浙江大學

基于模糊视觉的猫狗二分类

人工智能与机器学习

小组研究报告

刘一诺　3190102527
俞一诺　3190100994

基于模糊视觉的猫狗二分类

俞一诺[1]， 刘一诺[2]

[1]浙江大学控制科学与工程学院，[2]浙江大学心理与行为科学系

摘要: 摘下近视眼镜，人类依然能对模糊图像进行识别与分类。然而，机器能否像人一样拥有模糊视觉是图像分类中研究较少的问题。过往研究采用在模糊图像上进行训练的方式，证明了这种输入的有效性，但实际生活中人类不止拥有模糊图像的输入，更有清晰视觉的经验。因此，利用清晰图像的预训练也是该任务中应当考虑的因素。在本研究中，我们利用卷积神经网络中AlexNet和VGG的网络架构，完成猫狗二分类任务。我们采用了两种不同的训练方式，即是否在清晰图像上进行预训练，结果发现经过预训练的模型能够得到更高的准确率和更优的绩效表现，并且在这种训练方式上，网络结构更深的VGG略优于AlexNet，从而证明了经验与网络结构对模糊图像分类的作用。通过对AlexNet的卷积层进行可视化，我们还比较了人工神经网络模型与人类大脑视觉系统，发现只有经过大样本数据集训练后的网络才会表现出与视觉系统类似的层级特征，证明了视觉系统的复杂性与大样本数据集的重要性。

关键词: 模糊分类; AlexNet; VGG; 特征层; 视觉系统

Classification of Cats and Dogs Based on Fuzzy Vision

Yinuo Yu[1], Yinuo Liu[2]

[1]*College of Control Science and Engineering, Zhejiang Univeristy*

[2]*Department of Psychology and Behavioral Sciences, Zhejiang Univeristy*

Abstract: Humans can still recognize and classify blurred images after taking off nearsighted glasses. However, whether a machine has such fuzzy vision like a human is a less studied problem in image classification and computer vision. Previous studies have used the method of training on blurred images to prove the effectiveness of blurred input, while in real life humans have not only the input of blurred images but also clear vision experience. Therefore, pre-training with clear images is also a factor that should be considered in this task. In this research, we used the architecture of convolutional neural network AlexNet and VGG to complete the cat and dog classification task. We used two different training methods, that is, whether to pre-train on clear images or not, and found that the pre-trained model can get higher accuracy and better performance. For pre-trained model, the deeper network VGG performs slightly better than AlexNet. These results prove the effect of experience and network structure on blurred image classification. What is more, by visualizing the feature filters and convolutional layers of AlexNet, we also compared the artificial neural network model with the human brain visual system, and found that only the network trained with a large sample dataset can show hierarchical features similar to the visual system, which proves the complexity of visual system as well as the importance of large sample datasets.

Key Words: Fuzzy classification; AlexNet; VGG16; Feature layer; Visual system

1 引言

1.1 选题背景

近年来，机器学习领域中的深度学习越发热门，它是学习样本数据内在的规律与层级表示，力图让机器像人一样具有分析学习能力，目前已经在语音和图像识别、自然语言处理、数据挖掘等多个方面得到了广泛应用并取得了很多成果。

人工神经网络 (artificial neural network, ANN) 是深度学习中非常重要的元素，这种运算模型由大量节点相互连接构成。它最广泛的用途是进行分类，在大多数情况下，能在接受外界信息的时候改变内部的连接结构，更好地对输入和输出间复杂的关系进行建模或探索数据的模式。

人工神经网络的构筑理念是由生物大脑神经网络的运作方式启发的，其节点模拟了大脑的神经元，节点间连接模拟神经元间的突触，层级结构类似大脑皮层的组织方式。不仅在结构上有这样类似的组成，研究者也已经通过实验发现，经过调参、训练等方法使这些网络表现出优良性能后，深度人工神经网络的特征表征与在灵长类动物大脑中通过实验测量的内部神经表征非常相似(Yamins et al., 2013, 2014)。因此，随着深度人工神经网络的不断发展，或许这类模型在功能和机制上都能与大脑更为相似。

目前，这类模型已经在人工智能的很多领域中得到广泛应用，例如计算机视觉。计算机视觉是一门研究如何使机器“看”的学科，其中的经典问题包括图像分类、图像定位、图像恢复等。这些对于人类而言很简单的问题，对于机器来说却不容易，需要开发有效的模型与算法进行训练。在图像分类中，大量模型算法如支持向量机、迁移学习、K近邻算法等，但最为常用的仍是神经网络，通常采用卷积层提取特征，然后在全连接层上进行分类。目前提出的经典模型如LeNet、AlexNet、GoogleNet、VGGNet、ResNet等，这些模型极大地推动了计算机视觉中图像分类任务的发展。

然而，大量关于图像分类任务的研究所基于的数据集都是清晰图像，在实际生活中，却不免面临图像模糊失真问题。虽然人依然能在一定范围上识别模糊物体并进行分类，但这种失真却很可能导致分类器的性能显著下降。关于机器是否能对模糊图像同样进行精确分类，目前并没有广泛的研究。

基于此，我们自然想问，有没有一种训练方法，能够使机器像人类一样，在“看到”模糊图像时，依然能够进行准确分类？在本实验中，我们比较了在只接受模糊图像的训练、在接受清晰图像的预训练后再接受模糊图像的训练这两种条件下，机器模型对模糊图像识别的绩效表现，从而证明在模糊图像分类中引入预训练的有效性；我们也比较了不同模型结构对任务结果的影响，并通过观察模型所提取的特征及任务绩效，比较人类大脑与神经网络在视觉领域的异同，试图为计算机视觉在模糊图像分类上的发展提供一些思路。

1.2 研究意义

在理论层面，目前对于模糊图像分类问题的研究并不多，且多数关注对模糊图像本身模糊类型的分类 (如高斯模糊、散焦模糊、运动模糊等) 上，我们的研究可以填补这方面的空白。同时，通过对模型和人类大脑在特征提取、信息传输、任务绩效等方面的比较，不仅可以为后续改进模型提供参考，也能为视觉领域神经科学的研究提供新的灵感思路。

在实践层面，模糊图像识别对于图像处理领域中的失真图像恢复具有重要意义。生活中，虽然存在大量清晰图像，但同样有种种因素会导致图像模糊失真，例如拍摄时相机抖动、雾天可见度降低、聚焦失败等，使得人们常常需要将模糊图像清晰化。如果能够实现模糊图像的精确识别，可以在图像增强、恢复时起指导与帮助作用。

1.3 文献综述

1.3.1 经典卷积神经网络

图像分类任务中，卷积神经网络有非常广泛的应用，我们计划采用这一类模型进行训练，因此简单考察了经典的网络模型。

Yann LeCun (1998) 提出的LeNet是早期经典的卷积神经网络，主要用于识别手写数字图像，分为卷积层块和全连接层块，其中卷积层块里卷积层用于识别图像中的空间模式，后接的最大池化层用于降低位置敏感性。LeNet可以在早期的小数据集上取得不错的结果，但在更大的真实数据集上未能成功适用。

2012年开发的AlexNet (Alex Krizhevsky, 2012) 使用了八层神经网络，以很大的优势赢得了ImageNet 2012图像识别挑战赛，其中第一次使用了ReLu激活函数，使用了随机失活、随机梯度下降，并大量使用了数据扩充技术。其优秀的表现使卷积神经网络的研究在当时成为了热门。

VGGNet是2014年的亚军，但具有非常广泛的应用。这是一个系列的网络，有多种结果，如VGG11、VGG16、VGG19等，创新在于通过重复使用简单的基础块来构建深度模型，即采用堆积的小卷积核来取代大的卷积核，在保证具有相同感受野的同时，增加网络深度，提升网络效果。

但是深层网络常常难以训练，2015年的ResNet在解决这个问题上做出了很大贡献。这个模型引入了残差模块，这一较短路径的引入可以使前面的输入直接连接到输出。而不同于通过增大网络深度来提升训练效果，GoogleNet从改变基础卷积块入手，在一个卷积块中包含不同窗口形状的卷积层和最大池化层，并行抽取信息，然后进行联结。

随着技术的发展，经典CNN准确率不断提高，同时模型的复杂度也在提高。从最初由神经科学中引发的建模灵感，也逐渐转向了数据驱动。数据驱动下改进的模型能在任务上有优良的表现，但神经科学能否从中获取新研究的启发，还需要对两者进行比较。

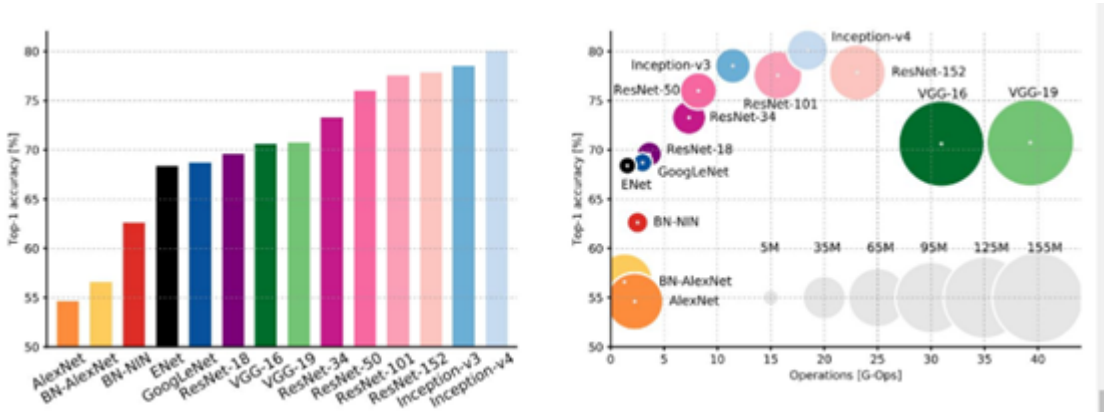

图1 近些年卷积神经网络的发展

1.3.2 人工神经网络与视觉系统

图像分类作为计算机的视觉问题，在某种程度上，可以与人类的视觉系统进行一定比较。对人类视觉系统的研究历时漫长，不仅大大促进了人们对大脑工作机制的理解，其中的一些发现也在神经网络中得到了迁移应用。

