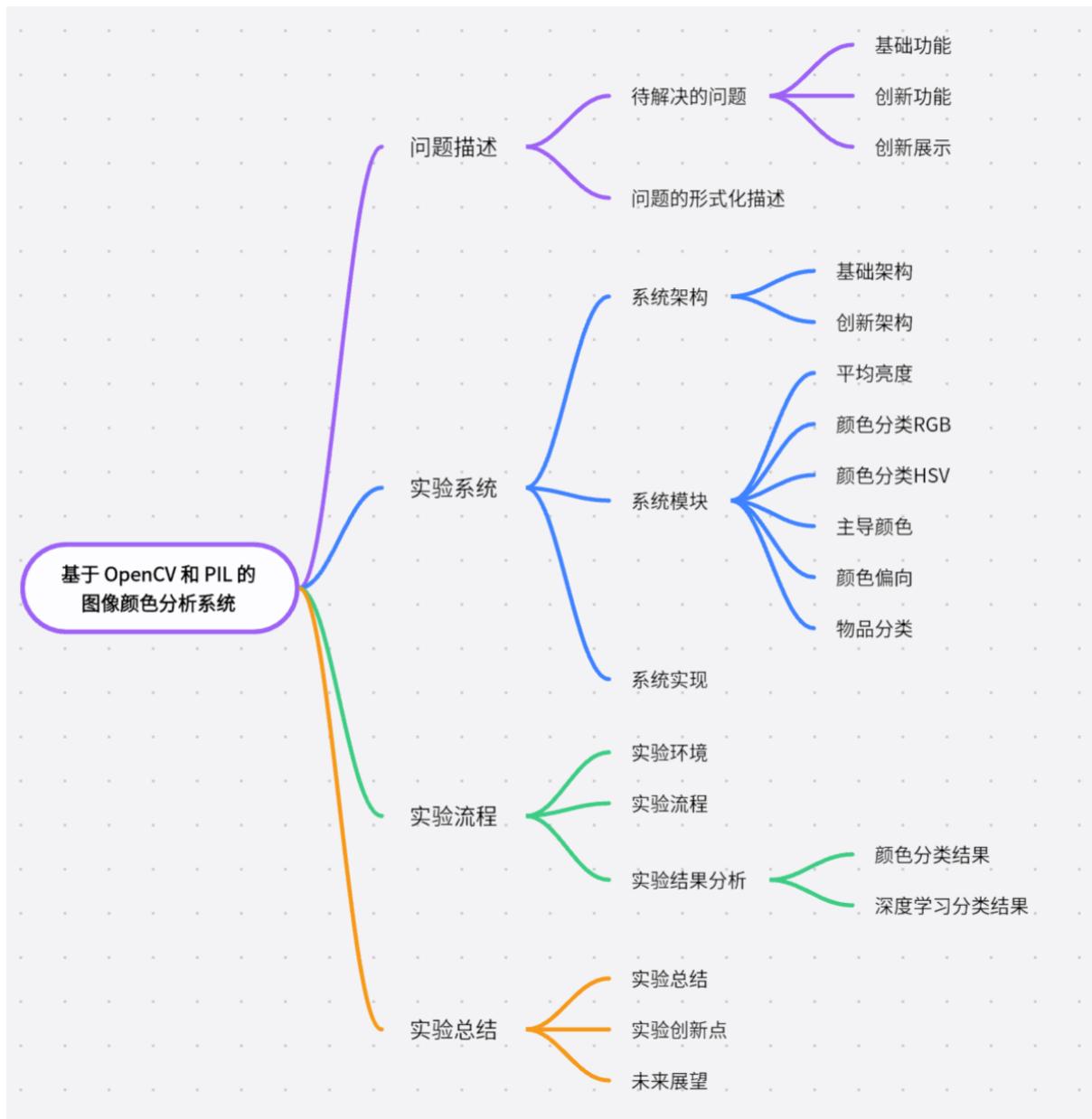


基于 OpenCV 和 PIL 的图像颜色分析系统

2024年北京邮电大学智能信息网络课程小实验 实验二

成员：汪雨生——2022211739

摘要：本实验主要探讨了基于图像平均亮度的颜色分类方法，并结合深度学习模型进行物体识别。在实验过程中，我使用了 OpenCV 和 PIL 读取图像，计算其 RGB 三通道的平均亮度，并基于设定的规则判断图像的主导颜色。此外，关于实验的创新方面：采用 HSV 颜色空间进行颜色分类，并使用 K-Means 算法分析图像的主导颜色，并且计算了颜色在RGB三种颜色中的偏向。为了进一步拓展实验功能，我还加载 PyTorch 预训练的 AlexNet 和 VGG16 模型（结合实验一），对图像中的物体类别进行分类。本实验的结果表明，不同图像处理库在颜色分类任务上的一致性较高，而结合深度学习模型的图像分类方法在识别物体类别方面具有良好的表现。



一、问题描述

任务背景：

通过手机或者网络获取几张红色、蓝色、绿色物品的照片，加载图像，转换为张量；同时对于每个图像张量，使用mean()函数计算图像各个通道平均亮度；通过平均亮度人工判断出红色、蓝色、绿色图像；用opencv(cv2)和PIL进行图像读取，看看计算出的平均亮度有什么区别。

除此之外，我还进行了创新任务设计，包括基于HSV颜色空间的颜色分类、基于K-Means聚类的主导颜色计算、颜色偏向计算和结合实验一的基于AlexNet和VGG16的物品分类。

1.1 待解决的问题

任务目标

- 目标：开发一个基于图像颜色分析与深度学习分类的系统，能够对输入的图像进行基于平均亮度等多种方法的颜色分类分析、颜色偏向分析以及识别其中的物体类别。
- 涉及领域：计算机视觉、数字图像处理、深度学习。

在数字图像处理领域，准确判断图像的主导颜色以及对图像中的物体进行分类是一项具有挑战性的任务。本实验旨在解决以下几个问题：

基础功能

- a. 使用 OpenCV 和 PIL 计算图像的平均亮度；
- b. 通过人工方式对图像颜色进行判别。

创新功能

- a. 机器自动判别颜色：基于 RGB 三通道亮度计算颜色分类；
- b. 基于 HSV 进行颜色分类：利用 HSV 颜色空间判断颜色类别；
- c. K-Means 计算主导颜色：通过 K-Means 聚类算法分析图像的主导颜色；
- d. RGB 值计算颜色偏向：分析颜色偏向的方向，如偏红、偏蓝或偏绿；
- e. 基于预训练模型进行物品分析：导入 AlexNet 和 VGG16 预训练模型，对非纯色图片进行物品分类。

创新展示

- a. 采用前后端分离架构，前端通过网页交互，后端处理图像并返回分析结果；
- b. 在网页上展示图片的平均亮度、颜色分类、主导颜色、颜色偏向、物品分类等信息。

1.2 问题的形式化描述

给定一张图像 I ，通过计算其 RGB 三个通道的平均亮度值 ($R_{avg}, G_{avg}, B_{avg}$)，使用 OpenCV 和 PIL 分别读取图像并计算平均亮度，比较两者在图像处理流程中的差异和结果一致性；同时，根据设定的阈值和规则判断图像的主导颜色 Color 和颜色分类；最后，利用预训练的深度学习模型 Model 对图像进行分类，得到图像中所包含物体的类别标签 Label。

给定一张输入图像 I ，我们的目标是：

1. 颜色主导性分析：

- 计算 RGB 三个通道的平均亮度值：($R_{avg}, G_{avg}, B_{avg}$)。
- 根据设定的阈值和规则，判断图像的主导颜色 Color 并且对颜色进行分类。

2. 对比不同图像处理库：

- 使用 OpenCV 和 PIL 计算图像亮度，并比较两者在图像处理流程中的差异和一致性。

3. 基于 HSV 进行颜色分类：

- 计算 HSV 色彩空间的均值 H_{avg} ，用于更精确的颜色判断。

4. 主导颜色提取 (K-Means 聚类)：

- 通过 K-Means 聚类分析图像的主要颜色分布。

5. 颜色偏向分析：

- 计算图像颜色偏向，确定图像是红色、绿色、蓝色偏向还是多色图像。

6. 深度学习模型分类：

- 使用预训练模型 (AlexNet 或 VGG16) 进行物品识别，输出物体类别标签 Label。

最终目标：通过计算机视觉方法全面分析图像颜色信息，并展示在网页前端。

二、实验系统

实验系统部分，我分为三个主要方面：

1. 系统架构部分：包括图像读取模块，图像预处理模块，颜色分析模块，图像分类模块和结果展示模块；
2. 系统模块部分：包括计算图像的平均亮度、图像的颜色分类、主导颜色计算、物品分类四个主要模块；
3. 系统实现部分：包括颜色分析实现（颜色分类和偏向）、物品分类实现和创新功能中的前后端功能实现。

2.1 系统架构

2.1.1 基础架构 (main.py)

