小作业

1.打印模型结构

相关代码

```
from torchvision import models
import torch

resnet18 = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
print(resnet18)
```

输出结果

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
```

```
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
```

```
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

---

2. 图片分类

相关代码

```python
# 检查并创建文件夹
def chec_folders():
    if not os.path.exists("classify_img"):
        os.makedirs("classify_img")
        print("Created 'classify_img' folder.")
```

```
# 定义图像变换
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# 加载 ImageNet 的类别名称
with open('imagenet_classes.txt') as f:
    classes = [line.strip() for line in f.readlines()]

def open_files():
    global avg_color_PIL, avg_color_opencv, top5_classes, top5_prob, img, img_cv
    check_folders()
    for img_name in os.listdir("input_img"):
        if img_name.lower().endswith((".png", ".jpg", ".jpeg")):
            img_path = os.path.join("input_img", img_name)
            img = Image.open(img_path).convert("RGB")
            img_cv = cv2.imread(img_path)
            img_cv = cv2.cvtColor(img_cv, cv2.COLOR_BGR2RGB)
            classify_image(img_path)  # 调用分类函数
            #calculate_brightness()   # 调用计算平均亮度函数
            #draw_and_save(img_path)  # 调用绘图和保存函数

def classify_image(img_path):
    global top5_classes, top5_prob
    img_transformed = transform(img).unsqueeze(0)  # 一次处理一张图片
    with torch.no_grad():
        outputs = resnet18(img_transformed)
        probabilities = F.softmax(outputs, dim=1)[0]
        top5_prob, top5_indices = torch.topk(probabilities, 5)
    top5_classes = [classes[idx] for idx in top5_indices]
```

---

3. 拓展:实现任务二

相关代码

```
def calculate_brightness():
    global avg_color_PIL, avg_color_opencv, img, img_cv
    img_np = np.array(img)
    img = torch.tensor(img_np, dtype=torch.float32) / 255.0
    img_cv = torch.tensor(img_cv, dtype=torch.float32) / 255.0
    avg_color_PIL = np.array(img).mean(axis=(0, 1))  # 使用 PIL 计算平均颜色
    avg_color_PIL = avg_color_PIL.tolist()
    avg_color_opencv = img_cv.mean(axis=(0, 1))  # 使用 OpenCV 计算平均颜色
    avg_color_opencv = avg_color_opencv.tolist()
```