早期Hubel和Wiesel (1959) 研究猫的单个视觉皮层神经元的感受野时，发现初级视觉皮层的神经元具有各自特定的感受野，只对其感受野内的刺激进行反应。感受野反映的是视觉中枢的空间编码规律，而在卷积神经网络中，感受野的概念被迁移为神经网络中特征图上一个点对应输入图上的区域，也就是卷积核的大小。

对于感受野的研究还发现，有些神经元细胞在接受信号传入时表现为放电增加，有些则相反。类似的，神经元间的突触连接也存在兴奋性连接与抑制性连接。这些说明在神经网络的权重更新中，无需规定权重为正。

和大脑皮层的分层类似，视觉皮层的信号传输也是分层进行的。视觉信号从外侧膝状体到初级视觉皮层，到次级视觉皮层，最后到高级认知区域，会经历一系列信号的传输。研究发现，低级皮层所表征的都是简单特征如颜色、朝向，而越到高级区域，在整合这些输出的简单特征外，皮层所表征的特征越是复杂，如整个图像、人脸等 (Maunsell & Newsome, 1987; Vogels and Orban, 1996; Hochstein & Ahissar, 2002). 在卷积神经网络中这也有所对应，随着网络深度的增加，每个卷积层所提取的特征会更具有代表性。

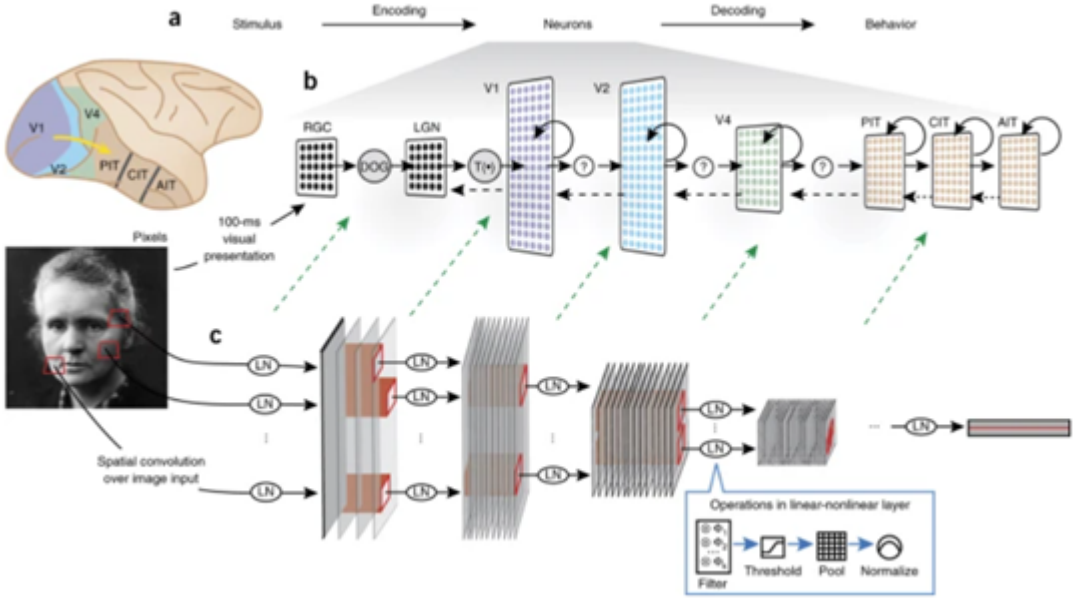

图2 人工神经网络和大脑视觉系统的类比

1.3.3 模糊图像分类

计算机视觉中关于模糊图像问题的研究大多关注对图像本身模糊类型的划分，例如Wang等人 (2021) 利用卷积神经网络和监督学习模型，实现了对散焦模糊、高斯模糊、雾霾模糊和运动模糊的识别。

而对模糊图像本身进行识别与分类的研究，最初多采用图像增强结合神经网络的方式。例如李光明等人 (2018) 利用多尺度形态学增强图像的边缘轮廓信息，然后再输入到网络中学习特征并分类。

与我们的研究更为类似的是Haritosh等人 (2019) 的工作。他们的训练思路来源于发展心理学对婴儿视觉的研究，即人的视觉系统相比于听觉、嗅觉等其他感官系统而言，进化时间相对更晚，新生儿的视力非常微弱，他们看世界上的物体都是模糊失真的。因此，他们模拟了婴儿期存在这类刺激输入的训练，先把模糊图像传给神经网络进行特征提取，提取的特征再传到两层人工神经网络模型进行分类。他们分别采用了两种训练方法，第一种是使用不同级别的模糊图像来分别训练若干模型，然后逐一用清晰图像进行测试；另一种是基于人类视觉系统引发的想法，最初用最高程度的模糊图像训练，类似新生儿的视觉模糊，然后模糊程

度逐渐递减，类似孩子成长，这样迭代地进行训练。他们发现迭代训练效果更好。同样的，常荣 (2021) 的研究利用了高速公路团雾图像，利用AlexNet构建CNN—SVM模型，并在优化网络结构后对模糊图像直接训练，得到较好结果。

这类研究采用的是直接利用模糊图像进行训练，实质上是在机器学习的训练过程中更换了数据集。他们的研究证明了利用模糊图像进行训练的有效性，在此基础上，考虑到目前广泛应用的“预训练”模型，我们认为，先在清晰图像上预训练过的模型，在接受这类模糊图像训练后，或许能有更好的任务表现，并能更好地模拟人类成熟视觉系统接收模糊输入时的表现。

2 研究内容

2.1 问题描述

这是一个基于模糊视觉分类的问题。目前，大部分的视觉识别 (比如MO平台上的口罩佩戴识别) 都是用清晰的图片作为训练样本。但是，如果图片没有那么清晰（例如加了噪声）而变得模糊，就像近视的人摘掉了眼镜，或者走进了大雾之中，那么现有的神经网络能不能够正确地出识别分类?

我们从Kaggle上拿到了Asirra Dataset (Animal Species Image Recognition for Restricting Access)，这是一个用于猫狗二分类的数据集，里面包括了猫猫和狗狗的照片，其中有清晰的，也有模糊的。

我们计划用卷积神经网络(CNN)来完成这项工作，即完成猫和狗的二分类，并在模糊图片输入的情况之下仍然能够保持较高的准确率。我们希望通过寻找不同结构的CNN，并改善对于图片处理的方式，来获得一个最好的结果。

2.2 研究重点与方案

- 对模糊图像的分类准确率：我们将通过两种不同方式对模型进行训练，一种是直接利用模糊图像进行训练，然后分别在清晰和模糊图像上进行检测；另一种是先在清晰图像上进行预训练，然后再在模糊图像上训练，最后分别在清晰和模糊图像上检测。
- 比较CNN结构：我们将从经典模型中选择准确率与复杂度比较合适的两个模型，根据数据集对模型结构进行调整，然后分别进行训练，比较模型结果。
- 与视觉神经系统比对：我们将观察模型卷积层训练到的特征，来与视觉感受野与视觉层级结构进行比较分析。

3 算法描述

3.1 AlexNet

AlexNet是2012年ILSVRC 2012 (ImageNet Large Scale Visual Recognition Challenge) 竞赛的冠军网络，分类准确率由传统方法的 70%+提升到 80%+。它是由Hinton和他的学生Alex Krizhevsky设计的。

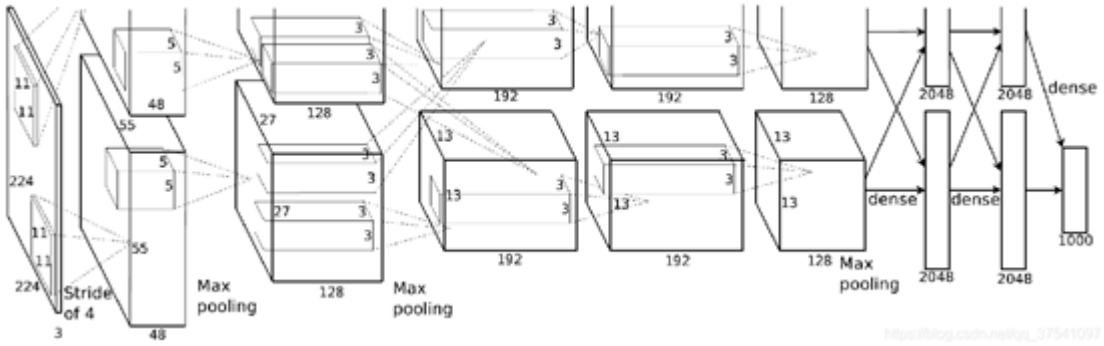

图3 AlexNet原始网络结构

在图中有上下两个部分是因为作者使用两块GPU进行并行训练，所以上下两个部分结构是一样的。该网络的亮点有：

- 首次利用 GPU 进行网络加速训练。
- 使用了 ReLU 激活函数，而不是传统的 Sigmoid 激活函数以及 Tanh 激活函数。
- 使用了 LRN 局部响应归一化。
- 在全连接层的前两层中使用了 Dropout方法按一定比例随机失活神经元，以减少过拟合。

经卷积或池化后的矩阵尺寸大小计算公式：$N = (W - F + 2P)/S + 1$，其中W是输入图片大小，F是卷积核或池化核的大小，P是padding的像素个数，S (stride)是步距。

