猫狗大战

I 问题的定义

项目概述

最近，在大规模图像识别中，卷积神经网络发挥了重大作用。这主要归功于大的图像数据库，比如 ImageNet，以及计算机性能的提升，比如 GPU。尤其是，ImageNet Large-Scale Visual Recognition Challenge(ILSVRC)挑战赛对深度视觉识别框架的发展起到了很大的推动作用。从较浅的高维度神经网络到深度卷积神经网络，ILSVRC 挑战赛孵化出了好几代大规模图像分类系统。

自从 Alexnet 卷积神经网络问世后，其优秀的图像识别能力使得卷积神经网络获得了很大的关注。随后 VGG 网络、Googlenet 网络、Resnet 等各种卷积神经网络应运而生。随着卷积神经网络的不断发展，目前图像分类已经发展的相当成熟。图像分类的发展促进了机器视觉其他方面的发展，比如图像检测，图像分割等。图像分类模型性能的不断提高，是机器视觉不断发展的标志。Kaggle 竞赛对图像分类、图像分割等各种技术的发展起着推动性作用。本文来探索 Kaggle 竞赛中一个简单的图像分类任务。

问题陈述

本文对猫和狗的图像进行分类。使用的数据集是 kaggle 竞赛上的猫狗数据集。其下载链接为(https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data)。数据集中包含各种各样的猫狗图像，训练集有 25000 张，其中猫狗分别占 12500 张，分别放在相应的类别文件夹中。测试集包含 12500 张图像。数据集具有多样性，拥有不同颜色，不同大小，不同角度，不同数量，不同场景，不同姿态等特征。图像的分辨率也是大小不一的，并且图像中或者有猫，或者有狗，但不会同时有猫和狗。要找到一种模型，在这些数据集上训练之后，使得模型具有分辨猫图像和狗图像的能力。

评价指标

因为该任务是要对猫和狗进行分类，所以这是一个二分类任务。在分类任务中，有两种评价指标。一种是准确率，另一种是精确度和召回率。准确率适合对称性分类任务，精确度和召回率适合不对称性分类任务。因为本文的猫狗数据集中猫和狗的图片比例相同，因此我选择分类准确率作为评价指标。

II 分析

数据探索

猫狗数据集分为训练集和测试集。训练集包含 25000 张图像，其中猫狗分别占 12500 张。测试集包含 12500 张图像。相比之下，ILSVRC 竞赛的数据集包含 120 万张图像，1000 个类别，平均每个类别包含 1200 张图像。对比猫狗数据集与 ILSVRC 竞赛的数据集，虽然总的图

像数差距很大，但如果只比较单个类别的图像数量，猫狗数据集比 ILSVRC 数据集的单个类别数高 10 倍。可以说，相比之下，猫狗数据集中单个类别的数据量很丰富。

因为测试集是不含标签的，因此为了验证训练效果，我从训练集随机挑选出 2500 张图像作为验证集，其中猫狗各占 1250 张。此时训练集包含 22500 张猫狗图像，验证集包含 2500 张猫狗图像。

探索性可视化

对猫狗数据集中的训练集和验证集的分析图像如图 1、图 2 所示。从这两幅图中可以看出，在训练集和验证集中猫和狗的图像分别占有的比例相同。因为验证集是从最初的训练集中分离出来的，因此训练集和验证集的图像服从同一分布。