---

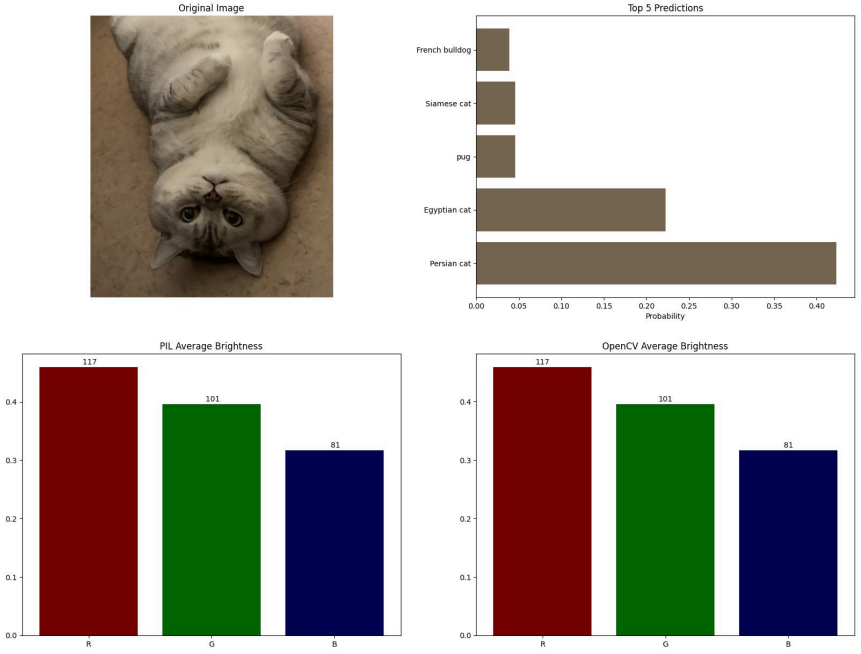
4. 绘制图像

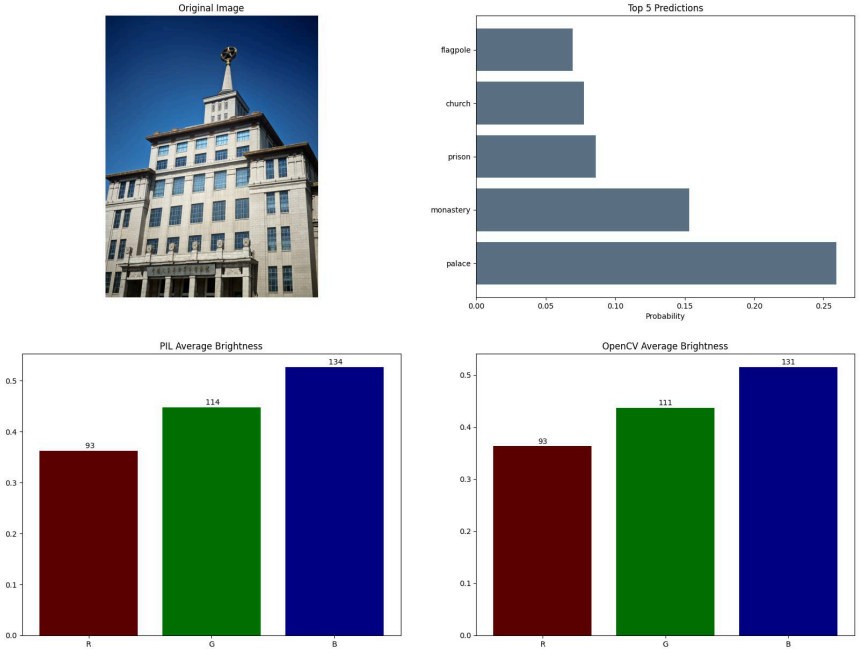
- 将原图像、分类结果、两种方式计算的平均亮度打印在一张图中

相关代码

```
def draw_and_save(img_path):
    global avg_color_PIL, avg_color_opencv, top5_classes, top5_prob, img, img_cv
    avg_color = [(i + j) / 2 for i, j in zip(avg_color_PIL, avg_color_opencv)]

    plt.figure(figsize=(20, 15))

    # 左上原始图像
    plt.subplot(2, 2, 1)
    plt.imshow(img)
    plt.axis("off")
    plt.title("Original Image")

    # 右上 Top 5 Predictions 图
    plt.subplot(2, 2, 2)
    avg_color = [(i + j) / 2 for i, j in zip(avg_color_PIL, avg_color_opencv)]
    plt.barh(top5_classes, top5_prob, color=avg_color)  # 使用默认颜色
    plt.xlabel("Probability")
    plt.title("Top 5 Predictions")

    # 左下 PIL 平均亮度柱状图
    plt.subplot(2, 2, 3)
    plt.title("PIL Average Brightness")
    channels = ['R', 'G', 'B']
    plt.bar(channels, avg_color_PIL, color=[(avg_color_PIL[0], 0, 0), (0,
avg_color_PIL[1], 0), (0, 0, avg_color_PIL[2])])
    for i, v in enumerate(avg_color_PIL):
        plt.text(i, v + 0.005, f"{v:.4f}", ha='center')

    # 右下 OpenCV 平均亮度柱状图
    plt.subplot(2, 2, 4)
    plt.title("OpenCV Average Brightness")
    plt.bar(channels, avg_color_opencv, color=[(avg_color_opencv[0], 0, 0), (0,
avg_color_opencv[1], 0), (0, 0, avg_color_opencv[2])])
    for i, v in enumerate(avg_color_opencv):
        plt.text(i, v + 0.005, f"{v:.4f}", ha='center')

    # 保存结果图像
    img_name = os.path.basename(img_path)
    save_name = f"{os.path.splitext(img_name)[0]}_{top5_classes[0]}.jpg"
    save_path = os.path.join("classify_img", save_name)
    plt.savefig(save_path)
    plt.close()
    print(f"Processed and saved: {save_name}")
```

5. 效果展示

- 图一
  - 
Original Image
Top 5 Predictions
French bulldog
Siamese cat
pug
Egyptian cat
Persian cat
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40
Probability
PIL Average Brightness
117
101
81
R
G
B
OpenCV Average Brightness
117
101
81
R
G
B

- 图二
  - 
Original Image
Top 5 Predictions
flagpole
church
prison
monastery
palace
0.00 0.05 0.10 0.15 0.20 0.25
Probability
PIL Average Brightness
93
114
134
R
G
B
OpenCV Average Brightness
93
111
131
R
G
B

6. 总结

- 完成了任务1 **打印网络结构**和**图像分类**，并用图片平均颜色的柱状图直观展示前五可能的分类结果
- 融合了任务2的内容，用`PIL`和`OpenCV`计算图片各通道的平均亮度，并用颜色对应的柱状图直观地打印出来
    - 多数情况下，`PIL`计算结果比`OpenCV`的结果略微偏大
- 代码分模块管理，不同的功能由不同的函数实现，减少相互干扰，便于排查问题
- 较好的规范了文件管理与命名,输入图像、分类/计算结果图像 分别储存在不同的子文件夹中，结果图像命名为*原名称_最有可能的分类*，便于索引和管理