AlexNet的网络结构总共层数为8层，其中5层卷积层，3层全连接层。我们根据本实验中数据的实际情况，对参数进行了调整更改，并采用了Batch归一化来缩短训练时间。最终构造的网络结构如下。`downscale` 指的是受制于计算能力，将通道数缩小的比例，在正式训练时 `downscale=2`，因此通道数仅为这里设计的一半。

- 第一层：卷积层1，输入为256 × 256 × 3的图像
- 卷积核的数量为64//downscale，大小为11 × 11；stride = 4，padding = 2
    - 激活函数采用ReLu并进行Batch的归一化
- 采用max_pooling，pool_size = (3, 3)，stride = 2，padding = 0
- 第二层：卷积层2，输入为上一层卷积的feature map
    - 卷积的个数为192//downscale，大小为5 × 5，padding = 2，stride = 1
    - 采用ReLu激活函数并进行Batch的归一化
    - 采用max_pooling，pool_size = (3, 3)，stride = 2，padding = 0。
- 第三层：卷积层3，输入为第二层的输出
    - 卷积核个数为384//downscale，大小3 × 3，padding = 1，stride = 1
    - ReLu激活函数并进行Batch的归一化
    - 没有池化层
- 第四层：卷积层4，输入为第三层的输出
    - 卷积核个数为256//downscale，大小3 × 3，padding = 1，stride = 1
    - ReLu激活函数并进行Batch的归一化
    - 没有池化层
- 第五层：卷积层5，输入为第四层的输出
    - 卷积核个数为256//downscale，大小3 × 3，padding = (1, 1)，stride = (1, 1)

    - ReLu激活函数并进行Batch的归一化
    - 采用max_pooling，pool_size = (3, 3)，stride = 2，padding = 0。
- 进行自适应大小的平均值池化，大小(6, 6)
- 第六层：全连接层
    - Dropout = 0.5，全连接层中输入$256 \times 6 \times 6//downscale$，输出4096//downscale，然后ReLu
- 第七层：全连接层
    - Dropout = 0.5，全连接层中输入4096//downscale，输出1024//downscale，然后ReLu
- 第八层：全连接层，并进行分类
    - 输入4096//downscale，输出2 (即二分类)

3.2 VGG-16

VGG网络是在2014年由牛津大学著名研究组VGG (Visual Geometry Group) 提出，斩获该年ImageNet竞赛中 Localization Task (定位任务)第一名和 Classification Task (分类任务) 第二名。原论文名称是 *Very Deep Convolutional Networks For Large-Scale Image Recognition*，在原论文中给出了一系列VGG模型的配置，下面这幅图是VGG-16模型的结构简图。

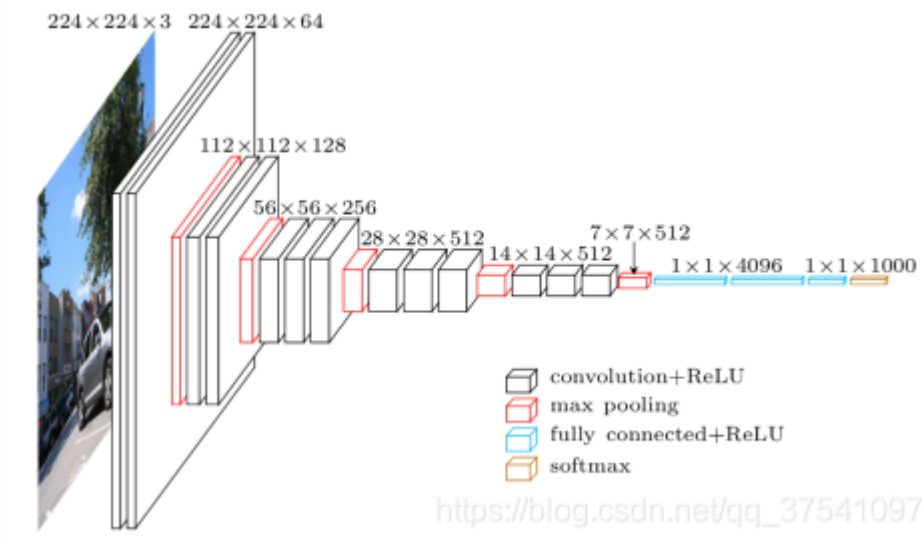

图4 VGG-16原始网络结构u

该网络中的亮点：通过堆叠多个3×3的卷积核来替代大尺度卷积核 (在拥有相同感受野的前提下能够减少所需参数)。论文中提到，可以通过堆叠两层的3×3卷积核替代一层5×5的卷积核，堆叠三层的3×3卷积核替代一层7×7卷积核。下面给出一个示例：使用7×7卷积核所需参数，与堆叠三个3×3卷积核所需参数(假设输入输出特征矩阵深度channel都为C)。

如果使用一层卷积核大小为7的卷积所需参数 (第一个C代表输入特征矩阵的channel，第二个C代表卷积核的个数也就是输出特征矩阵的深度)：$7 \times 7 \times C \times C = 49C^2$。如果使用三层卷积核大小为3的卷积所需参数：$3 \times 3 \times C \times C \times 3 = 27C^2$。经过对比明显使用3层大小为$3 \times 3$的卷积核比使用一层$7 \times 7$的卷积核参数更少。

我们在实验中主要修改了网络最初的输入和最后的输出，在最开始将256 × 256 × 3的输入进行了展平，最后输出`num_classes=2`，为二分任务。其他结构如图，没有进行较多更改。

4 模型训练

4.1 数据集

数据集来源于Asirra dataset，其中包含四类数据，即清晰的猫猫、清晰的狗狗、模糊的猫猫和模糊的狗狗。模糊的照片满足参数为5的高斯分布。

图5 数据集中的猫狗样例

4.2 训练过程

4.2.1 训练方法

训练集包括17953张清晰图像和17953张模糊图像，其中20%用做验证集；测试集包括2494张清晰图像和2494张模糊图像。

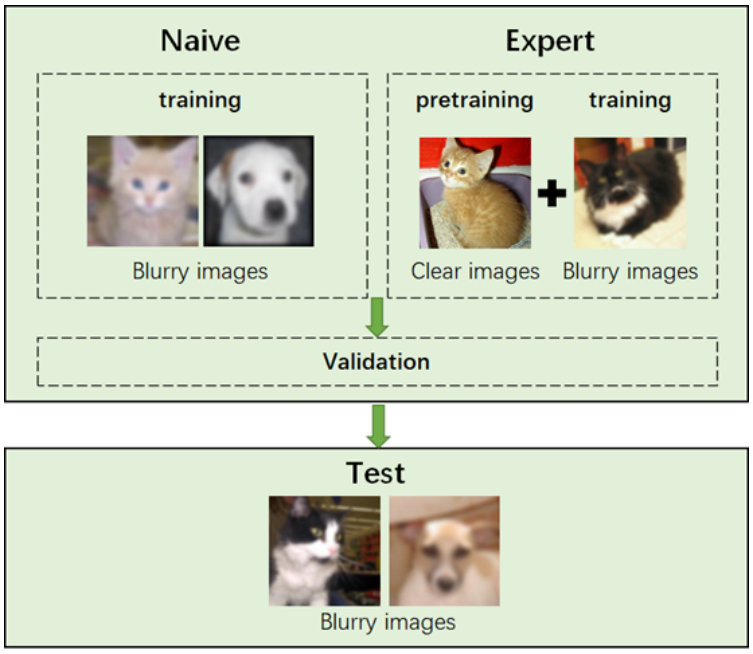

图6 研究中两种训练方法

我们采用了两种方法，分别训练了两个模型。

Naïve Model的数据集仅为Blurry images，数据集不做任何处理，即用最原始的方式进行训练。

Expert Model相比Naïve来说，多了一步预训练的过程，即用Clear images作为预训练数据集。在预训练的基础之上，再用Blurry images对模型进行训练。

4.2.2 训练代码

用上述AlexNet和VGG两个网络，分别进行Naïve和Expert两种方式，最终得到四个结果，形成对比。

这里以AlexNet的Expert Model为例，展示训练过程。

```
# 定义网络为ALEXNet，损失函数为交叉熵，优化器为Adam，初始学习率为3e-4
net = AlexNet(num_classes=2, downscale=2)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=3e-4)
net.to(DEVICE)
# 评估预训练之前的情况
evaluate(net, device=DEVICE, title="before training")
# 定义epoch：这里是Expert Training，pretraining epoch大于0，这里以1为例
num_pretraining_epochs = 1
num_training_epochs = 1
num_epochs = num_pretraining_epochs + num_training_epochs
# 保存网络权重
torch.save(net.state_dict(), "expert_before_training")
# 预训练过程 用clear images作为训练集（包含train和validation）
training_losses, validation_losses = train(num_pretraining_epochs,
                                           clear_train_batches,
                                           clear_val_batches,
                                           device=DEVICE)
# 评估预训练之后的情况
evaluate(net, device=DEVICE, title="after pretraining")
# 保存网络权重
torch.save(net.state_dict(), "expert_after_pretraining")
# 训练过程 用blurry images作为训练集（包含train和validation）
experienced_training_losses, experienced_validation_losses = train(num_training_epochs,
                                                                   noisy_train_batches,
                                                                   noisy_val_batches,
                                                                   training_losses=training_losses,
                                                                   validation_losses=validation_losses,
                                                                   device=DEVICE)
# 保存网络权重
torch.save(net.state_dict(), "expert_after_training")
# 评估训练之后的情况
evaluate(net, device=DEVICE, title="after training")
```

4.3 模型测试

为测试模型对清晰图像和模糊图像两种照片的分类准确性，测试数据集为2494张clear images以及 2494张blurry images的照片。同时也检验了在17953张clear images和17953张blurry images上的准确率。