本系统基于 Python 编程语言，利用 OpenCV、PIL、PyTorch 等库实现。

基础架构主要实现系统的核心功能模块，为后续的创新架构提供基础支持，具体包括以下几个部分：

1. 图像读取模块

- 使用 OpenCV 和 PIL 分别加载图像。
- OpenCV 读取的图像是 BGR 格式，需要转换为 RGB 格式以便后续处理。
- PIL 读取的图像默认为 RGB 格式，不需要特别转换格式。

▼ 代码块

```
1 # 使用 OpenCV 加载图像
2 def load_image_opencv(image_path):
3     """使用 OpenCV 加载图像并转换为 RGB"""
4     image = cv2.imread(image_path)
5     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
6     return np.array(image_rgb)
7
8 # 使用 PIL 加载图像
9 def load_image_pil(image_path):
10    """使用 PIL 加载图像"""
11    image = Image.open(image_path).convert("RGB")
12    return np.array(image)
```

2. 图像预处理模块

- 对图像进行调整大小、裁剪、归一化等操作。
- 为后续的颜色分析和分类做准备。

▼ 代码块

```
1 # 预处理步骤 (适用于ImageNet预训练模型)
2 transform = transforms.Compose([
3     transforms.Resize(256),    # 调整图像大小
4     transforms.CenterCrop(224), # 中心裁剪
5     transforms.ToTensor(),    # 转换为张量
6     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
7         0.225]),   # 归一化
8 ])
```

3. 颜色分析模块

- 计算图像的平均亮度、主导颜色、颜色偏向等信息。
 - 计算 RGB 平均亮度并判断主导颜色。
 - 使用 K-Means 算法计算图像的主导颜色。
 - 将图像转换到 HSV 颜色空间，根据色调（Hue）、饱和度（Saturation）和亮度（Value）的均值进行颜色分类。
 - 计算颜色偏向度，判断图像颜色的偏向情况。

▼ 代码块

```
1 # 下面是实验的颜色分析模块的 四个功能 的主要代码
2 def classify_image_color(brightness): """基于亮度判断主导颜色"""
3 def get_dominant_color(image, k=3): """使用 K-Means 计算主导颜色"""
4 def classify_color_hsv(image): """使用 HSV 颜色空间进行颜色分类"""
5 def get_color_bias(dominant_color): """计算颜色偏向"""
```

4. 图像分类模块

- 加载 PyTorch 中预训练的 AlexNet 或 VGG16 模型。
- 使用 ImageNet 数据集的标签对图像进行分类，得到图像中物体的类别名称。

▼ 代码块

```
1 def classify_image(model, image_path):
2     """使用预训练模型进行图像分类"""
3     image = Image.open(image_path).convert("RGB")
4     image_tensor = transform(image).unsqueeze(0)
5
6     model.eval()
7     with torch.no_grad():
8         outputs = model(image_tensor)
9         _, predicted = torch.max(outputs, 1)
10
11     # 加载 ImageNet 标签
12     labels_path = os.path.join(os.path.dirname(__file__),
13                               "imagenet_classes.txt")
14     if not os.path.exists(labels_path):
15         import urllib.request
16         url =
17             "https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt"
18         urllib.request.urlretrieve(url, labels_path)
19
20     with open(labels_path) as f:
21         labels = [line.strip() for line in f.readlines()]
```

```
20
21     return labels[predicted.item()]
22
23
24     # 主函数中显示的模型选择如下:
25     # 选择模型
26     model_name = input("请选择模型 (alexnet/vgg16) : ").strip().lower()
27     if model_name == "alexnet":
28         model = models.alexnet(weights=AlexNet_Weights.DEFAULT)
29     elif model_name == "vgg16":
30         model = models.vgg16(weights=VGG16_Weights.DEFAULT)
31     else:
32         print("不支持的模型! 默认使用 AlexNet")
33     model = models.alexnet(pretrained=True)
```

5. 结果展示模块

- 将计算得到的平均亮度、主导颜色、颜色偏向以及分类结果以格式化的文本形式输出。
- 方便用户查看和理解。

▼ 代码块

```
1  for image_path in image_paths:
2      print("\n===== 计算结果 =====")
3      print(f"图像文件: {image_path}")
4
5      # 使用 OpenCV 加载图像
6      image_opencv = load_image_opencv(image_path)
7
8      # 使用 PIL 加载图像
9      image_pil = load_image_pil(image_path)
10
11     # 计算亮度
12     brightness_cv = calculate_brightness(image_opencv, normalize)
13     brightness_pil = calculate_brightness(image_pil, normalize)
14
15     print(f"是否归一化: {'是' if normalize else '否'}")
16     print(f"OpenCV 计算的平均亮度: {brightness_cv}")
17     print(f"PIL 计算的平均亮度: {brightness_pil}")
18
19     # 颜色判断
20     color_cv = classify_image_color(brightness_cv)
21     color_pil = classify_image_color(brightness_pil)
22
23     print(f"主导颜色 (OpenCV): {color_cv}")
24     print(f"主导颜色 (PIL): {color_pil}")
```

```

25     print(f"是否多色图像: {'是' if color_cv == '多色图像' else '否'}")
26
27     # HSV 颜色分类
28     hsv_color = classify_color_hsv(image_opencv)
29     print(f"HSV 颜色分类: {hsv_color}")
30
31     # 计算主导颜色
32     dominant_color = get_dominant_color(image_opencv)
33     print(f"主导颜色 (RGB): {dominant_color}")
34
35     # 计算颜色偏向
36     color_bias = get_color_bias(dominant_color)
37     print(f"颜色偏向度: {color_bias}")
38
39     # 进行图像分类
40     predicted_label = classify_image(model, image_path)
41     print(f"识别的物体分类: {predicted_label}")
42

```

2.1.2 创新架构 (app.py 和 app.js)

创新架构在基础架构的基础上，构建了一个完整的前后端系统，提升了用户体验和系统的实用性。

本实验的系统采用 Flask 作为后端，HTML + JavaScript（React）作为前端，并使用 TorchVision 预训练模型进行物品分类。系统架构如下：

前端 (app.js)

- 使用 HTML + JavaScript（React）构建。
- 负责图片上传，并发送请求到后端。
- 显示分析结果，包括平均亮度、颜色分类、主导颜色、颜色偏向、物品分类等。

▼ 代码块

```

1 // 使用 useState 管理图片、预览、分析结果和图表数据的状态
2 const [image, setImage] = useState(null); // 图片
3 const [preview, setPreview] = useState(null); // 预览
4 const [result, setResult] = useState(null); // 分析结果
5 const [multipleColors, setMultipleColors] = useState(null); // 多色分析结果
6 const [meanBrightnessPIL, setMeanBrightnessPIL] = useState(null); // PIL的
平均亮度
7 const [meanBrightnessOpenCV, setMeanBrightnessOpenCV] = useState(null);
// OpenCV的平均亮度
8 const [chartData, setChartData] = useState(null); // 图表数据
9 const [brightnessData, setBrightnessData] = useState(null); // 亮度数据
10 const [normalize, setNormalize] = useState(false); // 默认不标准化
11 const [dominantColor, setDominantColor] = useState(null); // 主导颜色

```

```
12     const [hsvColor, setHsvColor] = useState(null); // HSV分类
13     const [colorBias, setColorBias] = useState(null); // 颜色偏向度
14     const [showDominantColor, setShowDominantColor] = useState(false); // 控制
    显示主导颜色
15     const [showHsvColor, setShowHsvColor] = useState(false); // 控制显示HSV分类
16     const [showColorBias, setShowColorBias] = useState(false); // 控制显示颜色偏
    向度
```

后端 (app.py)

- 使用 Flask 构建。
- 处理前端上传的图像。
- 调用基础架构中的功能模块，计算平均亮度，并进行颜色判别。
- 通过 K-Means、HSV 颜色空间、RGB 计算等方法分析颜色信息。
- 可以选择使用 AlexNet 和 VGG16 进行物品分类。
- 返回分析结果给前端。

▼ 代码块

```
1     # 允许前端返回的结果
2     return jsonify({
3         "filename": file.filename, # 文件名
4         "normalize": normalize, # 是否归一化
5         "mean_brightness_opencv": brightness_cv2, # 平均亮度: OpenCV
6         "mean_brightness_pil": brightness_pil, # 平均亮度: PIL
7         "color_classification_opencv": color_cv2, # 颜色分类: OpenCV
8         "color_classification_pil": color_pil, # 颜色分类: PIL
9         "multiple_colors": multiple_colors, # 是否多色图像
10        "dominant_color_rgb": dominant_color.tolist(), # 主要颜色 (RGB)
11        "dominant_color_hsv": hsv_color, # 主要颜色分类 (HSV)
12        "color_bias": color_bias, # 颜色偏向
13        "classification_results": {
14            "alexnet": classification_results["alexnet"], # 分类结果: AlexNet
15            "vgg16": classification_results["vgg16"] # 分类结果: VGG16
16        }
17    }), 200
18
```

