于预训练模型权重是针对 Image Net 数据集训练的，如果我们针对本项目数据集重新训练权重或者针对最后几层网络进行 Fine tune。应该可以得到更好的效果。

---

## 参考文献


[1] PAUL VIOLA, MICHAEL J. JONES Robust Real-Time Face Detection, International Journal of Computer Vision 57(2), 137–154, 2004
[2] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.
[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012
[4] Kaggle DogVCat Competation: http://www.kaggle.com/c/dogs-vs-cats
[5] Kaggle DogVCat leaderboard: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/leaderboard
[6] Kaggle DogVCat evaluation: https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition#evaluation

[7] He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015).
[8] Mordvintsev, Alexander; Olah, Christopher; Tyka, Mike (2015). "Inceptionism: Going Deeper into Neural Networks". Google Research
[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.
[10] keras 中文文档 https://keras-cn.readthedocs.io/en/latest/other/application/
[11] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014)
[12] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014)