5 求解结果与比较分析

5.1 模型训练结果

我们首先用epoch=10训练了Naive Model，得到的准确率如下表。

表1 两个网络的Naive Model训练前后准确率

		训练前		训练后	
		清晰图像	模糊图像	清晰图像	模糊图像
AlexNet	训练集	50	50	89.44	93.39
	测试集	50	50	87.81	87.01
VGG16	训练集	50	50	87.98	93.22
	测试集	50	50	81.76	83.48

可以发现，在运行10个epoch后，对于Naive训练方法，AlexNet表现得略微优于VGG，在模糊图像的测试集上得到更高的准确率；在清晰图像的测试集上，虽然训练时并没有清晰图像，但准确率都接近模糊图像测试集，甚至AlexNet在清晰图像上的准确率会略高。说明虽然直接利用模糊图像进行训练是可行的，但是这种方式并非“模糊图像分类”的特定解决方案，对于“模糊图像”本身没有很大提升。

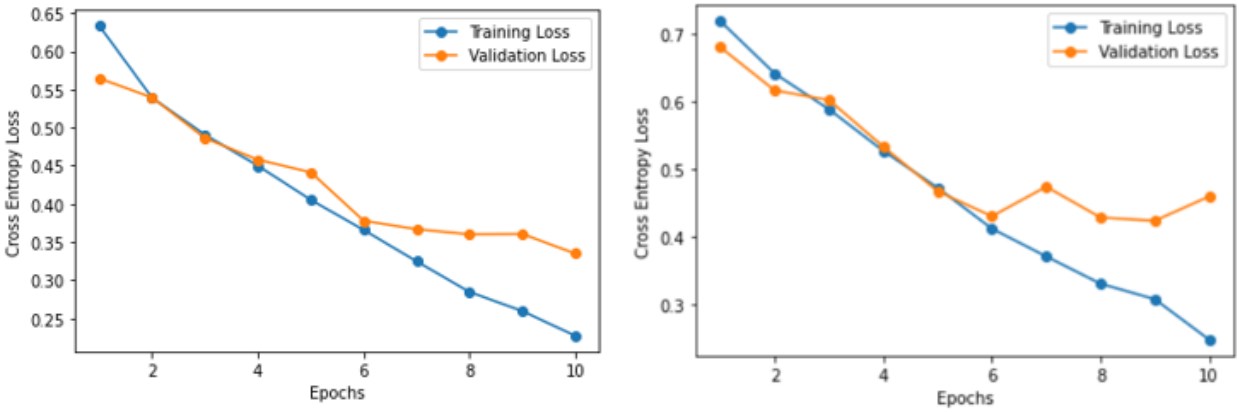

图7 两个网络的Naive Model训练过程损失函数变化

从损失函数值的变化来看，两个模型的训练损失一直在下降，没有达到收敛，但验证集上的损失已经基本稳定，很少有大幅度下降。因此虽然仅有10个epoch，仍然可以认为这一结果可以代表完全完成训练的结果。然而，在损失函数和准确率上，网络结构更深的VGG都没有优于AlexNet，这可能是由于训练迭代次数不多，VGG的优势尚未体现，也可能是AlexNet在这种视觉任务上更为合适。

在对Expert Model进行训练时，我们首先需要找到合适的预训练epoch，为此，我们先在AlexNet上进行了3个epoch的预训练+10个epoch的训练。

表2 AlexNet的Expert Model训练结果

		训练前		预训练后		训练后	
		清晰图像	模糊图像	清晰图像	模糊图像	清晰图像	模糊图像
AlexNet	训练集	50	50	85.26	62.70	87.44	95.73
	测试集	50	50	82.88	61.75	85.28	86.57

对于Expert Model，在经过清晰图像的预训练后，模型在清晰图像上的正确率显著高于对于模糊图像的正确率，证明模糊图像分类不能仅仅依托于清晰图像；当预训练后再进行训练，得到的模型正确率在模糊图像上有了显著提升，且在训练集上优于Naive Model。但与预期不同的是，Expert Model在测试集上的结果并没有优于Naive Model。

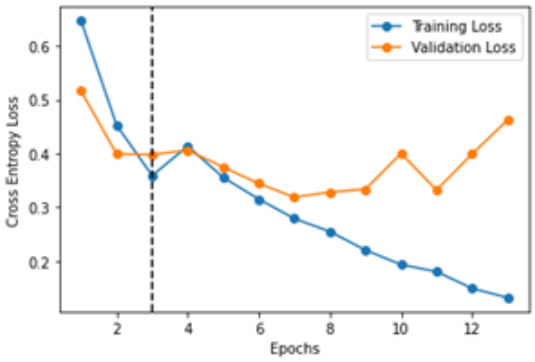

图8 AlexNet的Expert Model训练过程损失函数

通过绘制模型训练过程的损失函数变化图，可以发现在3个epoch后，Expert Model的预训练损失函数还远未接近收敛，可能导致预训练的效果没有体现出来，所以我们增加了预训练的epoch到10个，重新训练模型。

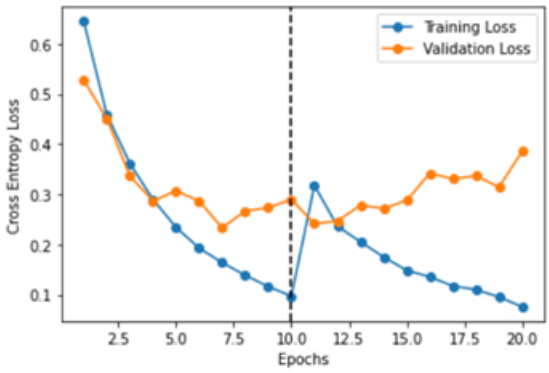

图9 增加预训练迭代次数后AlexNet的Expert Model损失函数变化

表3 增加预训练迭代次数后AlexNet的Expert Model训练结果

		训练前		预训练后		训练后	
		清晰图像	模糊图像	清晰图像	模糊图像	清晰图像	模糊图像
AlexNet	训练集	50	50	98.16	75.21	88.49	97.57
	测试集	50	50	92.34	73.10	86.09	88.41

此时可以发现，Expert Model的训练达到了非常好的效果，在预训练后，清晰图像达到了非常高的准确率，而在完成训练后，各个准确率也有所提升。虽然我们对模糊图像的训练仍然是10个epoch，但可以发现，增加预训练的epoch同样提高了模糊图像的测试准确率，这说明利用清晰图像进行训练是可以帮助机器识别模糊图像的。

考虑到此时清晰图像也经过了10个epoch的训练，与Naive Model得到的清晰图像正确率不具有可比性；而模糊图像在两个方法中均为10个epoch的训练，因此我们特别关注训练后模糊图像测试集上的正确率指标。我们对VGG也进行了Expert Model的训练，最终得到了以下结果。

表4 两个网络的Expert Model最终训练结果

	训练前	训练后
AlexNet	50	88.41
VGG	50	90.34

我们发现，VGG得到的准确率略高于AlexNet，考虑到两个模型在结构上的差异，我们认为VGG能得到更好的结果可能是由于它使用堆叠的小卷积核来代替大卷积核，使网络深度增加，而更深的神经网络在利用更多数据进行多次迭代训练后，能得到更优的模型。

5.2 模型特征层

两个模型的训练结果都不错，考虑到AlexNet在设计上更接近视觉系统，我们可视化了AlexNet第一个卷积层的16个filter，得到了以下结果 。

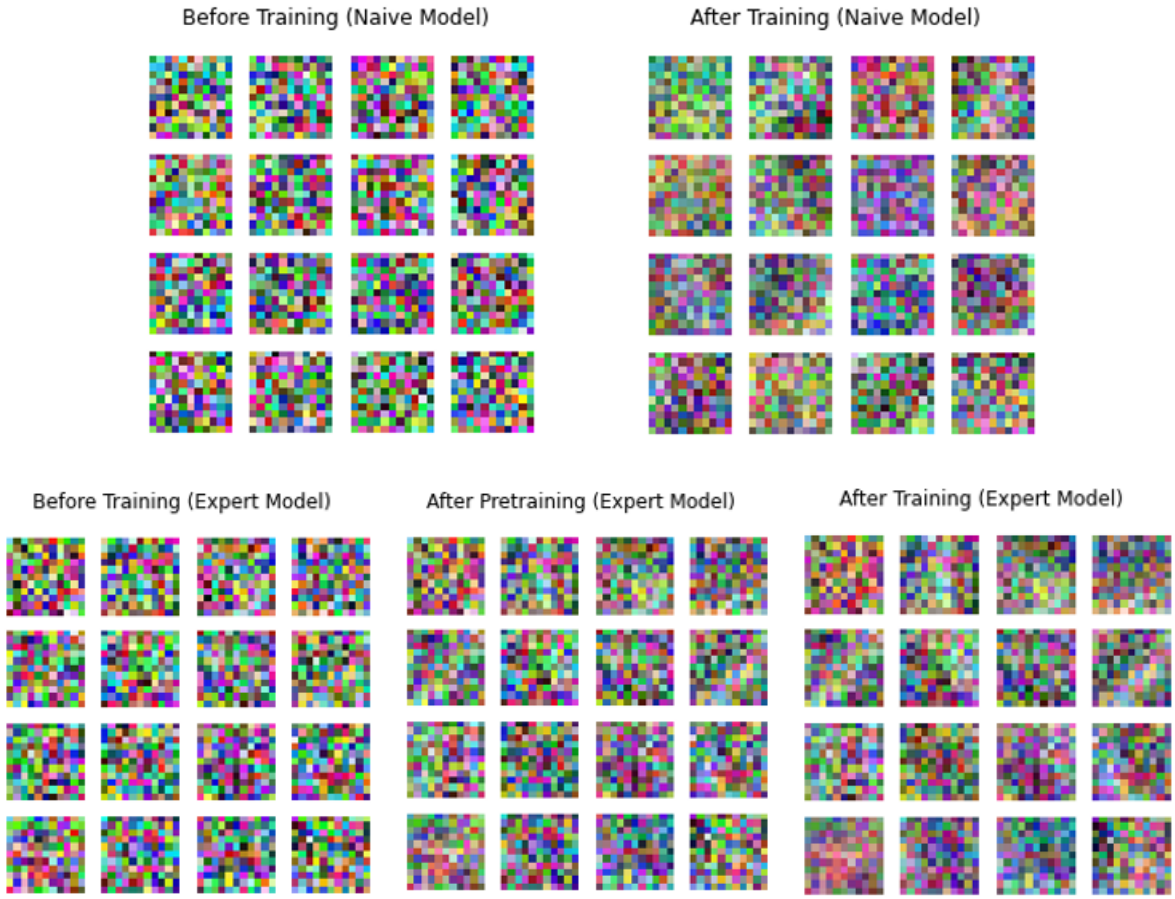

图10 AlexNet的第一个卷积层的filter（上：Naive；下：Expert）

看起来，在训练后，虽然模型的准确率得到了大幅度提升，但这些滤波器并没有特别的意义，且训练前后只有非常细微的一些变化。

我们又可视化了AlexNet第三四五层卷积层的输出，这些层在理论上对应于视觉系统的V1、V2、V4皮层，其中V1是初级视皮层，V2是次级视皮层，V4层级更高。在神经科学的研究中，V1对简单刺激反应，与边缘线条以及朝向有关，V2与轮廓形状有关，V4对物体和颜色特征敏感。

