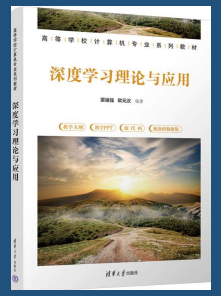
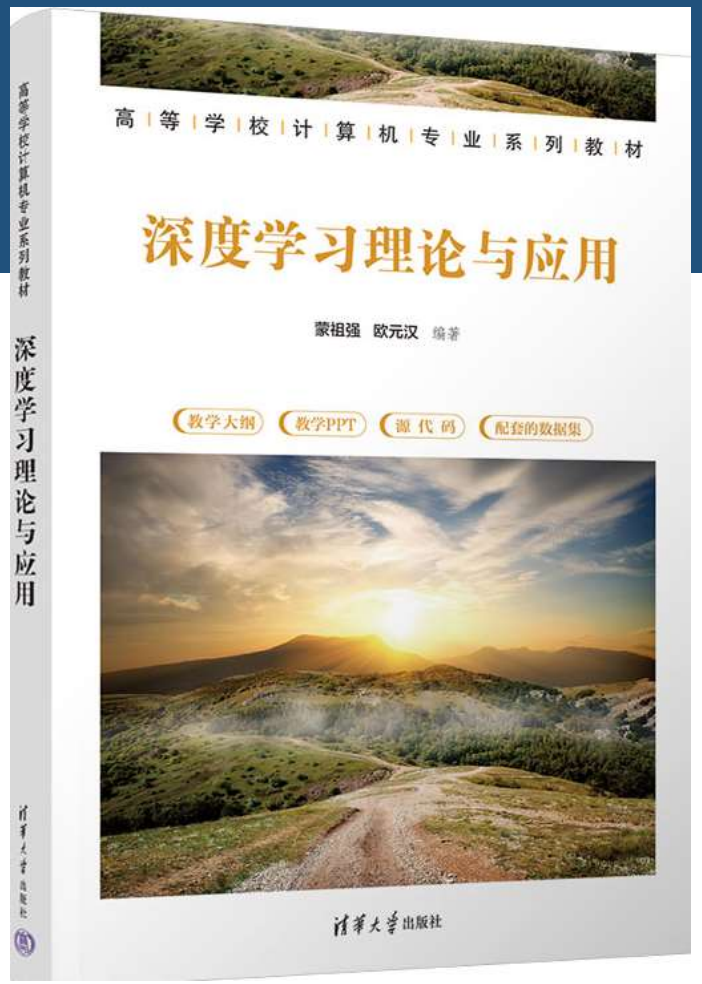


深度学习理论与应用

Deep Learning Theory and Applications

蒙祖强，欧元汉 编著



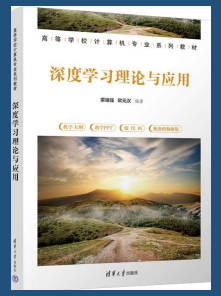
教　材

全国各大书店网店均有销售

- **教学大纲**：提供面向教育工程认证的教学大纲
- **教学PPT**：提供课堂教学用的PPT课件
- **源代码**：提供教材涉及的全部源代码
- **数据集**：提供教材示例、案例用到的全部数据集

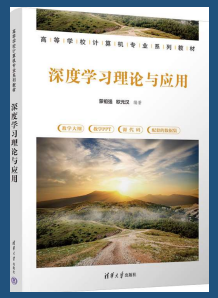
获取教学资源：
http://www.tup.tsinghua.edu.cn/booksCenter/book_09988101.html

教材： 蒙祖强，欧元汉 编著. 深度学习理论与应用. 北京: 清华大学出版社，2023年7月. (书号: 978-7-302-63508-6)



第 9 章 面向解释的深度神经网络可视化方法

蒙祖强，欧元汉 编著. 深度学习理论与应用. 北京: 清华大学出版社，2023年7月.



本章内容

contents

9.1 CNN各网络层输出的可视化

深度神经网络是一种典型的“**黑盒模型**”。对于训练好的模型，只要输入合规的数据，它就能输出一个结果，至于为什么得到这样的结果，我们就不得而知了。然而，对于许多实际应用而言，高精准的模型固然重要，但是为模型决策的过程和结果给出合理和可信的解释同样很重要。例如，在诊疗、金融、安全等领域，如果模型未能给出应有的解释，那么用户可能难以接受模型的结果，从而限制深度模型的应用范围。

深度模型的可解释性研究是当前人工智能研究的热点之一，目前主要分为两种：

- 一种是构造本身可解释性模型（本身可解释性）；
- 另一种是事后构造模型的可解释性（事后可解释性）。

前者对模型的结构要求很高，可能需要更改模型的结构，容易导致模型性能下降；后者则对训练好的模型构造可解释表达，实现对模型的解释，但容易产生额外的误差。

9.1 CNN各网络层输出的可视化

CNN 网络通常是用于提取图像特征的，它主要是由卷积层和池化层组成。

对于一个 CNN 网络层，其输入是一个特征图，其输出也是一个特征图。对于一个给定的卷积层，假设其输入特征图的通道数为 n，输出特征图的通道数为 m，则该层上每个卷积核的深度（或说通道数）为 n，一共有 m 个卷积核。可以看出，输出特征图的通道跟卷积核是一一对应的，即一个卷积核产生一个输出通道。