---

7. 附: 完整代码

```
import os
import numpy as np
from PIL import Image
import cv2
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torchvision import models, transforms
# 全局变量
global avg_color_PIL, avg_color_opencv, top5_classes, top5_prob, img, img_cv


# 加载预训练的 ResNet18 模型
resnet18 = models.resnet18(pretrained=True)
resnet18.eval()
#print(resnet18)

# 检查并创建文件夹
def check_folders():
    if not os.path.exists("classify_img"):
        os.makedirs("classify_img")
        print("Created 'classify_img' folder.")

# 定义图像变换
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# 加载 ImageNet 的类别名称
with open('imagenet_classes.txt') as f:
    classes = [line.strip() for line in f.readlines()]
def check_folders():
    if not os.path.exists("input_img"):
        os.makedirs("input_img")
    if not os.path.exists("classify_img"):
        os.makedirs("classify_img")
```

```
def open_files():
    global avg_color_PIL, avg_color_opencv, top5_classes, top5_prob, img, img_cv
    check_folders()
    for img_name in os.listdir("input_img"):
        if img_name.lower().endswith((".png", ".jpg", ".jpeg")):
            img_path = os.path.join("input_img", img_name)
            img = Image.open(img_path).convert("RGB")
            img_cv = cv2.imread(img_path)
            img_cv = cv2.cvtColor(img_cv, cv2.COLOR_BGR2RGB)
            classify_image(img_path)  # 调用分类函数
            calculate_brightness()   # 调用计算平均亮度函数
            draw_and_save(img_path)  # 调用绘图和保存函数


def classify_image(img_path):
    global top5_classes, top5_prob
    img_transformed = transform(img).unsqueeze(0)  # 一次处理一张图片
    with torch.no_grad():
        outputs = resnet18(img_transformed)
        probabilities = F.softmax(outputs, dim=1)[0]
        top5_prob, top5_indices = torch.topk(probabilities, 5)
    top5_classes = [classes[idx] for idx in top5_indices]


def calculate_brightness():
    global avg_color_PIL, avg_color_opencv, img, img_cv
    img_np = np.array(img)
    img = torch.tensor(img_np, dtype=torch.float32) / 255.0
    img_cv = torch.tensor(img_cv, dtype=torch.float32) / 255.0
    avg_color_PIL = np.array(img).mean(axis=(0, 1))  # 使用 PIL 计算平均颜色
    avg_color_PIL = avg_color_PIL.tolist()
    avg_color_opencv = img_cv.mean(axis=(0, 1))  # 使用 OpenCV 计算平均颜色
    avg_color_opencv = avg_color_opencv.tolist()


def draw_and_save(img_path):
    global avg_color_PIL, avg_color_opencv, top5_classes, top5_prob, img, img_cv
    avg_color = [(i + j) / 2 for i, j in zip(avg_color_PIL, avg_color_opencv)]

    plt.figure(figsize=(20, 15))

    # 左上原始图像
    plt.subplot(2, 2, 1)
    plt.imshow(img)
    plt.axis("off")
    plt.title("Original Image")

    # 右上 Top 5 Predictions 图
    plt.subplot(2, 2, 2)
    avg_color = [(i + j) / 2 for i, j in zip(avg_color_PIL, avg_color_opencv)]
    plt.barh(top5_classes, top5_prob, color=avg_color)  # 使用默认颜色
    plt.xlabel("Probability")
    plt.title("Top 5 Predictions")

    # 左下 PIL 平均亮度柱状图
```

```
    plt.subplot(2, 2, 3)
    plt.title("PIL Average Brightness")
    channels = ['R', 'G', 'B']
    plt.bar(channels, avg_color_PIL, color=[(avg_color_PIL[0], 0, 0), (0,
avg_color_PIL[1], 0), (0, 0, avg_color_PIL[2])])
    for i, v in enumerate(avg_color_PIL):
        plt.text(i, v + 0.005, f"{v:.4f}", ha='center')

    # 右下 OpenCV 平均亮度柱状图
    plt.subplot(2, 2, 4)
    plt.title("OpenCV Average Brightness")
    plt.bar(channels, avg_color_opencv, color=[(avg_color_opencv[0], 0, 0), (0,
avg_color_opencv[1], 0), (0, 0, avg_color_opencv[2])])
    for i, v in enumerate(avg_color_opencv):
        plt.text(i, v + 0.005, f"{v:.4f}", ha='center')

    # 保存结果图像
    img_name = os.path.basename(img_path)
    save_name = f"{os.path.splitext(img_name)[0]}_{top5_classes[0]}.jpg"
    save_path = os.path.join("classify_img", save_name)
    plt.savefig(save_path)
    plt.close()
    print(f"Processed and saved: {save_name}")


if __name__ == "__main__":
    open_files()
```