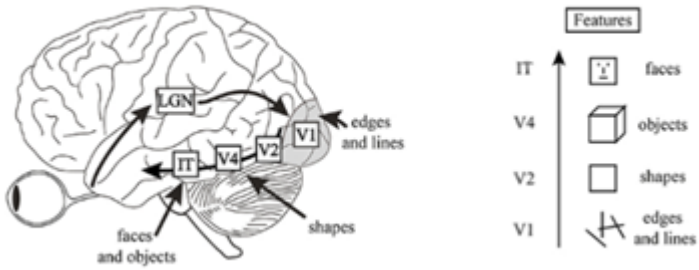

图11 人类视觉信息在皮层上传递过程

我们以数据集中的两张图像为例，将输出可视化。

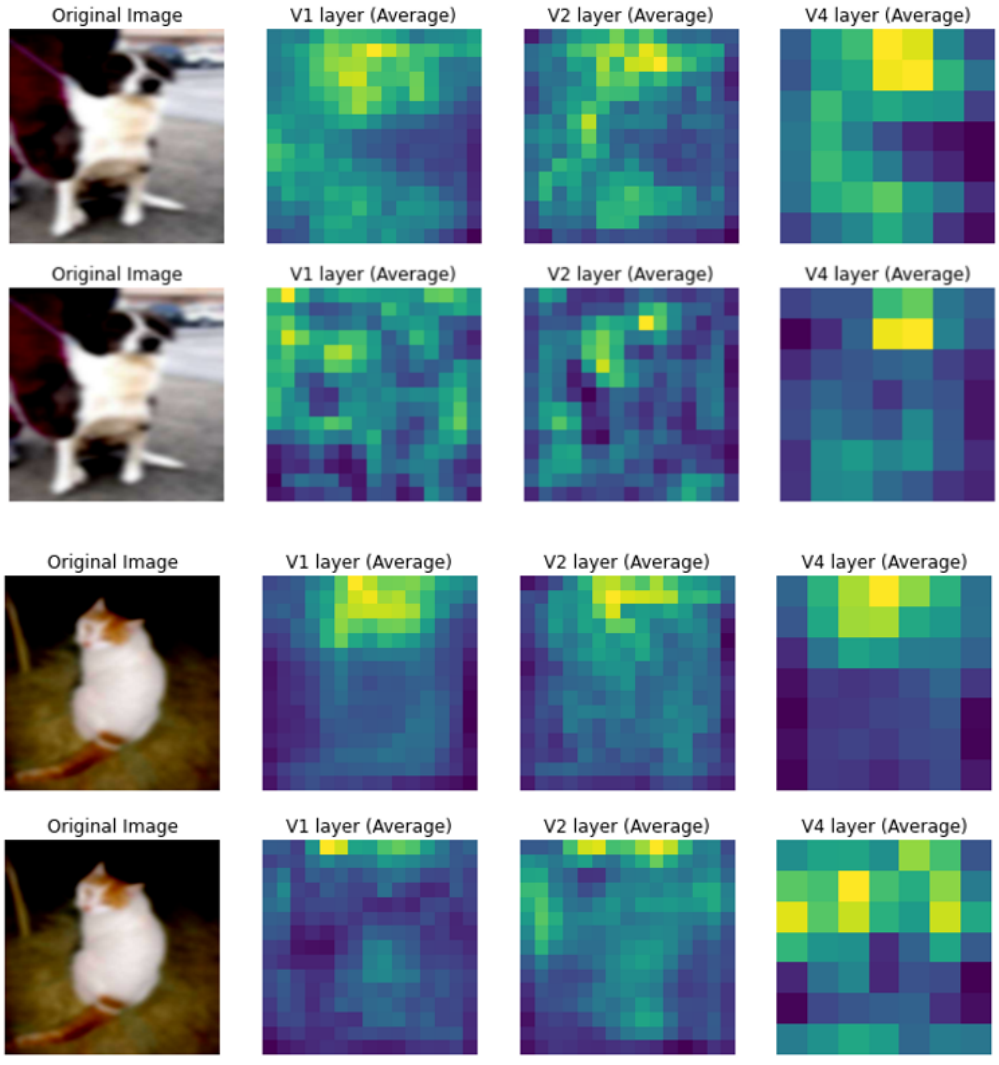

图12 AlexNet Naive Model对样例图片的第三四五层卷积层输出（上1：训练前，狗；上2：训练后，狗；下：猫）

对于Naive Model，从可视化的结果来看，模型经过训练，中间层的输出发生了改变，说明模型确实学到了一些信息。但这与我们大脑的视觉系统却没有什么共同之处。

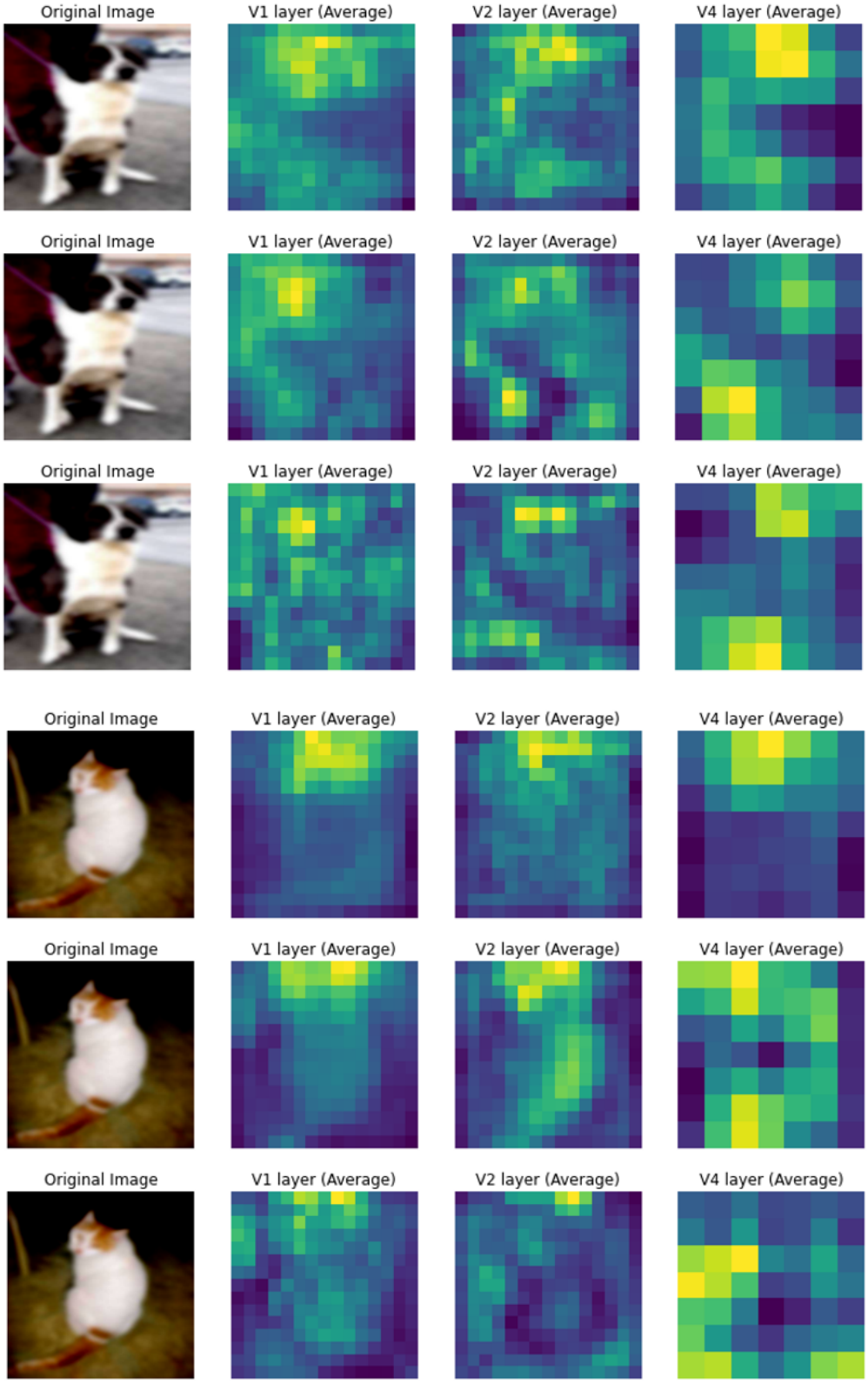

图13 AlexNet Expert Model对样例图片的第三四五层卷积层输出（上：狗，训练前/预训练后/训练后；下：猫）

对于Expert Model，经过预训练后的输出和最终完成训练后的输出并不相同，说明即使经过了清晰图像的训练，再用模糊图像训练时，模型依然从模糊图像里学到了新的信息。