- 卷积核的作用就是提取特征，提取的结果正是体现在输出特征图的相应通道上。
- 池化层一般不改变特征图的通道数，但会改变通道的大小，其主要作用是提取输入特征图的“显著”特征，同时起到降维的作用。

9.1 CNN各网络层输出的可视化

【例 9.1】对 VGG16 网络层的输出进行可视化，并总结各层输出的特征图的特点。

可视化的基本思路：将 VGG16 各个网络层“拆除”，然后重新“组装”成为跟原来VGG16 一样功能的模型，在“组装”的时候留下各层的输出“接口”，以便截获相关网络层的输出，从而对其可视化。

先加载 VGG16：

```
model = models.vgg16_bn(pretrained=True) #加载 VGG16
model.eval()
```

利用打印语句 print(model)查看 VGG16 的结构，然后据此将卷积网络部分的各个网络层依次抽取并存放到列表 cnn_layers 当中：

```
cnn_layers = []
for k in range(43+1): #添加卷积网络层
cnn_layers.append(model.features[k])
cnn_layers.append(model.avgpool) #自适应平均池化层
```

9.1 CNN各网络层输出的可视化

接着读入图片并张量化：

```
path = r'./data/Interpretability/images'
name = 'both.png'
fn = path + '\\' + name
tfs = transforms.Compose([transforms.Resize(256),
                          transforms.CenterCrop(224),
                          transforms.ToTensor(),
                          transforms.Normalize([0.5, 0.5, 0.5], [0.2, 0.2, 0.2])])
origin_img = Image.open(fn).convert('RGB') #打开图片并转换为 RGB 模型
img = tfs(origin_img) #图片预处理 torch.Size([3, 224, 224])
img = img.unsqueeze(0) #添加批次的维 torch.Size([1, 3, 224, 224])
```

将张量化后的图片依次输入列表 cnn_layers 中的相应网络层，进而可视化输出特征图的相应通道图像。为了能以一定方式观察到输出的结果，我们先定义一个“调色板”cmap：

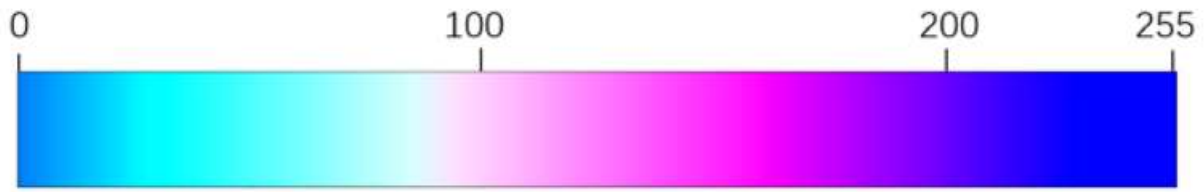
```
cmap = cm.get_cmap('jet')
```

9.1 CNN各网络层输出的可视化

调色板给出了[0, 255]范围内的无符号整数（颜色值）所对应的颜色，如下图所示。

- 0 对应调色板上最左的颜色，这是一种淡蓝色；
- 255 对应最右边的颜色，这是一种深蓝色。
- 随着颜色值从 0 到 255 逐渐增大，调色板上的颜色从左到右，逐渐变红、变深、变蓝。

通过将通道图像上的特征值归一化到[0, 255]后，利用该调色板对它们进行可视化，我们就可以通过颜色的深浅来判断特征值的大小，实现通道图像的可视化，从而判断一个输出的通道图像大致表示原始图像的哪些特征。



9.1 CNN各网络层输出的可视化

下面**函数功能**：利用该调色板在 plt 中显示给定的通道图像。

```
def showImg(channel_img): #channel_img 表示特征图的一个通道图像
    channel_img = torch.relu(channel_img) #torch.Size([224, 224])
    max_min = channel_img.max()-channel_img.min()
    if max_min == 0:
        max_min = 1e-6
        channel_img = (channel_img-channel_img.min())/max_min #归一化到[0,1]
    channel_img = (channel_img**2)*255 #在归一化到[0, 255]
    img = np.array(channel_img.data).astype(np.uint8)
    cmap = cm.get_cmap('jet') #定义调色板
    img = cmap(img )[:, :, 1:] #使用调色板
    plt.imshow(img) #显示图像
    plt.show()
return
```