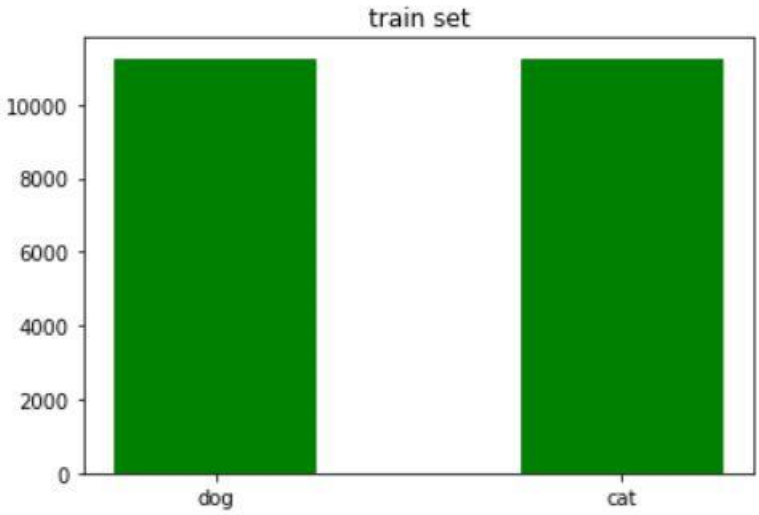

图 1 训练集可视化分析

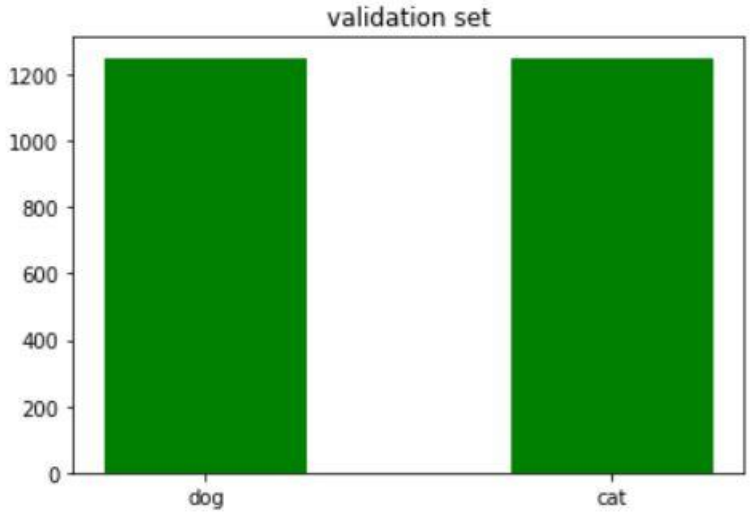

图 2 验证集可视化分析

算法和技术

VGG 论文研究的内容是在传统的 Alexnet 卷积网络框架(Krizhevsky at al,2012)基础

上，研究增加深度对网络性能的影响。他们发现深度模型不仅在 ILSVRC 数据集中表现好，而且在其他数据集表现也很好。该论文得出结论，随着网络深度的增加，计算机计算能力的提高，一定可以获得更好的性能。因此，我选择沿着网络深度这条线进行探索，不断增加网络的深度，探索在猫狗数据集上表现好的模型。

Pytorch 是一个基于 Python 的可续计算包，提供两个高级功能：第一，具有强大的 GPU 加速的张量计算（如 NumPy）。第二，包含自动求导系统的的深度神经网络。因此我选择了使用 pytorch 来进行实验。

基准模型

我选择传统的卷积网络框架 Alexnet 网络作为基准模型。该模型包含 5 个卷积层，3 个全连接层，60 million 参数量。该模型是卷积神经网络步入图像分类的第一步。

在 torchvision 库的 models 模块中，包含有已训练好的 alexnet 模型，VGG 模型。我在最初的计划是，用猫狗数据集依次分别来训练以下几个模型：未训练的 alexnet 模型，已训练好的 alexnet 模型，未训练的 VGG 模型，已训练的 VGG 模型。训练结束后，对各个模型的表现进行对比。

另外，对于猫狗数据集，人的表现可以到达 0%的错误率。因此 0%的错误率就是训练模型的目标。

III 方法

数据预处理

猫狗数据集包含各种分辨率图像，可网络需要输入的图像是固定的维度。因为 Alexnet、VGG 网络要求输入图像尺寸为 224 X 224 X 3，所以我们对图像进行下采样并剪切出一个固定分辨率为 224 X 224 的三通道 RGB 图像。在 torchvision 包的 transforms 模块中包含多中处理图像的方法，比如裁剪、翻转、缩放、标准化等。要将不同分辨率的图像转换成 224 X 224 的输入图像，首先要对图像进行缩放操作 transforms.Resize(256)，然后对缩放后的图像从中心裁剪出 224 X 224 的图像块 transforms.CenterCrop(224)。最后对图像标准化 transforms.Normalize()。即把训练集图像的每个像素减去所有训练图像的平均值，再除以标准差。其中的参数 mean 和 std 是从训练集 25000 张图像中随机选取 2500 张计算得到的近似值。除此之外，没有对训练集做其他预处理。