为了考察AlexNet和视觉系统的相似性，我们还找到了在大数据集ImageNet上预训练过的AlexNet模型参数并将其导入，然后将它的第一层16个filter进行可视化，此时可以发现，在ImageNet上完成预训练的模型表现出了与视觉系统非常相似的特征。在第一个卷积层，也就是网络中的初级层，filter对边缘线条、朝向信息非常敏感，这与视觉系统的V1非常相似。

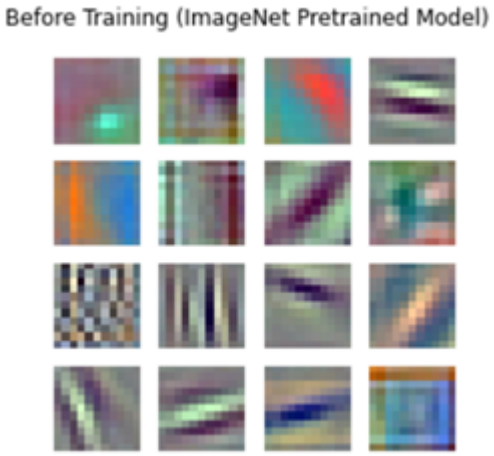

图14 ImageNet预训练过的AlexNet的第一个卷积层里16个filter

因此，我们的模型没有表现出与大脑的相似性，原因可能是数据集太小。

6 结论

在本研究中，我们采用了两种不同的训练方式，发现经过预训练的模型能够得到更优的绩效表现，这说明“经验”对模型的重要性；通过比较AlexNet和VGG两个模型，我们发现网络结构更深的VGG略优于AlexNet，即网络结构对于模型训练也有一定作用；通过对AlexNet进行可视化，我们比较了人工神经网络模型与大脑视觉系统，发现只有经过大样本数据集训练后的网络才会表现出与视觉系统类似的特征，证明了视觉系统的复杂性，也说明后续可以继续思考对模型结构的优化。如果模型能在小样本数据集上也表现出类似特征，或许能在计算机视觉的各个任务上有更优的表现。

7 感想与分工

7.1 感想

刘一诺:

这是一个结合了机器学习与神经科学的研究，主要也是因为我们组的两名成员分别来自控制学院和心理系，可以算是互相结合了各自专业所长，进行了学科的交叉融合，也让我认识到神经科学不止心理学中所学的那些实验研究，还可以通过数据建模等方式，从另一个角度看待问题。通过这个研究，我对于之前没怎么接触的python有了更熟练的掌握，也学到了很多，比如CNN网络结构、利用Pytorch进行训练的方法等。

我们提出的这个问题其实是非常基础但经典的机器学习问题，即图像分类。但是我们从生活出发，考虑到人类在看到模糊东西的时候 (比如近视的人摘下眼镜的时候)，依然能够很好地对物体进行分类，又结合新生儿的微弱视觉，于是想知道计算机视觉对于模糊图像的分类会是怎样。查阅资料发现这方面的研究并不多，因此确定了在这方面进行探索。

我们并没有过多地改进模型框架，而是借用了传统的卷积神经网络，因为不太熟悉，所以也从网上参考了不少代码。小组是两个人构成，每个人各自训练一个网络，刚好能很好地进行对比。我在训练过程中用的是谷歌的Colab平台，这个平台的GPU使用有时长上限，达到上限后要间隔24h才能再次使用，而且Colab经常会在经过一段漫长的训练后断开连接，于是前前后后无效训练了很多次。最后是通过五个epoch五个epoch地分开跑，每次保存网络的值，下次加载继续训练，慢慢才跑完，因为中途有忘记保存损失函数，所以在结果里Expert Model的损失函数图没能画出来。整个训练的过程还是非常漫长的，感觉如果我不是外专业而是真的做这个的话，非常需要一个稳定的服务器或者一个自己的GPU，否则会在训练上浪费很大时间。因为这个，我们也没有太多地进行调参，不过选定的参数已经能够有比较好的指标，可以进行对比分析了。

虽然没有对模型进行很多的创新，但是通过这个实验，我们更加深入地了解并学习了经典的CNN框架，对于之前对机器学习了解不多的我来说已经学到了很多的。我们在实验里也引入了不少神经科学的内容，可惜训练出的模型并没有给什么很有意义的神经科学上的启发，只能说人工神经网络虽然是依照大脑神经元网络构造的，但在实际工作的过程中还是存在非常多的区别。当然也有可能是数据集太小的问题，因为人脑毕竟已经有这么多年经验信息的输入了，而人工神经网络从零开始数据驱动训练，想要有类似表现确实困难。

机器学习(尤其是深度学习)这几年越来越火，在很多任务上的表现也都非常出色，可能这也是为什么我们系为我们班加了这么一门必修课。希望有机会的话可以在这方面进行更多的尝试。

俞一诺:

很高兴能与求心的同学一起组队，也没有想到在自己学院的专业课里碰到了外专业的好朋友。这次大作业是我们两位同学一起合作完成，虽然碰到了许多困难，但最终大部分都攻破了，最后得到的结果也比较令人满意。

我们选题选择了模糊视觉这个主题，并没有选择MO平台上的作业，是想有新意、有实际意义一些。因为，对我自己来说，我是个高度近视的人，在失去眼镜之后就几乎成为了瞎子。模糊视觉肯定有比较大的现实意义。而目前机器视觉也是十分热门。于是我们就查阅了相关机器视觉和模糊视觉的资料，发现在这个领域内并没有比较成体系的研究。所以就确定了这个选题。

一开始进行这个项目还是蛮难的，因为就我而言，虽然我是控制的学生但我之前没有机器学习或者大数据的基础，对于训练的流程、网络的搭建这类还比较陌生。所以我花了一些时间先自己去学习了一些这方面的内容，然后通过MO平台上小作业的完成来做了一些实战训练，这才慢慢地开始熟悉机器学习的领域。

我们一起搜索了不同的CNN网络的资料，再在外网上面找了一些现成的能用的网络代码，想通过不同结构的CNN来试试哪个效果更好。最终确定了两个网络，即AlexNet和VGG。我负责AlexNet这一个部分。我采用的是torch框架，原本以为完整的网络结构之下，训练并不会很难。但实际上，因为数据集很庞大，且cv的图像处理很耗时，所以在本地用cpu跑一轮就需要半个小时 (自己的电脑没有cuda)。最后借用了室友的电脑用gpu来跑，速度变快了一些，但用naïve model完整训练一遍还得半个小时。因为耗时，所以参数的调整并没

有那么理想。我们就选择了相对较好的学习率和epoch。这样已经能够形成较好的结果，也能够出对比了。

相比训练过程而言，最后报告的撰写和PPT制作并不是什么难事。这次的大作业虽然它的成绩占比不高，但仍然十分重要，且最后完成的结果和项目本身都十分有意义。我也从这次的大作业和这门课的实验环节里学到了很多东西。首先是pycharm的使用和python语言的熟悉。从c语言迁移到python，虽然不是一件难事，但也需要一些时间，只有自己亲身写代码才会正确的使用。其次是更加熟悉了网络训练的流程，也去仔细查了网络的结构、损失函数、优化器等与机器学习相关的内容。另外，也熟悉了torch框架和图像识别的过程。

最后，很感谢这次作业。实战训练确实比理论更加重要。希望本次作业学到的知识，能够在今后的课程中发挥作用，

7.2 分工

	刘一诺	俞一诺
收集资料	√	√
讨论与选题	√	√
设计研究方案	√	√
编写代码及训练网络	VGG	AlexNet
报告撰写	选题背景 研究综述 结果分析	问题描述 算法描述 特征层可视化
制作展示 PPT		√
修改 PPT 与展示	√	

参考文献

[1] 李光明, 薛丁华, 加小红, 李云彤, 雷涛. (2018). 基于多尺度图像增强结合卷积神经网络的纸病识别分类. 中国造纸, 37(8): 47-54.

[2] 常荣. (2021). 基于卷积神经网络的高速公路团雾图像分类的研究. 硕士论文, 安徽建筑大学.

[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*, vol. 25, pp. 1097-1105.

[4] Ankur Hariotosh, Chetan Ralekar, Taranjit Kaur, Tapan Kumar Gandhi. (2019). Human Visual Learning Inspired Effective Training Methods. *2019 IEEE 16th India Council International Conference (INDICON)*, pp. 1-4,

[5] Bang Liu, Yan Liu, Kai Zhou. (2014). Image Classification for Dogs and Cats. *TechReport*, University of Alberta.

[6] Cristian Szegedy, Wei Liu, Yangqing Jia, Rierre Sermanet, Scot Reed, Dragomir Auguelov, Dumitru Erhan, Vincent Hanhoucke, Andrew Rabinovich. (2015). Going Deeper with Convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9.

[7] Daniel L. K. Yamins, Ha Hong, Charles F. Cadieu, Ethan A. Solomon, Darren Seibert, and James J. DiCarlo. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *PNAS*, vol. 111, No. 23, pp. 8619-8624.

[8] Daniel L. K. Yamins, Ha Hong, Charles Cadieu, James J. DiCarlo. (2013). Hierarchical Modular Optimization of Convolutional Networks Achieves Representations Similar to Macaque IT and Human Ventral Stream. *Advances in Neural Information Processing Systems (NIPS)* 26.

[9] Daniel L. K. Yamins, James J. DiCarlo. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, vol.19, pp. 356-365.