创新架构通过前后端的分离与协作，使得系统更加模块化和易于扩展，同时也提升了用户交互的友好性和效率。

2.2 系统模块

本系统包含多个模块，各模块的核心功能如下：

模块	功能
图像读取	采用 OpenCV 和 PIL 两种方法读取图片，确保格式一致性
图像预处理	归一化、调整大小、转换为张量，适配颜色分析和 CNN 处理
颜色分析	计算 RGB 亮度，K-Means 聚类分析，HSV 颜色分类，颜色偏向分析
物品分类	采用 AlexNet 或者 VGG16 进行物体识别分类
结果展示	后端计算结束后通过前端网页展示分析结果

- 1. 图像读取模块：**通过 OpenCV 的 `imread` 函数和 PIL 的 `Image.open` 方法分别读取图像文件。OpenCV 读取的图像是 BGR 格式，需要转换为 RGB 格式以便后续处理，而 PIL 读取的图像默认为 RGB 格式。双重读取方式确保图像数据的正确获取，为后续处理奠定基础。
- 2. 图像预处理模块：**使用 PyTorch 的 `torchvision.transforms` 对图像进行标准化的预处理操作，具体步骤包括调整图像大小至 256×256 像素，进行中心裁剪为 224×224 像素，转换图像为张量格式，并按照 ImageNet 数据集的均值和标准差进行归一化。这些操作确保图像符合预训练模型的输入要求，提升模型对图像的处理效果。
- 3. 颜色分析模块：**
 - K-Means 主导颜色计算：**将图像像素点聚类为 K 类（代码中分为 3 簇），找出数量最多的类别对应的中心颜色作为主导颜色。通过 K-Means 聚类算法，系统能够识别图像中占据主要部分的颜色，为图像的整体色彩特征提供重要信息。
 - HSV 颜色分类：**根据 HSV 空间中 色调 (Hue)、饱和度 (Saturation) 和 亮度 (Value) 的均值，结合设定的阈值，判断图像的大概颜色类别。HSV 空间能够更直观地反映人眼对颜色的感知，系统据此对图像进行分类，进一步细化颜色特征描述。
 - 颜色偏向分析：**通过比较各通道亮度与另外两通道亮度平均值的差值，确定颜色的偏向情况。这一分析能够揭示图像颜色在红、绿、蓝三色中的偏向，为用户提供更细致的颜色特征描述。
- 4. 图像分类模块：**利用 PyTorch 提供的预训练模型（代码中使用 AlexNet 和 VGG16），结合 ImageNet 数据集的标签，实现对图像中物体的快速准确分类。预训练模型在大规模数据集上进行了训练，具有强大的特征提取和分类能力，能够准确识别图像中的物体类别。
- 5. 结果展示模块：**以清晰的文本格式输出各项计算结果和分类结果，包括图像的平均亮度、主导颜色、颜色偏向以及物体分类等信息。而在创新展示功能中，则以网页前端的形式展现图片处理后的各个数据。这种展示方式便于用户直观了解图像的颜色特征和内容信息，提升用户体验。

▼ 代码块

```
1 # 输入：图像路径 image_path, 是否归一化 normalize, 模型名称 model_name
```

```
2
3 # 1. 使用 OpenCV 和 PIL 分别读取图像:
4     image_cv = cv2.imread(image_path)
5     image_pil = Image.open(image_path).convert("RGB")
6
7 # 2. 计算 OpenCV 和 PIL 图像的平均亮度:
8     brightness_cv = calculate_brightness(image_cv, normalize)
9     brightness_pil = calculate_brightness(image_pil, normalize)
10
11 # 3. 根据平均亮度判断主导颜色:
12     color_cv = classify_image_color(brightness_cv)
13     color_pil = classify_image_color(brightness_pil)
14
15 # 4. 将 OpenCV 图像转换为 HSV 颜色空间, 计算 HSV 颜色分类:
16     hsv_color = classify_color_hsv(image_cv)
17
18 # 5. 使用 K-Means 计算 OpenCV 图像的主导颜色:
19     dominant_color = get_dominant_color(image_cv)
20
21 # 6. 计算颜色偏向度:
22     color_bias = get_color_bias(dominant_color)
23
24 # 7. 加载预训练模型并进行图像分类:
25     model = load_pretrained_model(model_name)
26     predicted_label = classify_image(model, image_path)
27
28 # 输出: 平均亮度、颜色分类、主导颜色、颜色偏向、物品分类结果
```

2.2.1 计算图像的平均亮度

1. 实验原理

图像的平均亮度是衡量图像整体明暗程度的重要指标。通过计算图像每个颜色通道（R、G、B）的像素值的平均值，可以得到图像在各个通道上的平均亮度。归一化处理可以将亮度值映射到[0, 1]区间，便于不同图像之间的比较。

2. 代码展示

▼ 代码块

```
1 # 计算平均亮度的函数 (使用OpenCV和PIL两种方式)
2 def calculate_brightness(image_tensor, normalize=False):
3     # 获取图像张量的通道数, 通常彩色图像有3个通道 (R, G, B)
4     channels = image_tensor.shape[2]
5     # 初始化一个空列表, 用于存储每个通道的平均亮度
```

```

6     avg_brightness = []
7     # 遍历每个颜色通道
8     for i in range(channels):
9         # 计算当前通道的平均亮度：当i为0、1、2时，分别对应R、G、B三个通道
10        channel_mean = np.mean(image_tensor[:, :, i])
11        # 如果有需求，可以对亮度进行归一化处理
12        if normalize:
13            channel_mean = channel_mean / 255.0
14        # 将当前通道的平均亮度添加到列表中
15        avg_brightness.append(channel_mean)
16    return avg_brightness

```

- 亮度 (Brightness) 可以通过 RGB 三通道的加权平均计算。
- 彩色图像的平均亮度可以通过分别计算每个颜色通道（红色、绿色、蓝色）的平均值来得到。假设图像的大小为 $M \times N$ ，则彩色图像的平均亮度 (LR, LG, LB) 可以用以下公式表示：

$$\begin{aligned}
 \circ \quad L_R &= \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} R(i, j) \\
 \circ \quad L_G &= \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} G(i, j) \\
 \circ \quad L_B &= \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} B(i, j)
 \end{aligned}$$

3. 实验结果与分析：

以后续的实验结果分析的动物图像 ([animal3.jpg](#)——猫cat) 为例，计算得到其RGB三个通道的平均亮度分别为114.18、101.11和85.41。归一化后，亮度值分别为0.447、0.396和0.334。

这表明图像在红色通道相对较亮，蓝色通道相对较暗，整体呈现出温暖的色调。

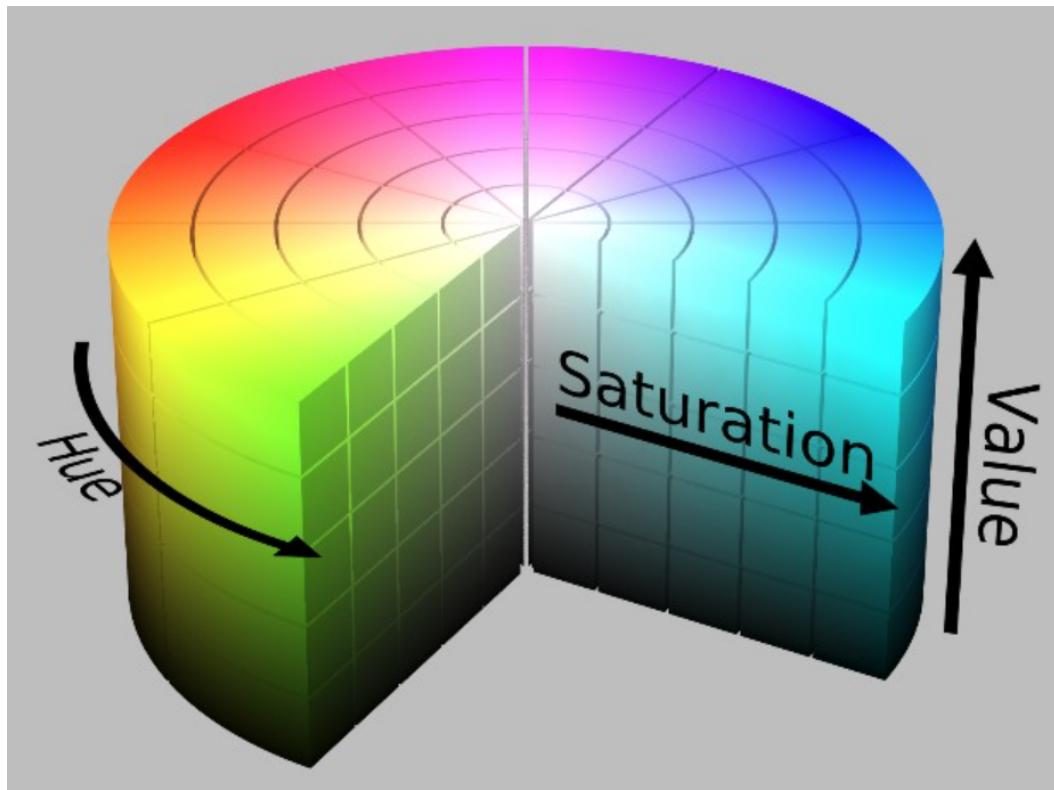
2.2.2 颜色分类 (RGB & HSV)