9.1 CNN各网络层输出的可视化

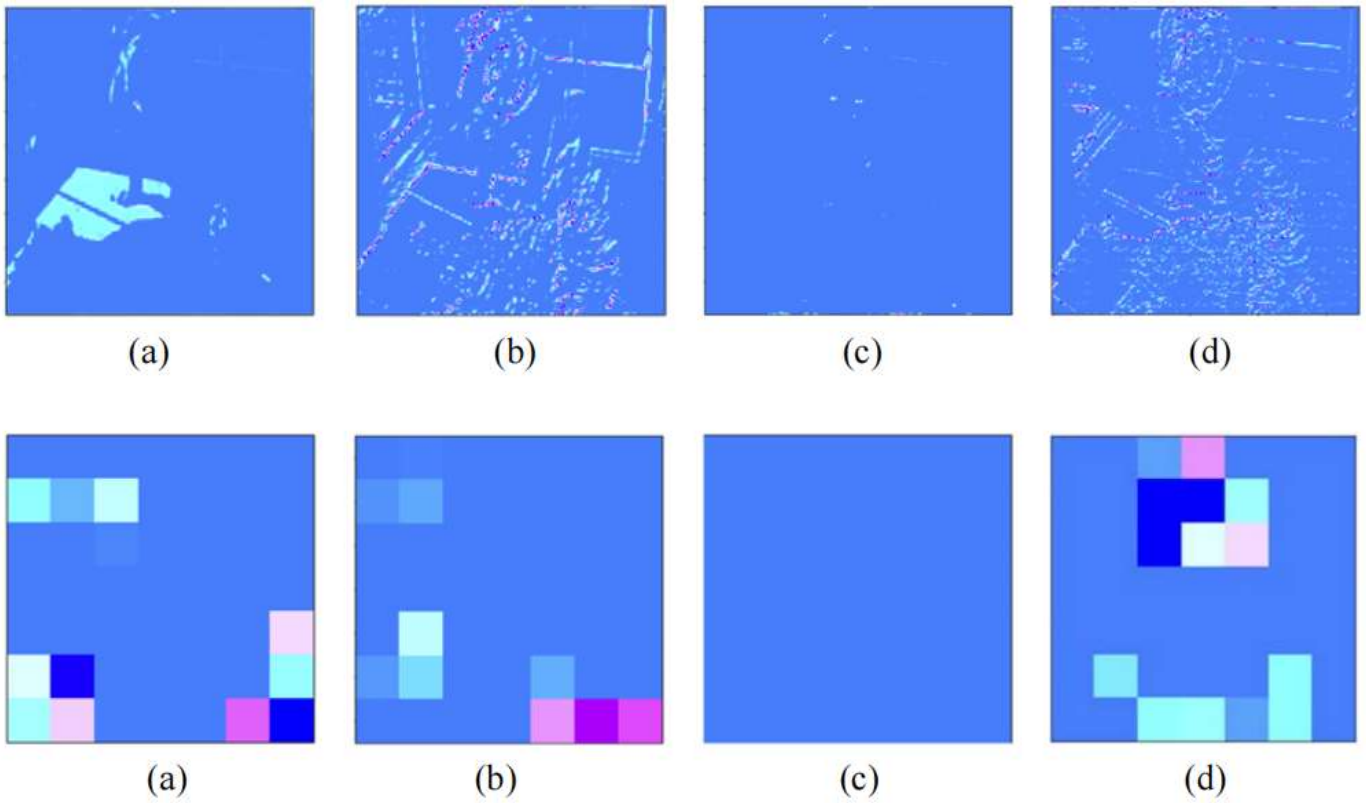
根据第 5 章的介绍我们知道，VGG16 第二卷积层输出的特征图一共有 64 个通道，各通道图像大小为 224*224；自适应平均池化层输出的特征图有 512 个通道，各通道图像大小为7*7。

作为例子，我们从这两个特征图中各选择 4 个通道图像来显示，代码如右所示：

```
out = img
for k,m in enumerate(cnn_layers):
        out = m(out)
        if k == 3: #cnn_layers[3]存放  VGG16 的第二个卷积层
                print('第二个卷积层输出特征图的形状：', out.shape)
                showImg(out[0, 0])
                showImg(out[0, 20])
                showImg(out[0, 40])
                showImg(out[0, 60])
        elif k == 44: #cnn_layers[44]存放  VGG16 的自适应平均池化层
                print('自适应平均池化层输出特征图的形状：', out.shape)
                showImg(out[0, 0])
                showImg(out[0, 20])
                showImg(out[0, 40])
                showImg(out[0, 60])
```

9.1 CNN各网络层输出的可视化

下图（左）所示是输入 VGG16 的原始图像（both.png）。执行上述代码后产生第二个卷积层和自适应平均池化层输出的特征图，从这两个特征图中分别选择四个通道图像来显示，结果如图（右）所示。



9.1 CNN各网络层输出的可视化

横向对比我们发现，对给定的输入图像，有的卷积核（一个输出通道对应一个卷积核）可能提取不到有用的特征，如图 9-3(c)和图 9-4(c)；有的卷积核可以提取到丰富的特征信息，如图 9-3(b)和图 9-3(d)以及图 9-4(a)和图 9-4(d)等。各个通道图一般都不一样，因此输出特征图的通道数越多，则能表征和提取的图像的特征信息就更多；当然，这意味着需要训练更多的参数，需要提供更多的算力来支撑。

纵向对比可以发现，低层的网络层能提取局部的纹理特征和边缘特征，如图 9-3(b)和图9-3(d)；而高层的网络层则提取抽象的全局语义特征，这种特征可能难以从视觉上看出来，如图 9-4(a)、图 9-4(b)和图 9-4(d)。但是，我们可以利用双三次插值方法将通道图像扩展为跟原来输入图像一样大小的图像，并将之覆盖到原来的图像上，从而观察该高层特征大致表示什么内容