[10] Devrim Akgun. (2020). An Evaluation of VGG16 Binary Classifier Deep Neural Network for Noise and Blur Corrupted Images. *Sakarya University Journal of Computer and Information Sciences*, vol. 3, No. 3, pp. 264-271.

[11] D. H. Hubel, T. N. Wiesel. (1961). Integrative Action in the Cat's Lateral Geniculate Body. *The Journal of Physiology*, vol. 155, No. 2, pp. 385-398.

[12] Jeremy Elson, John R. Douceur, Jon Howell, Jared Saul. (2007). Asirra: A Captcha that Exploits Interest-Aligned Manual Image Categorization. *CCS'07*, pp. 366-374.

[13] John H. R. Maunsell. (1987). Visual Processing in Monkey Extrastriate Cortex. *Annual review of neuroscience*, vol.10, No.1, pp. 363-401.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778.

[15] Karen Simonyan, Andrew Zisserman. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv*:1409.1556.

[16] M. Shanmukhi, K. Lakshmi Durga, Madela Mounika, Kurakula Keerthana. (2018). Convolutional Neural Network for Supervised Image Classification. *International Journal of Pure and Applied Mathematics*, vol. 199, No.14, pp. 77-83.

[17] Rufin Vogels, Guy A. Orban. (1996). Coding of Stimulus Invariances by Inferior Temporal Neurons. *Progress in Brain Research*, vol. 112, pp. 195-211.

[18] Rui Wang, Wei Li, Runnan Qin, JinZhong Wu. (2017). Blur Image Classification based on Deep Learning. *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 1-6.

[19] Shaul Hochstein, Merav Ahissar. (2001). View from the Top: Hierarchies and Reverse Hierarchies in the Visual System. *Neuron*, vol. 36, No.5, pp. 791-804.

[20] V. E. Dementyiev, N. A. Andriyanov, K. K. Vasilyiev. (2020). Use of Images Augmentation and Implementation of Doubly Stochastic Models for Improving Accuracy of Recognition Algorithms Based on Convolutional Neural Networks. *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, pp. 1-4.

[21] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol.86, No.11, pp. 2278-2324.