- RGB颜色模型通过红、绿、蓝三个通道的组合来表示颜色；
- 而HSV颜色模型则将颜色分为色调 (Hue) 、饱和度 (Saturation) 和亮度 (Value) 三个维度；
- 基于RGB的分类通过比较各通道的亮度值来判断主导颜色；
- 基于HSV的分类则利用色调的范围来确定颜色类别，同时考虑饱和度和亮度的影响。

1. 实验原理

- RGB 颜色空间：通过比较 R、G、B 三通道的亮度，确定主要颜色。
- HSV 颜色空间：**HSV对用户来说是一种直观的颜色模型。**

- **H (Hue, 色相)** : 决定颜色种类, 其中OpenCV中的色调取值范围 $0^{\circ} \sim 180^{\circ}$ 与标准HSV模型中的 $0^{\circ} \sim 360^{\circ}$ 不同, 这是OpenCV为了内部处理的高效性而进行的归一化处理。;
- **S (Saturation, 饱和度)** : 决定颜色的纯度;
- **V (Value, 亮度)** : 决定颜色的明暗程度。



2. 代码展示

▼ 代码块

```

1   # 判断图像的颜色 (基于平均亮度)
2   def classify_image_color(brightness, normalized=False):
3       """ 根据 RGB 亮度值判断主导颜色 """
4       # RGB亮度阈值设置
5       threshold = 200 if not normalized else 0.78 # 用于判断白色
6       black_threshold = threshold / 5 # 用于判断黑色
7
8       # 通道间亮度差异的阈值
9       color_diff_threshold = threshold / 4 # 当两个通道的亮度差异小于该阈值时, 认为
10      是颜色接近
11
12      r, g, b = brightness # 拆分 R, G, B 三个通道
13
14      # 找到最亮的通道
15      max_channel = max(brightness)
16      min_channel = min(brightness)
17
18      # 如果三通道亮度都高, 判断是否接近白色
19      if min_channel > threshold:

```

```

19         return "白色"
20     # 如果三通道亮度都低，判断是否接近黑色
21     elif max_channel < black_threshold:
22         return "黑色"
23
24     # 如果最大通道亮度差异小于阈值，判断为多色图像
25     if abs(r - g) < color_diff_threshold and abs(r - b) <
color_diff_threshold and abs(g - b) < color_diff_threshold:
26         return "多色图像"
27
28     # 具体颜色分类逻辑
29     elif r >= g and r >= b:    # 红色占主导
30         return "红色" if g < threshold and b < threshold else "橙色/黄色"
31     elif g >= r and g >= b:    # 绿色占主导
32         return "绿色"
33     elif b >= r and b >= g:    # 蓝色占主导
34         return "蓝色"
35     elif r >= b and g >= b:    # 黄绿色
36         return "黄色/绿色"
37     elif g >= r and b >= r:    # 青色
38         return "青色"
39     elif r >= g and b >= g:    # 品红
40         return "品红"
41     else:
42         return "多色图像"
43

```

▼ 代码块

```

1  # 使用 HSV 颜色空间分类颜色
2  def classify_color_hsv(image):
3      """ 使用 HSV 颜色空间分类颜色 """
4      image_hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)    # 转换为 HSV 颜色空间
5      h, s, v = cv2.split(image_hsv) # 拆分 H, S, V 三个通道
6
7      mean_hue = np.mean(h)    # 色调
8      mean_saturation = np.mean(s)    # 饱和度
9      mean_value = np.mean(v)    # 亮度
10
11     # 低饱和度颜色（黑、白、灰）
12     if mean_saturation < 40:
13         if mean_value < 50:
14             return "黑色"
15         elif mean_value > 200:
16             return "白色"
17     else:

```

```
18         return "灰色"
19
20     # 彩色部分的分类
21     if 0 <= mean_hue < 15 or 165 <= mean_hue <= 180:
22         return "红色"
23     elif 15 <= mean_hue < 30:
24         return "橙色"
25     elif 30 <= mean_hue < 45:
26         return "黄色"
27     elif 45 <= mean_hue < 90:
28         return "绿色"
29     elif 90 <= mean_hue < 150:
30         return "蓝色"
31     elif 150 <= mean_hue < 165:
32         return "紫色"
33     else:
34         return "未知颜色"
35
```

3. 代码解释分析：

a. `classify_image_color` 函数的内部判断逻辑是基于 RGB 亮度值对图像进行颜色分类：

i. 函数接受两个参数

- brightness: 一个包含三个元素的列表或元组，分别表示图像在 R、G、B 三个通道的平均亮度值。
- normalized: 一个布尔值，表示亮度值是否已经归一化（默认为 False）。

ii. 阈值设定

根据是否归一化，设定不同的阈值：

- threshold: 用于判断白色，非归一化时为 200，归一化时为 0.78。
- black_threshold: 用于判断黑色，为 threshold / 5。
- color_diff_threshold: 用于判断多色图像，为 threshold / 4。

iii. 黑白判断

- 如果所有通道的最小亮度值大于 threshold，判断为 **白色**。
- 如果所有通道的最大亮度值小于 black_threshold，判断为 **黑色**。

iv. 多色判断

- 如果 R、G、B 三个通道之间的亮度差异都小于 color_diff_threshold，判断为 **多色图像**。

v. 具体颜色分类

- **红色**: 如果 R 通道亮度最高, 且 G 和 B 通道亮度较低。
- **橙色/黄色**: 如果 R 通道亮度最高, 但 G 或 B 通道亮度不低。
- **绿色**: 如果 G 通道亮度最高。
- **蓝色**: 如果 B 通道亮度最高。
- **黄色/绿色**: 如果 R 和 G 通道亮度较高, B 通道亮度较低。
- **青色**: 如果 G 和 B 通道亮度较高, R 通道亮度较低。
- **品红**: 如果 R 和 B 通道亮度较高, G 通道亮度较低。

b. `classify_color_hsv` 函数的内部判断逻辑是基于 HSV 颜色空间对图像进行颜色分类:

i. 颜色空间转换

首先, 将输入图像从 RGB 颜色空间转换为 HSV 颜色空间。这是因为 HSV 颜色空间更符合人类对颜色的直观感受, 其中:

- **H (Hue)** : 色调, 表示颜色的类型 (如红色、绿色、蓝色)。
- **S (Saturation)** : 饱和度, 表示颜色的纯度。
- **V (Value)** : 亮度, 表示颜色的明暗程度。

ii. 计算均值

对转换后的 HSV 图像的每个通道 (H、S、V) 计算均值:

- mean_hue: 色调均值。
- mean_saturation: 饱和度均值。
- mean_value: 亮度均值。

iii. 低饱和度判断

如果饱和度均值 (mean_saturation) 较低 (小于 40), 说明图像颜色较为灰暗, 此时主要根据亮度均值 (mean_value) 进行分类:

- 如果亮度均值小于 50, 判断为 **黑色**。
- 如果亮度均值大于 200, 判断为 **白色**。
- 否则, 判断为 **灰色**。

iv. 高饱和度判断

如果饱和度均值较高 (大于等于 40), 说明图像颜色较为鲜艳, 此时根据色调均值 (mean_hue) 的范围进行分类:

色调值（度）	颜色
0	红色
30	黄色
60	绿色
90	青色
120	蓝色
150	品红色

CSDN @hai41

- **红色**: 色调在 0 到 15 之间，或者在 165 到 180 之间。
- **橙色**: 色调在 15 到 30 之间。
- **黄色**: 色调在 30 到 45 之间。
- **绿色**: 色调在 45 到 90 之间。
- **蓝色**: 色调在 90 到 150 之间。
- **紫色**: 色调在 150 到 165 之间。
- **未知颜色**: 如果色调不在上述范围内，判断为未知颜色。

4. 实验结果与分析

对于一张以蓝天白云为主的图像 ([image2.jpg](#))，基于RGB的分类结果为“蓝色”，基于HSV的分类结果也为“蓝色”。这表明两种方法在处理此类图像时结果一致。但对于一张含有多种颜色混合的图像，如向日葵 ([plant2.jpg](#))，RGB方法可能判断为红色，而HSV方法可能会根据主要色调给出更具体的颜色类别为橙色。

2.2.3 计算主导颜色 (K-Means 聚类)

1. 实验原理

- K-means聚类算法是一种无监督学习算法，通过将数据点划分为K个簇，使得簇内数据点的相似性最大，而簇间数据点的相似性最小。
- 在图像处理中，可以将每个像素的RGB值视为数据点，通过聚类找到图像中主要的颜色类别。选择出现次数最多的簇的中心颜色作为图像的主导颜色。
- K-Means 试图将 pixels 数据点分成 k 组（默认 k=3），每个簇（cluster）对应一个聚类中心（代表一种颜色）
- 停止条件：
 - 迭代次数达到 10 次 (MAX_ITER=10)。
 - 质心变化小于 1.0 (EPS=1.0)。
 - 只要满足其中之一，算法就会终止。