执行过程

第一，选择模型。

我预先计划在未训练的 Alexnet 模型、预训练的 Alexnet 模型、未训练的 VGG16 模型、预训练的 VGG16 模型上分别进行训练，并对比训练结果。Alexnet 模型是一个 8 层的卷积神经网络，VGG 是一个 16 层的卷积神经网络。可以提前预料的是，VGG 网络比 Alexnet 网络表现好，因为 VGG 网络的深度是 Alexnet 网络的两倍。但是因为 VGG 网络深度是 Alexnet 的两倍大，且 VGG 网络的参数量也是 Alexnet 的两倍大，因此训练速度会慢很多。我之所以选择尝试在 Alexnet 网络、VGG 网络上训练，是因为我想知道不同深度的卷积神经网络在该猫狗分类任务中表现如何。

第二，选择损失函数和优化器。

损失函数主要有两大类，回归损失和分类损失。回归损失主要包括均方误差损失函数，平均绝对误差损失函数等。分类损失主要包括均方误差损失函数，交叉熵损失函数，负对数似然损失函数等。本文是一个二分类问题，我选择分类损失中的交叉熵损失函数作为模型的损失函数。

在机器学习和深度学习中，除了常见的 SGD 优化器，还有 Adagrad，RMSProp，Adam 等各种优化器。Adagrad，RMSProp，Adam 等这些优化器都是在 SGD 的基础上一步步进行改进的结果。他们解决了 SGD 固定学习率的问题，使得不同的参数对应不同的学习率，并且在训练过程中学习率是一直变化的。

对于猫狗图像分类任务，我选择的优化器是 Adam 和 SGD。因为 Adam 训练速度比 SGD 快，因此最开始的训练的时候我选择 Adam 作为优化器，等到训练集损失值趋于平缓，再使用 SGD 作为优化器继续训练。SGD 虽然比 Adam 训练速度慢，但 SGD 训练起来更稳定，训练集损失值波动性小。