源代码

```
# ==============================Colab dependencies ==============================
!pip install torch_intermediate_layer_getter --quiet
!pip install Pillow --quiet

# ====================================Imports===================================
import random
import time
import numpy as np
import tqdm.notebook as tqdm
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
from shutil import copyfile, rmtree
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
# For getting output of intermediate layers in Pytorch
from torch_intermediate_layer_getter import IntermediateLayerGetter as LayerGetter
# For interactive visualization
from torch.utils.tensorboard import SummaryWriter
import torchvision
from torchvision import transforms
import os, zipfile, requests

# ==================================Environment=================================
def set_seed(seed=None, seed_torch=True):
   if seed is None:
      seed = np.random.choice(2 ** 32)
   random.seed(seed)
   np.random.seed(seed)
   if seed_torch:
      torch.manual_seed(seed)
      torch.cuda.manual_seed_all(seed)
      torch.cuda.manual_seed(seed)
      torch.backends.cudnn.benchmark = False
      torch.backends.cudnn.deterministic = True

# In case that `DataLoader` is used
def seed_worker(worker_id):
   worker_seed = torch.initial_seed() % 2**32
   np.random.seed(worker_seed)
   random.seed(worker_seed)

def set_device():
```

```
    device = "cuda" if torch.cuda.is_available() else "cpu"
    if device != "cuda":
        print("WARNING: For this notebook to perform best, "
              "if possible, in the menu under `Runtime` -> "
              "`Change runtime type.`  select `GPU` ")
    else:
        print("GPU is enabled in this notebook.")
    return device

SEED = 42
set_seed(seed=SEED)
DEVICE = set_device()

# ===================================Dataset=====================================
filenames = ["catvdog_clear.zip", "catvdog_blur_2.zip", "catvdog_blur_5.zip"]
urls = ["https://osf.io/hj2gd/download", "https://osf.io/xp6qd/download",
"https://osf.io/wj43a/download"]
for fname, url in zip(filenames, urls):
    if not os.path.isfile(fname):
        try:
            r = requests.get(url)
        except requests.ConnectionError:
            print("!!! Failed to download data !!!")
        else:
            if r.status_code != requests.codes.ok:
                print("!!! Failed to download data !!!")
            else:
                with open(fname, "wb") as fid:
                    fid.write(r.content)

for fname in filenames:
    zip_ref = zipfile.ZipFile(fname, 'r')
    zip_ref.extractall()
    zip_ref.close()
    os.remove(fname)

# ================================Preprocessing=================================
# Define Preprocessing Filters
preprocessing = transforms.Compose([transforms.ToTensor(), transforms.Resize((256,
256)),transforms.Normalize((0.5, 0.5, 0.5),(0.5, 0.5, 0.5))])
clear_train_data =
torchvision.datasets.ImageFolder(root="dataset/train",transform=preprocessing)
clear_test_data =
torchvision.datasets.ImageFolder(root="dataset/test",transform=preprocessing)
noisy_train_data =
torchvision.datasets.ImageFolder(root="dataset_blur_5/train",transform=preprocessing)
noisy_test_data =
torchvision.datasets.ImageFolder(root="dataset_blur_5/test",transform=preprocessing)
```

```
# function to apply a training-validation set split on a dataset
def validation_split(train_data, val_ratio = 0.2):
   train_indices, val_indices, _, _ =
train_test_split(range(len(train_data)),train_data.targets,stratify=train_data.targets,
test_size=val_ratio)
   train_split = torch.utils.data.Subset(train_data, train_indices)
   val_split = torch.utils.data.Subset(train_data, val_indices)
   return train_split,val_split

# Define Dataloaders for Training, Validation and Test sets
batch_size = 32
clear_train_split,clear_val_split = validation_split(clear_train_data)
clear_train_batches = torch.utils.data.DataLoader(clear_train_split,
batch_size=batch_size,shuffle=True)
clear_val_batches = torch.utils.data.DataLoader(clear_val_split,
batch_size=batch_size,shuffle=True)
clear_test_batches = torch.utils.data.DataLoader(clear_test_data,
batch_size=batch_size,shuffle=True)
noisy_train_split,noisy_val_split = validation_split(noisy_train_data)
noisy_train_batches = torch.utils.data.DataLoader(noisy_train_split,
batch_size=batch_size,shuffle=True)
noisy_val_batches = torch.utils.data.DataLoader(noisy_val_split,
batch_size=batch_size,shuffle=True)
noisy_test_batches = torch.utils.data.DataLoader(noisy_test_data,
batch_size=batch_size,shuffle=True)


# ==================================VGG=======================================
class VGG(nn.Module):
      def __init__(self, features, num_classes=2, init_weights=False):
            super(VGG, self).__init__()
            self.features = features
            #先会进行一个展平处理
            self.classifier = nn.Sequential( #定义分类网络结构
                  nn.Dropout(p=0.5), #减少过拟合
                  nn.Linear(512*8*8, 4096),
                  nn.ReLU(True),
                  nn.Dropout(p=0.5),
                  nn.Linear(4096, 4096),
                  nn.ReLU(True),
                  nn.Linear(4096, num_classes)
            )
            if init_weights:
                  self._initialize_weights()

      def forward(self, x):
            # N x 3 x 224 x 224
```

```
            x = self.features(x)
            # N x 512 x 7 x 7
            x = torch.flatten(x, start_dim=1)#展平处理
            # N x 512*7*7
            x = self.classifier(x)
            return x

      def _initialize_weights(self): #初始化权重函数
            for m in self.modules():
                  if isinstance(m, nn.Conv2d):
                        nn.init.xavier_uniform_(m.weight)
                        if m.bias is not None:
                              nn.init.constant_(m.bias, 0)#初始化偏置为0
                  elif isinstance(m, nn.Linear):
                        nn.init.xavier_uniform_(m.weight)
                        # nn.init.normal_(m.weight, 0, 0.01)
                        nn.init.constant_(m.bias, 0)

def make_features(cfg: list):#提取特征函数
      layers = [] #存放创建每一层结构
      in_channels = 3 #RGB
      for v in cfg:
            if v == "M":
                  layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                  conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
                  layers += [conv2d, nn.ReLU(True)]
                  in_channels = v
      return nn.Sequential(*layers)    #通过非关键字参数传入

cfgs = {
      'vgg11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'], #数字
为卷积层个数，M为池化层结构
      'vgg16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M',
512, 512, 512, 'M'],
      'vgg19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512,
512, 'M', 512, 512, 512, 512, 'M'],
}

#实例化VGG网络
def vgg(model_name="vgg16", **kwargs): # 选用VGG16
      try:
            cfg = cfgs[model_name]
      except:
            print("Warning: model number {} not in cfgs dict!".format(model_name))
            exit(-1)
      model = VGG(make_features(cfg), **kwargs) #
      return model
```

```
vgg_model = vgg(model_name='vgg11')

# ====================================AlexNet====================================
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000, downscale=1):
        super(AlexNet, self).__init__()
        self.retina = nn.Sequential(
              nn.Conv2d(3, 64//downscale, kernel_size=11, stride=4, padding=2),
              nn.ReLU(inplace=True),
              nn.BatchNorm2d(64//downscale),
              nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.lgn = nn.Sequential(
              nn.Conv2d(64//downscale, 192//downscale, kernel_size=5, padding=2),
              nn.ReLU(inplace=True),
              nn.BatchNorm2d(192//downscale),
              nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.v1 = nn.Sequential(
              nn.Conv2d(192//downscale, 384//downscale, kernel_size=3, padding=1),
              nn.ReLU(inplace=True),
              nn.BatchNorm2d(384//downscale),
        )
        self.v2 = nn.Sequential(
              nn.Conv2d(384//downscale, 256//downscale, kernel_size=3, padding=1),
              nn.ReLU(inplace=True),
              nn.BatchNorm2d(256//downscale),
        )
        self.v4 = nn.Sequential(
              nn.Conv2d(256//downscale, 256//downscale, kernel_size=3, padding=1),
              nn.ReLU(inplace=True),
              nn.BatchNorm2d(256//downscale),
              nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.it = nn.Sequential(
              nn.Dropout(),
              nn.Linear(256//downscale * 6 * 6, 4096//downscale),
              nn.ReLU(inplace=True),
              nn.Dropout(),
              nn.Linear(4096//downscale, 1024//downscale),
              nn.ReLU(inplace=True)
        )
        self.classifier = nn.Linear(1024//downscale, num_classes)

    def forward(self, x):
        x = self.retina(x)
```

```
        x = self.lgn(x)
        x = self.v1(x)
        x = self.v2(x)
        x = self.v4(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.it(x)
        x = self.classifier(x)
        return x

# ================================Training Function============================
def train(num_epochs, train_batch, val_batch,
                training_losses=None, validation_losses=None, device='cpu'):
    net.train()
    if training_losses is None:
        training_losses = []
    if validation_losses is None:
        validation_losses = []
    for epoch in tqdm.tqdm(range(num_epochs)):
        for batch_idx, (data, target) in enumerate(train_batch):
            data = data.to(device).float()
            target = target.to(device).long()
            optimizer.zero_grad()
            prediction = net(data)
            loss = criterion(prediction, target)
            loss.backward()
            optimizer.step()
            training_losses += [loss.item()]
        for batch_idx, (data, target) in enumerate(val_batch):
            data = data.to(device).float()
            target = target.to(device).long()
            prediction = net(data)
            loss = criterion(prediction, target)
            validation_losses += [loss.item()]
    return training_losses, validation_losses

# ================================Accuracy Function============================
def accuracy(dataloader, device='cpu'):
    net.eval()
    correct = 0
    count = 0
    for data, target in tqdm.tqdm(dataloader):
        data = data.to(device).float()
        target = target.to(device).long()
        prediction = net(data)
        _, predicted = torch.max(prediction, 1)
        count += target.size(0)
        correct += (predicted == target).sum().item()
```

```
    acc = 100 * correct / count
    return count, acc

def evaluate(net, device='cpu', title=""):
    net.eval()
    train_count, train_acc = accuracy(clear_train_batches, device=device)
    test_count, test_acc = accuracy(clear_test_batches, device=device)
    print(f'Accuracy on the {train_count} clear training samples {title}:
{train_acc:0.2f}')
    print(f'Accuracy on the {test_count} clear testing samples {title}:
{test_acc:0.2f}')
    train_count, train_acc = accuracy(noisy_train_batches, device=device)
    test_count, test_acc = accuracy(noisy_test_batches, device=device)
    print(f'Accuracy on the {train_count} blurry training samples {title}:
{train_acc:0.2f}')
    print(f'Accuracy on the {test_count} blurry testing samples {title}:
{test_acc:0.2f}')

# ================================Naive Model=================================
#net = AlexNet(num_classes=2,downscale=2)
net = vgg_model   # 二选一
# 以下过程是共同的
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=3e-4)
net.to(DEVICE)
evaluate(net, device=DEVICE, title="before training") # Evaluate before training

num_pretraining_epochs = 0
num_training_epochs = 10
num_epochs = num_pretraining_epochs + num_training_epochs
torch.save(net.state_dict(), "naive_before_training") # Save network weights
# Training loop
naive_training_losses, naive_validation_losses = train(num_training_epochs,
                                                       noisy_train_batches,
                                                       noisy_val_batches,
                                                       device=DEVICE)
torch.save(net.state_dict(), "naive_after_training")
evaluate(net, device=DEVICE, title="after training") # Evaluate after training

# Plot Loss over epochs
plt.figure()
plt.plot(np.arange(1, num_epochs + 1),
            [np.mean(x) for x in np.array_split(naive_training_losses,
num_training_epochs)],
            "o-", label="Training Loss")
plt.plot(np.arange(1, num_epochs + 1),
            [np.mean(x) for x in np.array_split(naive_validation_losses,
num_training_epochs)],
```

```
            "o-", label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Cross Entropy Loss")
plt.legend()
plt.show()

# ================================Expert Model================================
#net = AlexNet(num_classes=2,downscale=2)
net = vgg_model
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=3e-4)
net.to(DEVICE)
evaluate(net, device=DEVICE, title="before training")   # Evaluate before pretraining

num_pretraining_epochs = 10
num_training_epochs = 10
num_epochs = num_pretraining_epochs + num_training_epochs
torch.save(net.state_dict(), "expert_before_training")

# Pretraining loop
training_losses, validation_losses = train(num_pretraining_epochs, clear_train_batches,
                                                                   clear_val_batches,
                                                                   device=DEVICE)
evaluate(net, device=DEVICE, title="after pretraining") # Evaluate after pretraining
torch.save(net.state_dict(), "expert_after_pretraining")

# Training loop
experienced_training_losses, experienced_validation_losses = train(num_training_epochs,
                                                                   noisy_train_batches,
                                                                   noisy_val_batches,
                                                         training_losses=training_losses,
                                                   validation_losses=validation_losses,
                                                                   device=DEVICE)
evaluate(net, device=DEVICE, title="after training") # Evaluate after training

# Plot Loss over epochs
plt.figure()
plt.plot(np.arange(1, num_epochs + 1),
            [np.mean(x) for x in np.array_split(experienced_training_losses,
num_epochs)],
            "o-", label="Training Loss")
plt.plot(np.arange(1, num_epochs + 1),
            [np.mean(x) for x in np.array_split(experienced_validation_losses,
num_epochs)],
            "o-", label="Validation Loss")
plt.axvline(num_pretraining_epochs, linestyle='dashed', color='k')
plt.xlabel("Epochs")
plt.ylabel("Cross Entropy Loss")
```

```
plt.legend()
plt.show()

# ============================卷积层可视化==================================
def plot_filter(net, title=""):
    layer = 0
    with torch.no_grad():
        params = list(net.parameters())
        fig, axs = plt.subplots(4, 4, figsize=(4, 4))
        filters = []
        for filter_index in range(min(16, params[layer].shape[0])):
            row_index = filter_index // 4
            col_index = filter_index % 4

            filter = params[layer][filter_index, :, :, :]
            filter_image = filter.permute((1,2,0))
            scale = np.abs(filter_image).max()
            scaled_image = filter_image / (2 * scale) + 0.5
            filters.append(scaled_image)
            axs[row_index, col_index].imshow(scaled_image)
            axs[row_index, col_index].axis('off')
        plt.suptitle(title)
        plt.show()

net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('naive_before_training'))
plot_filter(net, "Before Training (Naive Model)")

net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('naive_after_training'))
plot_filter(net, "After Training (Naive Model)")

net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('expert_before_training'))
plot_filter(net, "Before Training (Expert Model)")

net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('expert_after_pretraining'))
plot_filter(net, "After Pretraining (Expert Model)")

net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('expert_after_training'))
plot_filter(net, "After Training (Expert Model)")

# ============================中间层输出可视化==========================
return_layers = {
        'v1': 'v1',
        'v2': 'v2',
```

```
        'v4': 'v4',
    }
def plot_intermediate_layers(image,net,return_layers=return_layers):
    intermediate_output = LayerGetter(net, return_layers=return_layers)(image)
    fig = plt.figure(figsize=(12, 3))
    ax=fig.add_subplot(141)
    ax.imshow(image.squeeze(0).permute(1, 2, 0))
    plt.axis('off')
    plt.title('Original Image')
    ax=fig.add_subplot(142)
    ax.imshow(intermediate_output[0]['v1'].detach().cpu().squeeze(0).mean(axis=0))
    plt.axis('off')
    plt.title('V1 layer (Average)')
    ax=fig.add_subplot(143)
    ax.imshow(intermediate_output[0]['v2'].detach().cpu().squeeze(0).mean(axis=0))
    plt.axis('off')
    plt.title('V2 layer (Average)')
    ax=fig.add_subplot(144)
    ax.imshow(intermediate_output[0]['v4'].detach().cpu().squeeze(0).mean(axis=0))
    plt.axis('off')
    plt.title('V4 layer (Average)')
    plt.show()

print("Naive Model\n=============")
print("Before Training")
net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('naive_before_training'))
plot_intermediate_layers(noisy_dog_image, net)
plot_intermediate_layers(noisy_cat_image, net)
print("After Training")
net = AlexNet(num_classes=2, downscale=2)
net.load_state_dict(torch.load('naive_after_training'))
plot_intermediate_layers(noisy_dog_image, net)
plot_intermediate_layers(noisy_cat_image, net)

print("Expert Model\n=============")
print("Before Training")
net = AlexNet(num_classes=2,downscale=2)
net.load_state_dict(torch.load('expert_before_training'))
plot_intermediate_layers(noisy_dog_image,net)
plot_intermediate_layers(noisy_cat_image,net)
print("After Pretraining")
net = AlexNet(num_classes=2,downscale=2)
net.load_state_dict(torch.load('expert_after_pretraining'))
plot_intermediate_layers(noisy_dog_image,net)
plot_intermediate_layers(noisy_cat_image,net)
print("After Training")
net = AlexNet(num_classes=2,downscale=2)
```

```
net.load_state_dict(torch.load('expert_after_training'))
plot_intermediate_layers(noisy_dog_image,net)
plot_intermediate_layers(noisy_cat_image,net)

# ==========================ImageNet Pretrained Model=======================
AlexNet_model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)
AlexNet_model.eval()
# modify the architecture
AlexNet_model.classifier[4] = nn.Linear(4096,1024)
AlexNet_model.classifier[6] = nn.Linear(1024,2)
AlexNet_model.eval()
plot_filter(AlexNet_model, "Before Training (ImageNet Pretrained Model)")
```