2. 代码展示

```

1 代码块
2 def get_dominant_color(image, k=3):
3     """
4     使用 K-Means 计算主导颜色:
5     输入值:
6         - pixels: 输入的像素点数据。
7         - k: 指定聚类数 (颜色数)。
8         - None: 忽略预定义的标签。
9         - criteria: 终止条件。
10        - 10: 表示执行 10 次 K-Means 以求得最佳分类 (避免局部最优)。
11        - cv2.KMEANS_RANDOM_CENTERS: 初始质心随机选择。
12     返回值:
13         - _: K-Means 计算的压缩误差SSE。
14         - labels: 每个像素点所属的聚类索引0~k-1。
15         - centers: k 个聚类中心, 即代表 k 种颜色。
16     """
17     # 将图像转换为二维数组, 每行是一个像素的 RGB 值
18     pixels = image.reshape(-1, 3).astype(np.float32)
19
20     # 终止条件:
21     #   1. 迭代次数达到 10 次 (MAX_ITER=10)
22     #   2. 质心变化小于 1.0 (EPS=1.0)
23     criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
24     _, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
25                                     cv2.KMEANS_RANDOM_CENTERS)
26     # 统计最常见的颜色
27     counts = np.bincount(labels.flatten())
28     dominant_color = centers[np.argmax(counts)]
29
30     return dominant_color.astype(int) # 返回 RGB 格式

```

3. 代码解释分析

- 图像像素展平:** 将图像从三维数组 (高度 x 宽度 x 通道数) 展平为二维数组, 其中每一行代表一个像素的RGB值。
- K-means聚类:** 使用OpenCV的kmeans函数进行聚类, 将像素分为k个簇。函数返回每个像素的簇标签和每个簇的中心点 (即颜色)。

i. 输入值:

- pixels: 输入的像素点数据。
- k: 指定聚类数 (颜色数)。
- None: 忽略预定义的标签。
- criteria: 终止条件。

- 10: 表示执行 10 次 K-Means 以求得最佳分类（避免局部最优）。
- cv2.KMEANS_RANDOM_CENTERS: 初始质心随机选择。

ii. 返回值:

- _: K-Means 计算的压缩误差 (SSE)。
- labels: 每个像素点所属的聚类索引 (0~k-1)。
- centers: k 个聚类中心，即代表 k 种颜色。

c. **统计像素数量**: 统计每个簇的像素数量，找到出现次数最多的簇。

d. **提取主导颜色**: 返回出现次数最多的簇的中心颜色作为主导颜色。

4. 实验结果与分析

以一张以绿色为主的森林图像为例，K-means聚类 (k=3) 得到的主导颜色为：红色: 113，绿色: 175，蓝色: 33。这与直观感受一致。

2.2.4 计算颜色偏向

1. 实验原理

- 颜色偏向反映了图像中某一颜色通道相对于其他通道的主导程度。通过计算每个通道的亮度与另外两通道亮度平均值的差值，可以判断图像在红、绿、蓝三个方向上的偏向情况。正的差值表示该通道亮度高于另外两个通道的平均值，负值则相反。

2. 代码展示

▼ 代码块

```

1  # 计算颜色偏向
2  def get_color_bias(dominant_color):
3      """ 根据 RGB 值计算颜色偏向 """
4      r, g, b = dominant_color
5      # 计算各通道的偏向值
6      biases = {
7          "红色偏向": r - (g + b) / 2,
8          "绿色偏向": g - (r + b) / 2,
9          "蓝色偏向": b - (r + g) / 2
10     }
11
12     # 判断是否为灰色 (RGB 三个通道的最大最小差值小于 threshold)
13     threshold = 5
14     max_rgb, min_rgb = max(r, g, b), min(r, g, b)
15     if max_rgb - min_rgb < threshold:
16         return "无明显偏向 (可能是灰色)"
17
18     # 选择偏向最大的颜色

```

```
19     max_bias_color, max_bias = max(biases.items(), key=lambda x: x[1])
20
21     return max_bias_color if max_bias > 0 else "无明显偏向"
```

3. 代码解释分析

- a. **计算偏向值**: 对于每个通道，计算其亮度与另外两个通道亮度平均值的差值。
- b. **判断偏向**: 比较三个通道的偏向值，返回偏向值最大的通道对应的颜色偏向。
- c. **阈值判断**: 在比较通道偏向时，通过阈值threshold=5，来加强判断颜色的“无偏向”性。

4. 实验结果与分析

对于一张以红色为主的图像，计算得到红色偏向明显。而对于一张灰度图像，三个通道的亮度差异较小，判断为“无明显偏向”。

2.2.5 物品分类 (AlexNet & VGG16)

1. 实验原理

深度学习中的卷积神经网络 (CNN) 在图像分类任务中表现出色。AlexNet和VGG16是两种经典的CNN模型，预训练在大规模的ImageNet数据集上，能够提取图像的高级特征并进行分类。通过加载预训练模型，可以利用其强大的特征提取能力，对输入图像进行快速准确的分类。

- 卷积神经网络 (CNN) 通过学习层级特征，进行物品分类。
- AlexNet: 经典 CNN 模型，适用于常见物品分类。
- VGG16: 更深的 CNN，能识别更多细节。

2. 代码展示

▼ 代码块

```
1  # 加载模型
2  def load_models():
3      # 加载预训练的AlexNet和VGG16模型
4      alexnet = models.alexnet(weights=models.AlexNet_Weights.DEFAULT)
5      vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
6
7      # 设置为评估模式
8      alexnet.eval()
9      vgg16.eval()
10
11     return alexnet, vgg16
12
13     # 图像分类函数
14     def classify_image(image_path, models, class_idx):
```

```
15     transform = transforms.Compose([
16         transforms.Resize(256), # 调整图像大小
17         transforms.CenterCrop(224), # 中心裁剪
18         transforms.ToTensor(), # 转换为张量
19         transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
20             0.225]), # 归一化
21     ])
22
23     image = Image.open(image_path)
24     image = transform(image).unsqueeze(0)
25
26     results = {}
27     for model_name, model in models.items():
28         with torch.no_grad(): # 禁用梯度计算
29             output = model(image)
30             _, predicted_idx = torch.max(output, 1) # 获取预测的类别索引
31             predicted_idx = predicted_idx.item()
32             # 获取类别名称
33             class_name = class_idx[str(predicted_idx)][1]
34             results[model_name] = class_name
35
36     return results
37
38 # 加载模型
39 alexnet, vgg16 = load_models()
40 models = {"alexnet": alexnet, "vgg16": vgg16}
```

3. 代码解释分析

- 模型加载：**加载预训练的AlexNet和VGG16模型，并设置为评估模式。
- 图像预处理：**对输入图像进行标准化预处理，包括调整大小、裁剪、转换为张量和归一化。
- 分类推理：**使用每个模型对预处理后的图像进行分类，获取预测的类别索引，并根据类别索引获取类别名称。

4. 实验结果与分析

以一张包含小熊猫的图像 ([animal1.jpg](#)) 为例，AlexNet分类结果为“lesser_panda”，VGG16分类结果也为“lesser_panda”，两种模型结果一致。但对于一些细节复杂或类别相似度高的图像，不同模型可能会给出不同的分类结果，这反映了不同模型架构和参数对分类性能的影响。

2.3 系统实现

本系统采用前后端分离架构，前端使用 React + JavaScript，后端使用 Flask + Python，结合 OpenCV、PIL、K-Means 及 PyTorch 深度学习模型，实现自动颜色分析与物品识别。以下是系统的

关键实现。

2.3.1 前端实现 (app.js)

前端负责图片上传、与后端交互、结果展示，主要流程如下：

1. 选择或拖拽图片上传；
2. 将图片数据发送至后端（Flask API）；
3. 接收后端返回的颜色分类、主导颜色、颜色偏向、物品分类等信息；
4. 将分析结果动态显示在网页上。

前端主要代码为：app.js

- 使用 HTML + JavaScript（React）构建。
- 负责图片上传，并发送请求到后端。
- 显示分析结果，包括平均亮度、颜色分类、主导颜色、颜色偏向、物品分类等。

▼ 代码块

```
1      // 使用 useState 管理图片、预览、分析结果和图表数据的状态
2      const [image, setImage] = useState(null); // 图片
3      const [preview, setPreview] = useState(null); // 预览
4      const [result, setResult] = useState(null); // 分析结果
5      const [multipleColors, setMultipleColors] = useState(null); // 多色分析结果
6      const [meanBrightnessPIL, setMeanBrightnessPIL] = useState(null); // PIL的
平均亮度
7      const [meanBrightnessOpenCV, setMeanBrightnessOpenCV] = useState(null);
//OpenCV的平均亮度
8      const [chartData, setChartData] = useState(null); // 图表数据
9      const [brightnessData, setBrightnessData] = useState(null); // 亮度数据
10     const [normalize, setNormalize] = useState(false); // 默认不标准化
11     const [dominantColor, setDominantColor] = useState(null); // 主导颜色
12     const [hsvColor, setHsvColor] = useState(null); // HSV分类
13     const [colorBias, setColorBias] = useState(null); // 颜色偏向度
14     const [showDominantColor, setShowDominantColor] = useState(false); // 控制
显示主导颜色
15     const [showHsvColor, setShowHsvColor] = useState(false); // 控制显示HSV分类
16     const [showColorBias, setShowColorBias] = useState(false); // 控制显示颜色偏
向度
```