9.1 CNN各网络层输出的可视化

例如，对于图 9-4(d)所示的通道图像，扩展后的效果如图 9-5 所示，之后将之覆盖到原图像上，效果如图 9-6 所示。

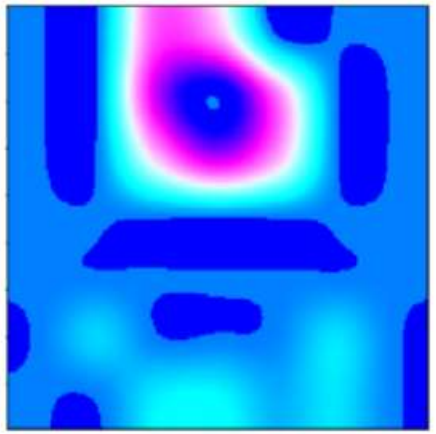


图 9-5 图 9-4(d)所示的通道图像扩展后的效果

图 9-6 扩展图像覆盖到原图像上的效果

9.1 CNN各网络层输出的可视化

例如，对于图 9-4(d)所示的通道图像，扩展后的效果如图 9-5 所示，之后将之覆盖到原图像上，效果如图 9-6 所示。主要实现代码如右：

```
out = img
for k,m in enumerate(cnn_layers):
        out = m(out)
        if k == 44:
                Img44_60 = out[0, 60]
#归一化到[0,1]
img44_60 = (img44_60-img44_60.min())/(img44_60.max()-img44_60.min())
img44_60 = np.array(img44_60.data) #转化为数组
#转化为 PIL 格式，以准备调用 resize()方法进行插值缩放
img44_60 = to_pil_image(img44_60, mode='F')
h,w,_ = np.array(origin_img).shape #获取原图的尺寸
#通过双三次插值方法扩展为跟原图一样大小
over_img = img44_60.resize((h,w), resample=Image.BICUBIC)
over_img = np.array(over_img)*255
over_img = over_img.astype(np.uint8) #归一化到[0,255]范围内
cmap = cm.get_cmap('jet')
over_img = cmap(over_img)[:, :, 1:] #使用调色板
over_img = (255*over_img).astype(np.uint8) #over_img 为数组类型
origin_img = np.array(origin_img)
a = 0.7
#融合两张图片
origin_over_img = Image.fromarray((a * origin_img + (1 - a) * over_img).astype(np.uint8))
plt.imshow(over_img ) #显示扩展后的通道图像
plt.show()
plt.imshow(origin_over_img ) #显示融合后的图像
plt.show()
```

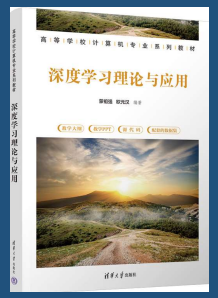
9.1 CNN各网络层输出的可视化

从图 9-5 和图 9-6 可以看出，该通道图像主要表征了狗的脸部及其四周。该网络层输出的特征图一共有 512 个通道，不同通道表征不同的模式，从而形成丰富的表征模型。

但是，如何表示各通道表征的模式和内容，并对其进行有效的量化分析，进而量化研究不同卷积核所能识别的模式及其表达，这仍然是可解释深度模型需要进一步研究的问题。

代码主要需要导入了右边的库和模块：

```
import torch
from torchvision import models
import numpy as np
import torchvision.transforms as transforms
from PIL import Image
import torch.nn as nn
import matplotlib.pyplot as plt
from torchvision.transforms.functional import to_pil_image
from matplotlib import cm
```



本章内容

contents

- 9.1 CNN各网络层输出的可视化
- 9.2 CNN模型决策原因的可视化方法
- 9.3 面向NLP任务的可视化方法

9.2 CNN 模型决策原因的可视化方法

CNN 模型主要用于对图像进行分类。当将输入图像被划分到一个类别时，这其实就是一个决策。在许多场合下，我们希望能够知道 CNN 模型为何做出这样的决策？它的决策依据和原因是什么？遗憾的是，模型本身不提供这样的信息，但我们可以在事后通过一定的方法，找到模型决策时所依赖输入图像的关键区域。

决策原因可视化：指对输入图像中对决策起关键作用的区域进行标注（用不同的颜色来区分），以让用户明白模型主要是根据哪一个区域进行决策的，从而为用户提供一定程度的解释。

决策原因可视化一般是利用类激活图（CAM）来实现，实现方法可分为基于类别权重的方法和基于梯度的方法。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

第4章也已经指出，一个深度神经网络一般是由两部分组成：卷积神经网络（CNN） 和全连接神经网络（FC）。前者是用于提取特征，后者则用于分类。也就是说，严格意义上讲，卷积神经网络（CNN）是不包含全连接神经网络的。本章提及的 CNN 均指此意义下的CNN，而 CNN 的输出则是指 CNN 最后一层的输出。对于 CNN 输出的特征图，当对其通道图像进行加权叠加并进行 ReLU 激活后，对得到的单通道特征图进行插值扩充，还原为跟输入图像一样大小，则非负值所在的区域即为输入图像中物体所在区域，这个区域就是模型决策的主要依据。当将扩充的单通道特征图叠加到输入图像上时，即得到高亮物体区域的新图，称为类激活图（Class Activation Map, CAM）；而实现类激活图的方法称为类激活映射（Class Activation Mapping, CAM）。这就是论文《Learning Deep Features for Discriminative Localization》[7]的主要贡献。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

在类激活映射中，各通道的权值的产生是非常关键的。根据论文[7]介绍的方法，先对 CNN 输出特征图的各个通道图像进行全局平均池化，这样每个通道图像在维度坍塌后就形成一个数值，然后将这些数值输入到一个全连接网络层进行分类。

假设这样的通道个数为 n，类别个数为 c，则该全连接层的参数矩阵 W 为 c×n 矩阵。

如果输入图像被分类为 y 类（y 为索引），则向量 W[y, :]即为通道图像的权重向量。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