第三，超参数的选择

SGD 的学习率最初选择了 1e-4，之后以 10 的倍数递减。

因为笔记本显存有限的缘故，把 Batch_size 设为 64。

训练模型

1，在未训练的 Alexnet 模型上训练时，网络参数都是随机初始化的。我的网络结构如图 3 所示：在全连接层没有包含 dropout 层。用该网络进行训练，5 个 epoch 后，网络在验证集上的准确率达到 88%后很难再提升，再继续训练就会过拟合。

```
myNet(
  (features): Sequential(
    (0): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4), padding=(5, 5))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=9216, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=4096, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

图 3 alexnet 网络训练（不含 dropout 层）

然后，我分别在 alexnet 网络的第一个和第二个全连接层前加上了 dropout 层，参数 p=0.5。网络结构如图 4 所示。再次进行训练。5 个 epoch 后，网络在验证集上的准确率达到了 90%。相比不含 dropout 层的网络提升了 2%。

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

图 4 alexnet 网络结构

最初训练的时候没有对输入图像进行标准化，因此为了进一步提升 Alexnet 网络的性能，我对输入网络的训练图像进行标准化，使得输入图像的像素分布的中心置零。即 transforms.Normalize(mean=[0.488,0.455,0.417],std=[0.230,0.225,0.225])。训练 8 个 epoch 后网络在验证集上的准确率达到 91%，相比又提升了 1%。

2，用 pytorch 库 models 模块中预训练的 Alexnet 网络进行迁移学习。训练结果如图 5 所示。最后验证集的准确率可达到 95%，但有较严重的过拟合。加入 L2 正则化后，可降低过拟合，但是验证集的准确率达到 95%后较难增长。表现最好的节点是第 2 的 epoch，此时没有过拟合，验证集准确率为 95.68%。

	训练集准确率	验证集准确率
1 epoch	93.85%	95.36%
2 epoch	95.85%	95.68%
3 epoch	96.44%	95.84%
4 epoch	96.76%	96.04%
5 epoch	97.29%	96.32%

图 5　Alexnet 迁移学习训练结果

3，用随机初始化的 VGG 模型训练时，发现需要很长时间才能训练好，因此放弃。

4，接下来用预训练的 VGG16 网络进行迁移学习。VGG16 网络含有 16 层，138 Million 参数量，其模型结构如图 6 所示。由于 VGG 网络较大，在自己的笔记本上无法展开训练，因此选择在云端进行训练。我选择了阿里云。训练之后的结果如图 7 所示。

	训练集准确率	验证集准确率
1 epoch	97%	97%
2 epoch	97%	98%
3 epoch	98%	98%
4 epoch	98%	98%
5 epoch	98%	98%

图 7 VGG-16 迁移学习训练结果

	训练集准确率	验证集准确率
1 epoch	90.28%	96.36%
2 epoch	96.28%	97.16%
3 epoch	96.84%	97.52%
4 epoch	97.04%	97.64%
5 epoch	97.20%	97.72%

图 8 resnet-18 迁移学习训练结果

	训练集准确率	验证集准确率
1 epoch	94.67%	97.04%
2 epoch	97.31%	97.72%
3 epoch	97.44%	98.04%
4 epoch	97.70%	98.32%
5 epoch	97.81%	98.04%

图 9 resnet-50 迁移学习训练结果

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

图 6 VGG16 网络结构

完善

VGG16 网络的准确率达到 98%之后收敛速度大大降低，相比人的水平仍然有 2%的差距。这 2%的错误率足以使得 kaggle 得分排在一千名以后。距离 0%的错误率还有很长一段路。

为了提高准确率，要选择更深的模型进行训练。但模型越深，训练难度越大。残差网络解决了深度网络训练慢，训练难的问题。我用 Resnet-18 网络和 Resnet-50 网络分别进行迁移学习，其训练结果如图 8，图 9 所示。从训练结果可以看出，即使深度为 50 层的 Resnet 网络仍然有 2%左右的错误率。距离 0%的人类水平还差很远。

最后，我参考了知乎上优达学城账户发布的一篇文章。把已经训练好的不同模型的特征向量结合在一起，后面加上自己创建的全连接层。训练模型时只对自己创建的全连接层进行训练。文中选择了 resnet50，Xception，Inception-V3 这三个模型进行结合。因为这三个模型都很大，为了节省计算量和训练时间，预先把训练集图像转换成各个网络的特征向量，并保存下来。也就是说，一幅图像映射出三个不同的特征向量，之后将这三个不同的特征向

量结合成一个特征向量。结合后的特征向量作为自己创建的全连接网络的训练集进行训练。这种方法降低了训练网络时对电脑配置的要求，而且大大提高了训练速度。其训练结果如图10所示。验证集的错误率在0.4%左右。将结果上传至kaggle竞赛，loss得分为0.041，排名在前2%。图11为kaggle竞赛得分截图。可见这种多模型融合的方法表现很优秀。

	训练集准确率	验证集准确率
1 epoch	97.69%	99.40%
2 epoch	99.17%	99.60%
3 epoch	99.37%	99.64%
4 epoch	99.45%	99.60%
5 epoch	99.49%	99.60%

图10 多模型融合训练结果

图11 多模型融合kaggle得分

IV 结果

模型的评价与验证

对于猫狗数据集，人的表现可以到达0%的错误率。因此训练模型的目标就是0%的错误率。对比Alexnet模型与VGG16模型训练结果，Alexnet模型包含8层，60 million的参数量，该模型的错误率在5%左右。VGG16网络包含16层，138 million参数量，层数和参数量都是alexnet网络的两倍，该模型的错误率在2%左右。VGG16网络训练起来比Alexnet网络慢,如果有足够的计算资源和训练时间,VGG网络和alexnet网络都能获得更低的错误率。

VGG网络、Alexnet网络以及Resnet网络都获得了较低的错误率，但距离0%的错误率还有一段距离。最后把Resnet50，Xception，Inception-V3这三个模型结合在一起后，发现可以获得0.6%的错误率，可见这种多模型融合的方法表现很优秀。

V 项目结论

深度模型在本文的猫狗图像分类任务中表现很好，准确度都超过了98%。只是要想达到100%的准确度,用多模型融合的方法可以更快的实现我们的目的。只是多个模型结合在一起，使得计算量成倍增加，不适合在配置较低的硬件上实时预测。

对项目的思考

对猫狗图像分类这样一个项目，听起来很简单的事情，发现训练起来却不简单。从8层的Alexnet模型获得了92%的准确率，到50层的Resnet-50模型获得了98%的准确率。从这之间的差距可以看出，如果要获得100%的准确率，也许需要创建几百层的网络。在kaggle网站上有人用Resnet-101进行迁移学习，获得了0.05的优秀得分。这个分数相当接近多模

型融合的结果。如果要训练几百层的网络，需要计算机拥有更快的计算速度。

需要做出的改进

由于硬件资源有限，网络的训练时间不长。如果增大训练时间，我认为可以进一步提高各个模型的准确率。多模型融合这种方法虽然可以达到较高的准确率，却需要更多的计算资源。我认为接下来的任务是探索不仅准确率高而且计算效率也高的模型。

十分感谢优达学城，在这个项目中我学到了很多东西。