2.3.2 后端实现 (app.py)

后端使用 Flask 处理图片分析，包括：

- 接收前端上传的图片，转换格式；
- 计算平均亮度、颜色分类、主导颜色、颜色偏向；

- 调用 AlexNet / VGG16 进行物品分类；
- 返回 JSON 结果给前端。

后端主要代码为：app.py

- 使用 Flask 构建。
- 处理前端上传的图像。
- 调用基础架构中的功能模块，计算平均亮度，并进行颜色判别。
- 通过 K-Means、HSV 颜色空间、RGB 计算等方法分析颜色信息。
- 可以选择使用 AlexNet 和 VGG16 进行物品分类。
- 返回分析结果给前端。

▼ 代码块

```

1      # 允许前端返回的结果
2      return jsonify({
3          "filename": file.filename, # 文件名
4          "normalize": normalize, # 是否归一化
5          "mean_brightness_opencv": brightness_cv2, # 平均亮度: OpenCV
6          "mean_brightness_pil": brightness_pil, # 平均亮度: PIL
7          "color_classification_opencv": color_cv2, # 颜色分类: OpenCV
8          "color_classification_pil": color_pil, # 颜色分类: PIL
9          "multiple_colors": multiple_colors, # 是否多色图像
10         "dominant_color_rgb": dominant_color.tolist(), # 主要颜色 (RGB)
11         "dominant_color_hsv": hsv_color, # 主要颜色分类 (HSV)
12         "color_bias": color_bias, # 颜色偏向
13         "classification_results": {
14             "alexnet": classification_results["alexnet"], # 分类结果: AlexNet
15             "vgg16": classification_results["vgg16"] # 分类结果: VGG16
16         }
17     }), 200
18

```

2.3.3 颜色分析实现

本系统采用多种颜色分类方法，以提高颜色识别的准确性。

1. RGB 颜色分类

- **实现原理：**通过比较图像在红、绿、蓝三个通道的平均亮度值，判断图像的主导颜色。计算每个通道的平均亮度，然后根据亮度值的大小关系进行分类。
- **分类过程：**
 - 计算三个通道的平均亮度值。

- 根据亮度值判断是否为黑白图像。
- 若不是黑白图像，比较三个通道的亮度值，确定主导颜色。

2. HSV 颜色分类

- **实现原理：**将图像从 RGB 颜色空间转换为 HSV 颜色空间，利用H、S、V三种属性的范围来判断颜色类别。
- **分类过程：**
 - 将图像转换为 HSV 颜色空间。
 - 计算色调、饱和度和亮度的均值。
 - 根据色调均值的范围判断颜色类别。

3. 颜色偏向分析

- **实现原理：**通过计算每个通道的亮度与另外两个通道亮度平均值的差值，判断图像在红、绿、蓝三个方向上的偏向情况。
- **分析过程：**
 - 计算每个通道的亮度偏差。
 - 比较偏差值，确定最大偏差的方向。
 - 根据最大偏差值判断颜色偏向。

2.3.4 物品分类实现

- **采用 AlexNet / VGG16 进行分类：**
 - **实现原理：**利用深度学习中的卷积神经网络（CNN）进行图像分类。AlexNet 和 VGG16 是两种经典的 CNN 模型，预训练在大规模的 ImageNet 数据集上，能够提取图像的高级特征并进行分类。
 - **分类过程：**
 - 加载预训练的 AlexNet 和 VGG16 模型。
 - 对输入图像进行标准化预处理。
 - 使用每个模型对预处理后的图像进行分类，获取预测的类别索引，并根据类别索引获取类别名称。

三、实验流程

3.1 实验环境

本实验在高性能计算环境下进行，包含硬件、软件及数据集三个方面：

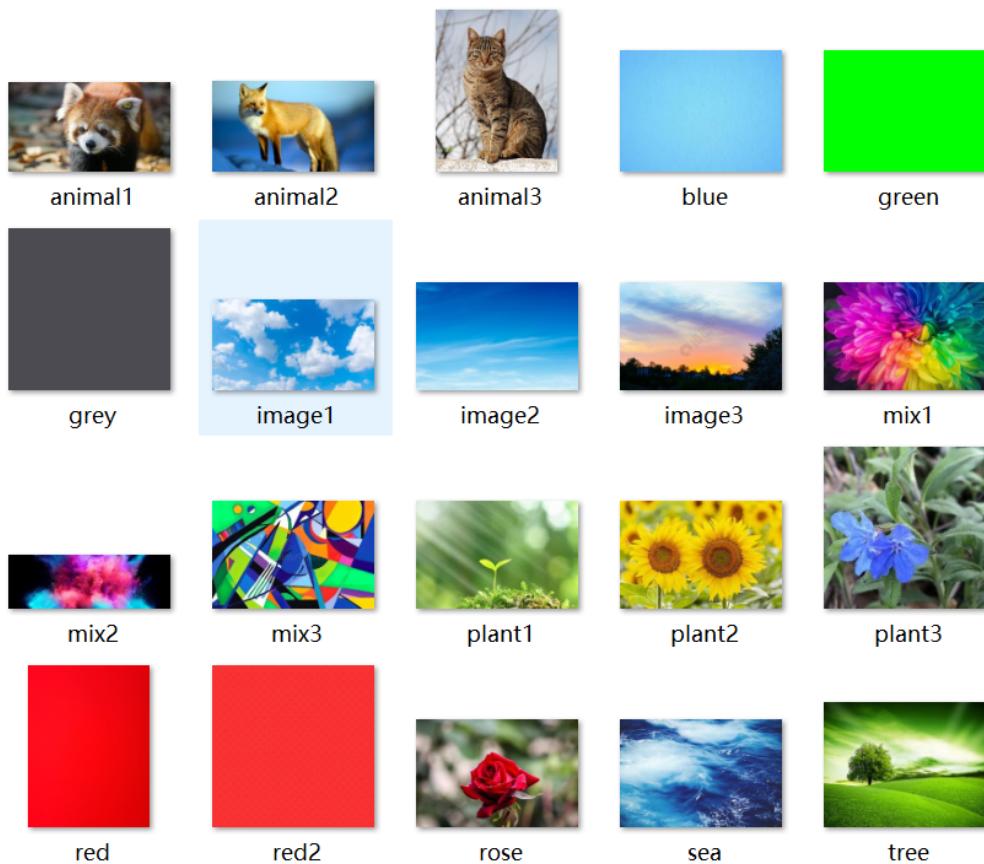
(1) 硬件环境

- 处理器 (CPU): AMD Ryzen 7
- 显卡 (GPU): NVIDIA GeForce RTX 3060

(2) 软件环境

- 操作系统: Windows 11
- 编程语言: Python 3.11.4
- 主要依赖库:
 - PyTorch 2.0.1 (深度学习模型)
 - Torchvision 0.15.2 (模型预训练及数据处理)
 - OpenCV 4.9.0 (图像处理及 HSV 转换)
 - NumPy 1.24.3 (数学计算及 K-Means 聚类)
 - Flask 2.2.2 (后端 API 开发)

(3) 实验数据集



本实验采用**20张真实图片**，来源包括：

- 手机拍摄：在不同光照条件下拍摄的物品图像，如自然光、室内灯光和阴影环境下的照片，以测试模型在各种光照条件下的表现。
- 网络下载：从网络获取的不同材质、颜色及背景复杂度的图片，确保数据集的多样性和代表性。

数据集特点：

- **丰富的颜色种类**：包含红色（如图片red、red2）、蓝色（如blue）、绿色（如green）等纯色物品，用于测试颜色分类模型对基本颜色的区分能力。
 - **多样的光照条件**：涵盖自然光、室内灯光、阴影等多种光照环境下的图像（如animal1、animal2、animal3），以评估模型在不同光照条件下的适应性和稳定性。
 - **复杂的纹理和形状**：包含不同纹理和形状的图像，如纯色（如grey）、渐变色（如image1、image2、image3）、复杂图案（如mix1、mix2、mix3）以及自然景物（如plant1、plant2、plant3、rose、sea、tree），用于验证分类方法对复杂图像的鲁棒性和准确性。
 - **多类别图像**：涵盖动物、植物、自然风景等多个类别的图像，如动物图像（animal1、animal2、animal3）、植物图像（plant1、plant2、plant3）、自然风景图像（sea、tree）以及花卉图像（rose），用于全面评估模型的分类性能。
-