【例 9.2】基于类别权重构建类激活图 CAM，进而实现决策原因的可视化。

本例以 resnet18 作为基本模型，介绍类激活图的构建方法。

根据上面的介绍，构建 CAM 图的一个关键是获取 CNN 部分的输出。例 9.1 通过“拆除”一个预训练模型，然后再将其“组装”而获得每一层的输出。但如果只是想要某一层的输出，那么用这种方法显得过于“繁杂”。

为此，下面先介绍一种用 register_forward_hook()函数通过网络层注册来获取某一层输入和输出的方法。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

register_forward_hook()函数可以为指定的某一网络层注册一个前向 hook（钩子）。当在调用 forward()函数而执行到该层时，这个 hook 会被调用并获得三个输入，分别是该网络层本身、该网络层的输入和输出（也可简单理解为：该 hook 会“勾住”这三个对象，而不“破坏”模型的网络结构）。

假如 layer 为一个网络层，我们先定义一个 hook：

```
def hook_fun(model, input, output):
... ...
```

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

然后为网络层 layer 注册该 hook：

handle = layer.register_forward_hook(hook_fun)

这样，在调用模型的 forward()函数而执行到网络层 layer 时，参数 model, input 和 output 将分别“截获”layer 本身、layer 的输入和输出。在本例中，我们主要是利用网络层的输出，因此只需参数 output。

在用完了以后可以去掉该 hook：

handle.remove()

使用 register_forward_hook()函数的优点是：在不需要“拆装”现有模型的情况下获得某一层的输入和输出。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

基于 resnet18 实现类激活图的方法：

（1）加载预训练模型 resnet18，然后定义一个 hook；接着用 print()查看 resnet18 的结构，发现其 CNN 部分的最后一个网络层是 model.layer4[1].bn2，因此该层的输出即为 CNN 部分的输出，于是为该网络层注册已定义的 hook。相关代码如下：

```
model = models.resnet18(pretrained=True) #加载模型
def hook_fun(model, input, output): #定义 hook
global out_FM #定义全局变量，用于存放输出的特征图
out_FM = output
handle = model.layer4[1].bn2.register_forward_hook(hook_fun) #注册一个前向 hook
```

注意，在执行了模型的 forward()方法以后，全局变量 out_FM 才被定义和赋值，而在此之前它是不存在的，因而也不能被引用，否则会出现运行报错误。

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

（2）读入图像，在经过适当变换后送入 resnet18 进行分类，代码如下：

```
tfs = transforms.Compose([transforms.Resize(256), transforms.CenterCrop(224),
                          transforms.ToTensor(),
                          transforms.Normalize([0.5, 0.5, 0.5], [0.2, 0.2, 0.2])])
path = r'./data/Interpretability/images'
name = 'both.png'
img_path = path + '\\' + name
img = Image.open(img_path).convert('RGB') #打开图片并转换为 RGB 模型
origin_img = img #保存原图
img = tfs(img) #转化为张量
img = torch.unsqueeze(img, 0) #增加 batch 维度 torch.Size([1, 3, 224, 224])
#下列语句在前向计算而执行到 model.layer4[1].bn2 层时会调用到 hook_fun()函数
#传入该函数的分别是 model.layer4[1].bn2 层本身以及该层的输入和输出
out = model(img)
#这时 out_FM 已经保存了 model.layer4[1].bn2 层的输出
pre_y = torch.argmax(out).item() #获取预测类别的索引
```

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

（3）实现通道图像的加权求和。在特征图从网络层 model.layer4[1].bn2 输出后，送入一个全局池化层AdaptiveAvgPool2d。经过该层后，每个通道图像都被平均池化，形成一个数值，一共有 512 个这样的数值。而后，再进入一个全连接层——model.fc 层。因此，model.fc.weight.data[pre_y,:]实际上是预测的类别 pre_y 到各个通道图像的权重向量，其中的每个分量值实际上就是相应通道连接到类别 pre_y 的边的权重，不妨称为类别权重。也就是说，以这 512 个类别权重作为对应通道图像的权重，然后进行加权求和。当然，在加权求和之前，还要做相应的形状变换。相关代码如下：

```
weights = model.fc.weight.data[pre_y,:] #获取类别权重, 作为通道加权时的权重
#对特征图的通道进行加权叠加，获得 CAM
# (512, 1, 1) * (512, 7, 7) ----> (512, 7, 7)
weights = weights.reshape(*weights.shape,1,1) # (512, 1, 1)
out_FM = out_FM.squeeze(0) #(512, 7, 7)
weighted_FM = (weights * out_FM).sum(0) #加权求和，结果形状为(7, 7)
handle.remove()
```

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

（4）构建类激活图 CAM。对加权后形成的单通道特征图 weighted_FM 运用 relu 激活函数，以保留非负值，并做一次平方运算，以降低小特征值对显示结果的干扰；然后通过双三次插值方法，将 weighted_FM 扩充为跟原图一样大小的数值矩阵；接着定义一个调色板，将 expanded_FM 转化为可视图，其中值大的像素显示为红色或深蓝色，值小的为浅蓝色;最后将 expanded_FM 覆盖到原图上，形成类激活图 CAM，并进行显示。相关代码如右：

```
#运用激活函数,仅保留非负值; 平方的目的是为了降低小特征值的干扰
weighted_FM = torch.relu(weighted_FM)**2
weighted_FM = (weighted_FM-weighted_FM.min()) \
                    /(weighted_FM.max()-weighted_FM.min()) #归一化
#将 weighted_FM 转换成 PIL 格式，以调用 resize()函数
weighted_FM = to_pil_image(np.array(weighted_FM.detach()), mode='F')
#通过插值，将加权求和后的单通道特征图扩充为跟原图一样大小
expanded_FM = weighted_FM.resize(origin_img.size, resample=Image.BICUBIC)
#运用调色板，将 expanded_FM 转化为可视图，其中值大的像素显示为红色或深蓝
#色，值小的为浅蓝色
expanded_FM = 255 * cm.get_cmap('jet')(np.array(expanded_FM))[:, :, 1:]
expanded_FM = expanded_FM.astype(np.uint8)
#将原图和可视化后的单通道特征图叠加（融合），形成类激活图 CAM
CAM = cv2.addWeighted(np.array(origin_img), 0.6, np.array(expanded_FM), 0.4, 0)
plt.imshow(CAM) #显示类激活图 CAM
plt.show()
```

9.2 CNN 模型决策原因的可视化方法

9.2.1 基于类别权重的类激活图（CAM）

执行由上述代码构成的.py 文件，结果输出的类激活图如图 9-7(a)所示（图 9-7(b)是原图）

从图 9-7(a)所示的类激活图看，resnet18 将原图（图 9-7(b)）判别为狗的图片，其主要是根据狗的嘴巴、颈部及其周围的特征。直觉告诉我们，这种决策依据似乎是可信的，因而这种类激活图为我们提供了一种直观的解释，即提供了决策原因的可视化解释。

(a)　　(b)

图 9-7　形成的类激活图

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

【例 9.3】基于梯度构建类激活图 CAM，进而实现决策原因的可视化。

- 本例则以 VGG16 作为基本模型，介绍使用梯度信息来构建类激活图的方法。
- 对于 VGG16，其 CNN 部分也可以视为由 features 部分和 AdaptiveAvgPool2d 部分组成，即把自适应平均池化层 AdaptiveAvgPool2d(output_size=(7, 7))也视为 CNN 部分。
- 这意味着，CNN 部分输出的特征图的形状恒定为 512×7×7。该特征图在经过扁平化处理后送入含有多个全连接层组成的分类网络，最后实现分类。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

问题：**我们如何为输出特征图的 512 个通道图像计算各自的权重？**

我们注意到，利用网络的最后预测输出，可计算相对于各通道图像中每个参数的导数；导数值越大，说明相应参数对分类的贡献越大，反之越小。

显然，如果一个通道包含做贡献大的参数越多，则该通道越重要，其权重也应该越大，反之越小。利用这种思路来构建各通道图像的权重，进而构建类激活图，这就是所谓的基于梯度的类激活图的构建方法。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

Pytorch 框架在反向传播过程中，默认不保留中间变量的梯度。但是，有时候我们又需要某一个中间变量的梯度信息（如上面这种情况），这时 register_hook()函数就派上用场了。

用 register_hook()函数可以为某一个中间变量注册一个 hook，此后每当求导“经过”该中间变量时，相对于该中间变量的导数会被该 hook“勾住”而被保留下来。

例如，假设给定函数：

$y = x2$,
$z = 4y$

那么，可以使用下列代码计算 z 关于 x 在 x=3 上的导数：

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

```
x = torch.tensor([3.], requires_grad=True)
y = torch.pow(x, 2)
z = 4*y
z.backward() #反向求导
print('z 关于 x 的导数为：', x.grad.item())
```

执行后输入结果如下：
z 关于 x 的导数为： 24.0
这与运用数学方法计算的结果是一样的。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

从数学上还可以推知，z 关于中间变量 y 的导数为 4。但是，由于 Pytorch 默认不保留中间变量 y 的导数，因此想通过“y.grad.item()”来获取这个导数，那是不行的。但可以用register_hook()函数来实现，代码如下：

```
def grad_hook(grad): #定义一个 hook
    global temp_grad #定义全局变量，用于存放梯度
    temp_grad = grad
    return None
x = torch.tensor([3.], requires_grad=True)
y = torch.pow(x, 2)
z = 4*y
handle = y.register_hook(grad_hook) #对 y 注册一个反向
hook
z.backward() #求导
print('z 相对于 y 的导数为：',temp_grad.item() )
handle.remove()
```

执行上述代码，输出结果为：
z 相对于 y 的导数为： 4.0
这与我们预想的是一致的。这说明，register_hook()函数能够截取指定中间变量的梯度。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

下面我们介绍本例的实现过程：

（1）加载 VGG16，然后定义一个前向 hook，进而为 model.avgpool 层注册该 hook（model.avgpool 层视为 CNN 部分的最后一层），当调用模型而执行 forward()方法时该 hook 会自动被调用而将 CNN 部分输出的特征图保存到全局变量 out_FM 中。相关代码如右：

```
model = models.vgg16_bn(pretrained=True) #加载 VGG16
model.eval()
def hook_fun(model, input, output): #定义前向 hook
    global out_FM #定义全局变量，用于存放输出的特征图
    out_FM = output
    return None
model.avgpool.register_forward_hook(hook_fun) #注册一个前向 hook
tfs = transforms.Compose([transforms.Resize(256), transforms.CenterCrop(224),
                          transforms.ToTensor(),
                          transforms.Normalize([0.5, 0.5, 0.5], [0.2, 0.2, 0.2])])
path = r'./data/Interpretability/images'
name = 'both.png'
img_path = path + '\\' + name
img = Image.open(img_path).convert('RGB') #打开图片并转换为 RGB 模型
origin_img = img #保存原图
img = tfs(img) #转化为张量
img = torch.unsqueeze(img, 0) #增加 batch 维度
out = model(img) #执行该语句后，out_FM 才产生
pre_y = torch.argmax(out).item() #预测类别
```