3.2 实验流程

本实验分为 **图像处理**、**颜色分析**、**物品分类**、**前端展示** 四个核心阶段，如下。

同时给出对应的图像展示具体的实验结果：

3.2.1 基础文件操作（main.py）

1. 图像处理

- 图像读取与格式统一：利用 OpenCV 和 PIL 两种库读取图像，确保图像数据的准确获取。
 - OpenCV 的 imread 函数读取图像后，通过 cvtColor 方法将 BGR 格式转换为 RGB 格式；
 - PIL 的 Image.open 方法直接读取图像并以 RGB 格式存储。
- 归一化处理：对图像数据进行归一化（Normalization）处理，将像素值从 [0, 255] 映射到 [0, 1] 区间，以满足深度学习模型的输入要求，提高模型的训练效率和稳定性。
- 颜色空间转换：将图像从 RGB 颜色空间转换到 HSV 颜色空间，便于后续基于色调（Hue）的颜色分析。

2. 颜色分析

- RGB 平均亮度计算：分别计算图像在红、绿、蓝三个通道的平均亮度值，通过比较这些值来初步判断图像的亮度分布情况。
- RGB 颜色分类：根据 RGB 亮度值的大小关系，使用预定义的规则对图像进行颜色分类，判断其主导颜色是否为红色、绿色、蓝色等。
- HSV 颜色分类：在 HSV 颜色空间中，根据色调（Hue）、饱和度（Saturation）和亮度（Value）的范围对图像进行颜色分类，进一步提高颜色分类的准确性和鲁棒性。

- K-Means 聚类提取主导颜色：使用 K-Means 聚类算法（K=3）对图像像素进行聚类，找出数量最多的类别对应的中心颜色作为图像的主导颜色。
- 颜色偏向度计算：计算每个颜色通道的亮度与另外两个通道亮度平均值的差值，判断图像在红、绿、蓝三个方向上的偏向程度。

3. 物品分类

- 预训练模型加载：加载 PyTorch 中预训练的 AlexNet 和 VGG16 模型，并设置为评估模式。
- 图像预处理：对输入图像进行标准化预处理，包括调整大小、裁剪、转换为张量和归一化。
- 分类推理：使用预训练模型对预处理后的图像进行分类，获取预测的类别索引，并根据 ImageNet 标签返回具体的分类结果。

```
请输入图像文件路径（多个路径用逗号分隔）: images/animal1.jpg
是否归一化亮度？(yes/no): yes
请选择模型（alexnet/vgg16）: alexnet

===== 计算结果 =====
图像文件: images/animal1.jpg
是否归一化: 是
OpenCV 计算的平均亮度: [0.447776731901263, 0.3965382423967119, 0.3349538487475238]
PIL 计算的平均亮度: [0.44776911417381215, 0.3965461098857185, 0.3349557219591921]
主导颜色 (OpenCV): 黑色
主导颜色 (PIL): 黑色
是否多色图像: 否
HSV 颜色分类: 黄色
主导颜色 (RGB): [121 109 93]
颜色偏向度: 红色偏向
识别的物体分类: lesser panda
```

使用归一化的main.py文件结果

```
请输入图像文件路径（多个路径用逗号分隔）: images/animal1.jpg
是否归一化亮度？(yes/no): no
请选择模型（alexnet/vgg16）: alexnet

===== 计算结果 =====
图像文件: images/animal1.jpg
是否归一化: 否
OpenCV 计算的平均亮度: [114.18306663482207, 101.11725181116154, 85.41323143061858]
PIL 计算的平均亮度: [114.1811241143221, 101.11925802085821, 85.41370909959399]
主导颜色 (OpenCV): 多色图像
主导颜色 (PIL): 多色图像
是否多色图像: 是
HSV 颜色分类: 黄色
主导颜色 (RGB): [121 109 93]
颜色偏向度: 红色偏向
识别的物体分类: lesser panda
```

不使用归一化的main.py文件结果

```

转换后的图像张量 (OpenCV):
tensor([[[0.2078, 0.2039, 0.1961, ..., 0.1412, 0.1216, 0.1216],
       [0.2078, 0.2039, 0.1961, ..., 0.1373, 0.1216, 0.1216],
       [0.2039, 0.2000, 0.1961, ..., 0.1333, 0.1176, 0.1176],
       ...,
       [0.3686, 0.3608, 0.3529, ..., 0.7843, 0.7804, 0.7804],
       [0.3961, 0.3882, 0.3725, ..., 0.7843, 0.7725, 0.7725],
       [0.4157, 0.3765, 0.3529, ..., 0.7882, 0.7922, 0.7843]],

      [[0.1647, 0.1608, 0.1529, ..., 0.1255, 0.1176, 0.1176],
       [0.1647, 0.1608, 0.1529, ..., 0.1216, 0.1176, 0.1176],
       [0.1608, 0.1569, 0.1529, ..., 0.1176, 0.1137, 0.1137],
       ...,
       [0.4275, 0.4196, 0.4118, ..., 0.6353, 0.6314, 0.6314],
       [0.4588, 0.4510, 0.4353, ..., 0.6353, 0.6235, 0.6235],
       [0.4902, 0.4510, 0.4275, ..., 0.6392, 0.6392, 0.6353]],

      [[0.1569, 0.1529, 0.1451, ..., 0.1216, 0.1098, 0.1098],
       [0.1569, 0.1529, 0.1451, ..., 0.1176, 0.1098, 0.1098],
       [0.1529, 0.1490, 0.1451, ..., 0.1137, 0.1059, 0.1059],
       ...,
       [0.4000, 0.3922, 0.3843, ..., 0.4431, 0.4392, 0.4392],
       [0.4471, 0.4392, 0.4235, ..., 0.4510, 0.4471, 0.4392],
       [0.4824, 0.4431, 0.4196, ..., 0.4627, 0.4706, 0.4588]]])

```

OpenCV展示图像张量结果

```

转换后的图像张量 (PIL):
tensor([[[0.2078, 0.2039, 0.1961, ..., 0.1412, 0.1216, 0.1216],
       [0.2078, 0.2039, 0.1961, ..., 0.1373, 0.1216, 0.1216],
       [0.2039, 0.2000, 0.1961, ..., 0.1333, 0.1176, 0.1176],
       ...,
       [0.3686, 0.3608, 0.3529, ..., 0.7843, 0.7804, 0.7804],
       [0.3961, 0.3882, 0.3725, ..., 0.7843, 0.7725, 0.7725],
       [0.4157, 0.3765, 0.3529, ..., 0.7882, 0.7922, 0.7843]],

      [[0.1647, 0.1608, 0.1529, ..., 0.1255, 0.1176, 0.1176],
       [0.1647, 0.1608, 0.1529, ..., 0.1216, 0.1176, 0.1176],
       [0.1608, 0.1569, 0.1529, ..., 0.1176, 0.1137, 0.1137],
       ...,
       [0.4275, 0.4196, 0.4118, ..., 0.6353, 0.6314, 0.6314],
       [0.4588, 0.4510, 0.4353, ..., 0.6353, 0.6235, 0.6235],
       [0.4902, 0.4510, 0.4275, ..., 0.6392, 0.6431, 0.6353]],

      [[0.1569, 0.1529, 0.1451, ..., 0.1216, 0.1098, 0.1098],
       [0.1569, 0.1529, 0.1451, ..., 0.1176, 0.1098, 0.1098],
       [0.1529, 0.1490, 0.1451, ..., 0.1137, 0.1059, 0.1059],
       ...,
       [0.4078, 0.4000, 0.3843, ..., 0.4431, 0.4392, 0.4392],
       [0.4471, 0.4392, 0.4235, ..., 0.4588, 0.4471, 0.4471],
       [0.4824, 0.4431, 0.4196, ..., 0.4627, 0.4667, 0.4588]]])

```

PIL展示图像张量结果

3.2.2 前后端创新操作 (app.py和app.js)