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

（2）定义一个后向 hook，并将之注册到上述代码执行后产生特征图 out_FM 上。代码如下：

```
def grad_hook(grad): #定义一个后向 hook
    global temp_grad #定义全局变量，用于存放梯度
    temp_grad = grad
    return None
out_FM.register_hook(grad_hook) #对 out_FM 注册一个反向 hook
```

注意，register_hook()函数用于对张量注册一个反向 hook，而register_forward_hook()函数是用于对一个网络层注册一个前向 hook。这也是它们在运用对象上的重要区别。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

（3）利用预测的索引 pre_y，获取网络的概率预测值 out[:, pre_y]，然后据此反向求导。由于我们已对 CNN 部分输出的特征图（一种中间结果）注册了一个反向 hook，因此在特征 图包含的参数上都得到相应的导数，形成跟特征图一样形状的张量，并保存到全局变量 temp_grad 中，最后根据该变量来构建各通道的权重，进而完成通道的加权求和。代码如下：

```
pre_class = out[:, pre_y]
pre_class.backward() #执行该语句后，temp_grad 才产生
#out_FM 和  temp_grad 的形状完全一样——torch.Size([1, 512, 7, 7])，
#但前者是特征图，后者是关于特征图的导数，是构造权重的依据
out_FM = out_FM[0] #torch.Size([512, 7, 7])
temp_grad = temp_grad[0] #torch.Size([512, 7, 7])
weights = torch.nn.AdaptiveAvgPool2d((1, 1))(temp_grad) #对各通道平均池化
#(512, 1, 1)*(512, 7, 7)-->(512, 7, 7)
weighted_FM = weights * out_FM
weighted_FM = weighted_FM.sum(0) #torch.Size([7, 7])
```

执行上述代码后，得到的weighted_FM就是各通道加权求和后的结果。

9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

（4）剩下的工作就是将 weighted_FM 扩展，然后覆盖到原图上，最后得到类激活图。相关代码如下（这些代码跟例 9.2 后半代码相同）：

```
weighted_FM = torch.relu(weighted_FM)**2
weighted_FM = (weighted_FM-weighted_FM.min()) \
                /(weighted_FM.max()-weighted_FM.min()) #归一化
#将 weighted_FM 转换成 PIL 格式，以条用 resize()函数
weighted_FM = to_pil_image(np.array(weighted_FM.detach()), mode='F')
expanded_FM = weighted_FM.resize(origin_img.size, resample=Image.BICUBIC)
expanded_FM = 255 * cm.get_cmap('jet')(np.array(expanded_FM))[:, :, 1:]
expanded_FM = expanded_FM.astype(np.uint8)
#将原图和可视化后的单通道特征图叠加（融合），形成类激活图 CAM
CAM = cv2.addWeighted(np.array(origin_img), 0.6, np.array(expanded_FM), 0.4, 0)
plt.imshow(CAM) #显示类激活图 CAM
plt.show()
```

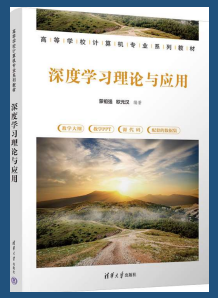
9.2 CNN 模型决策原因的可视化方法

9.2.2 基于梯度的类激活图（CAM）

执行由上述代码构成的.py 文件，产生如图 9-8 所示的结果。

直觉告诉我们，利用梯度构建的类激活图的效果也很好，似乎更能聚焦关键部位，而且这种方法可以适用于更多类型的网络结构。

图 9-8 利用梯度构建的类激活图



本章内容

contents

- 9.1 CNN各网络层输出的可视化
- 9.2 CNN模型决策原因的可视化方法
- 9.3 面向NLP任务的可视化方法

9.3 面向 NLP 任务的可视化方法

9.3.1 NLP 任务中注意力机制可视化的一般方法

一般情况下，当我们向 NLP 模型输入一个句子以后，模型就需要结合这个句子中的词（或 token）和自身的“知识”做出决策（如分类等）。显然，句子中不同的词对决策的形成有 不同程度的影响。注意力机制的一般做法是，设计一个网络来为句子中的每个词预测相应的权重参数，使得一个词的影响程度越大，它的权重就越大。

可视化的目的就是通过视觉的方式让用户快速感知哪些词的影响最大？哪些次之？哪些几乎没有作用？等。其中，常用的视觉标记方式就是采用不同深浅的颜色（如红色）来标 注有不同影响程度的词，即影响程度越大的词，用越深的红颜色来标注；影响程度越小的词，则用越浅的红颜色来标记。

9.3 面向 NLP 任务的可视化方法

9.3.1 NLP 任务中注意力机制可视化的一般方法

假设输入的句子是“努力实现中华民族的伟大复兴”，分词后得到如下的词序列：

```
words = ['努力', '实现', '中华', '民族', '的', '伟大', '复兴']
```

并假设注意力网络生成如下的权重：

```
att_weights = [0.05, 0.10, 0.20, 0.05, 0.00, 0.20, 0.40]
```

从权重看，'复兴'的权重最大，为 0.40，其次是'中华'和'伟大'的权重，均为 0.20，的是'的'的权重为0。

9.3 面向 NLP 任务的可视化方法

9.3.1 NLP 任务中注意力机制可视化的一般方法

按如图 9-9 所示采用不同深浅的红颜色来标注，用户可以一目了然（这在学术论文中也经常用到）

努力 实现 中华 民族 的 伟大 复兴

图 9-9 词的注意力机制可视化效果

9.3 面向 NLP 任务的可视化方法

9.3.2 自注意力机制的可视化

依赖 Transformer 框架的 BERT 和 GPT 等 NLP 处理模型基本上都是由多层自注意力网 络堆叠而成。例如，在比较小的 BERT 模型中，有 12 个网络层（由 12 个 encoder 堆叠而成），每层有 12 个自注意力网络，词向量维度是 768。

自注意力刻画了词（token）与词之间的关系；多头自注意力可以刻画不同类型的关系，从而可以表达丰富的语义。

因此，对自注意力的可视化有助于理解基于 Transformer 框架的预训练模型对词之间关系的学习能力。

9.3 面向 NLP 任务的可视化方法

9.3.2 自注意力机制的可视化

HuggingFace 网站（https://huggingface.co）针对多种预训练模型，提供了自注意力机制的在线可视化工具。打开链接 https://huggingface.co/exbert，可以看到如图 9-10 所示的界面。

HuggingFace 可视化工具主要是对注意力权重进行可视化，体现在⑧处线条的粗细上，即注意力权重越大，则对应的线条越粗。在介绍各线条具体意义之前，我们先说明一些基本设置。

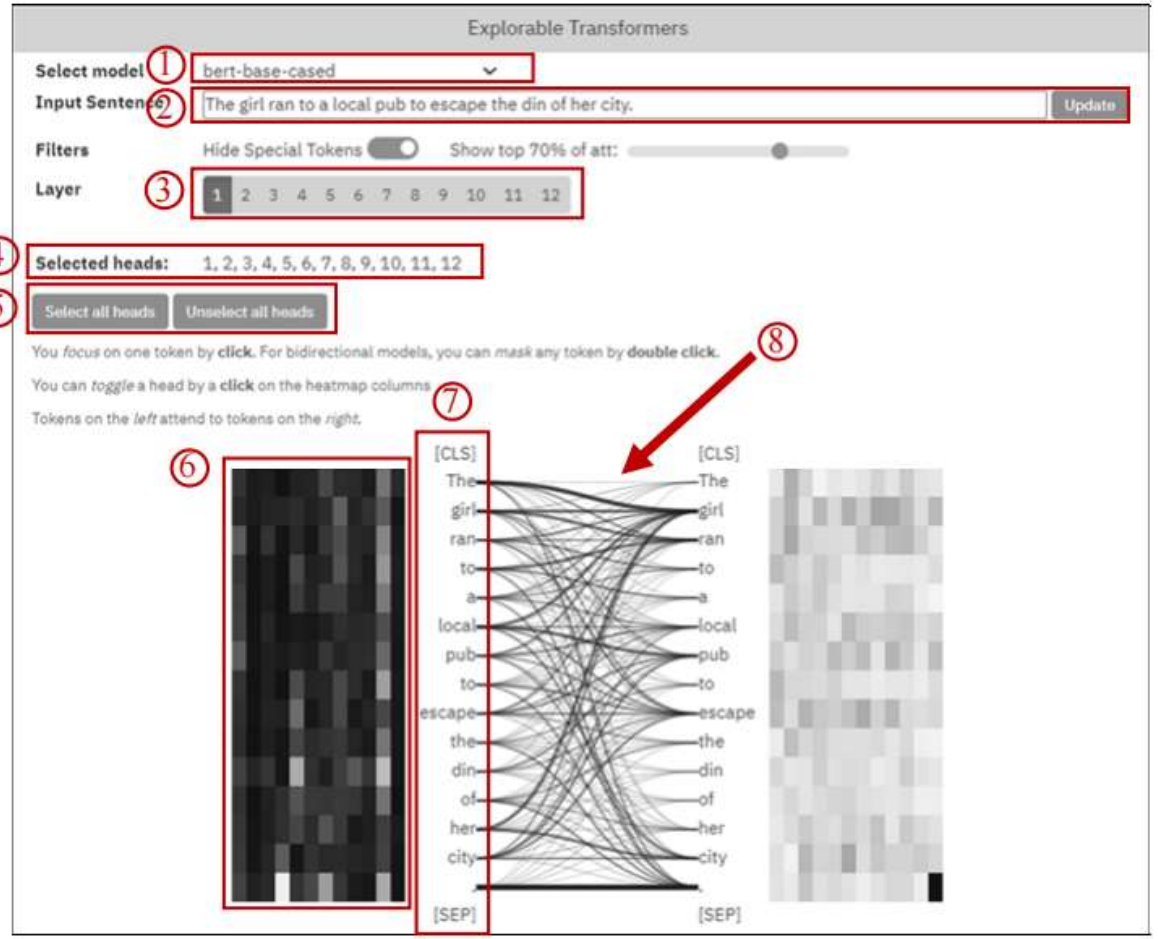


图 9-10　HuggingFace 可视化工具界面

9.4 本章小结

本章内容：

- 特征图进行可视化
- 用类激活图（CAM）
- 面向 NLP 任务的可视化方法