4. 前端展示

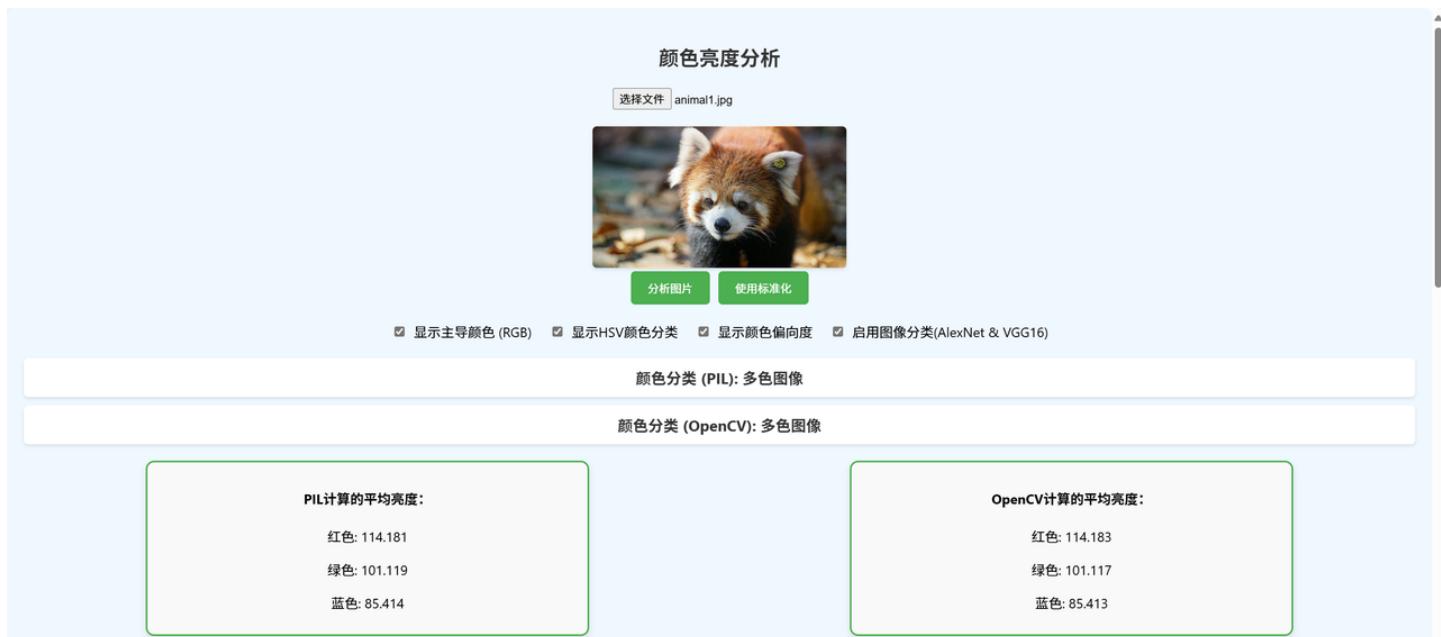
- React 前端实现：

- 结果展示：通过 React 构建用户界面，展示图像及颜色分析结果，包括：
 - 平均亮度 (OpenCV、PIL) ；
 - 颜色分类 (RGB、HSV) ；
 - 主导颜色 (K-Means 聚类) ；

- 颜色偏向（R/G/B 偏向度）；
- 物品分类（AlexNet 和 VGG16 结果）。
- 交互式体验：实现交互式展示，用户上传新图片后，前端发送请求到后端，后端处理并返回分析结果，前端实时更新展示内容。
- Flask 后端实现：
 - API 开发：使用 Flask 开发后端 API，处理前端用户上传的图像，调用基础文件操作中的功能模块进行图像处理、颜色分析和物品分类。
 - 结果返回：将分析结果以 JSON 格式返回给前端，确保数据传输的高效性和兼容性，实时更新分析结果。

下面给出前端的显示结果（以animal1.jpg为例），包括：

- 1是OpenCV和PIL的平均亮度计算和机器的颜色分类结果，同时可以调节是否显示2的画面；
- 2是基于K聚类的主导颜色、HSV颜色分类、颜色偏向和图像分类结果这4个结果的显示；
- 3是1中平均亮度的可视化折线图，体现两种计算方法的对比；
- 4是1中平均亮度的可视化柱状图，体现两种计算方法的对比。



前端显示结果1

主导颜色 (RGB)

■ 红色: 121 绿色: 109 蓝色: 93

HSV 颜色分类

黄色

颜色偏向度

红色偏向

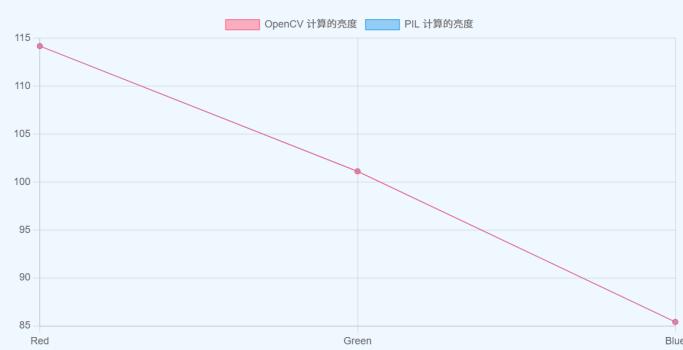
图像分类结果:

AlexNet 分类结果: lesser_panda

VGG16 分类结果: lesser_panda

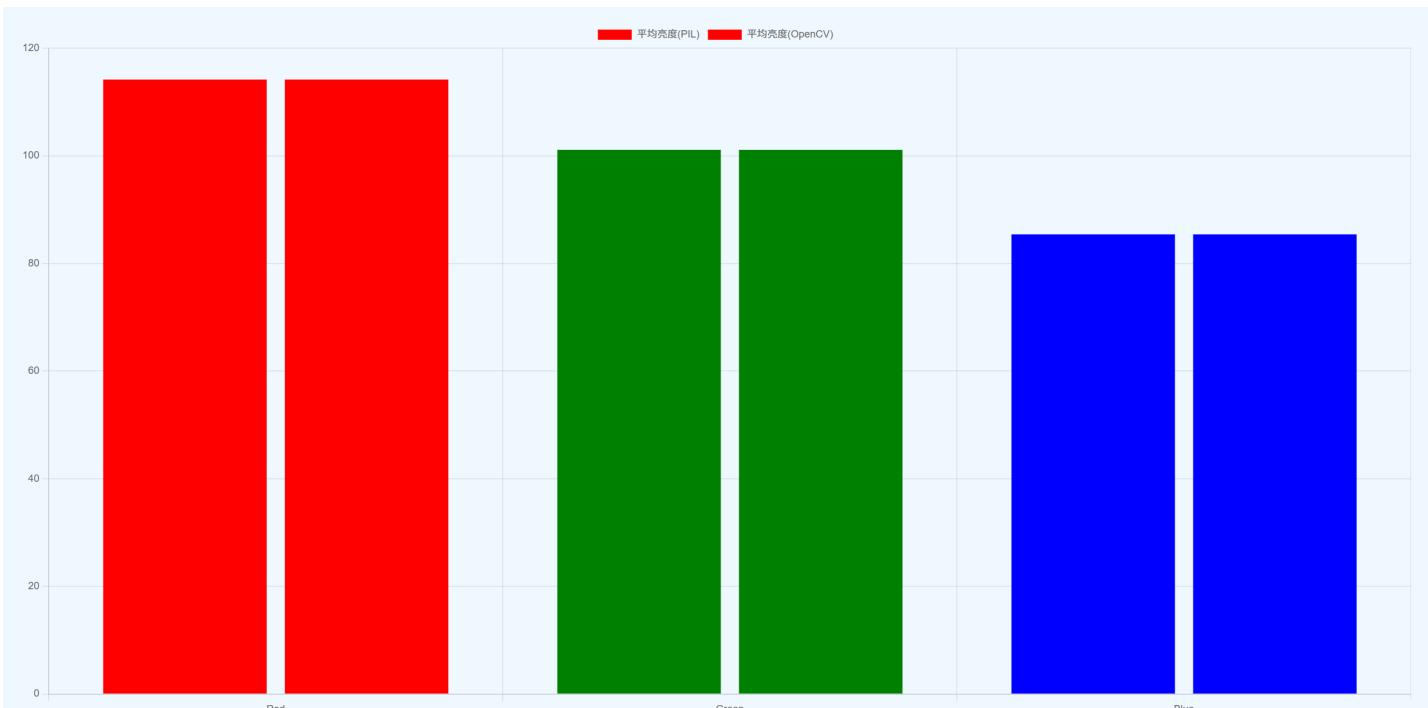
前端显示结果2

亮度图



前端显示结果3

平均亮度(PIL) 平均亮度(OpenCV)



前端显示结果4

3.3 实验结果分析

本实验的结果分为 颜色分类分析 和 深度学习物品分类分析 两个部分

3.3.1 颜色分类结果

方法	优点	缺点	适用场景
RGB 颜色分类	直接基于 RGB 通道亮度值，计算简单，效率高；能够有效区分基本颜色	对复杂颜色或颜色分布不均的图像分类准确性低；对光照条件敏感	适用于颜色分布较为单一、光照条件稳定的图像分类任务
HSV 颜色分类	基于色调、饱和度和亮度分类，符合人类直观感受；对光照变化具有一定的鲁棒性	颜色范围划分主观，需调整阈值；对复杂颜色图像分类结果可能不准确	适用于需要处理不同光照条件、颜色鲜明的图像分类任务
K-Means 聚类	自动提取图像主要颜色，无需定义颜色范围；对多色图像有效	结果受初始中心点和聚类数目影响；计算复杂度高，处理大量图像速度慢	适用于需要提取图像主要颜色特征的场景
颜色偏向分析	细致分析图像在 RGB 三个通道上的偏向情况，提供丰富颜色信息；准确判断偏向方向	对颜色分布均衡的图像无法有效判断偏向；需结合其他方法，单独使用信息有限	适用于需要对颜色进行细致分析，判断颜色倾向性的图像处理任务

1. 红色图片 (red、red2)

- RGB 分类：红色通道亮度显著高于其他两个通道，平均亮度值分别为 243、6、17，R 通道亮度远高于 G 和 B，系统判断为红色。
- HSV 分类：确认为紫色。
- K-Means 主导颜色：聚类结果显示主导颜色为红色，RGB 值接近 (236,3,8)。
- 颜色偏向：红色偏向明显，R 通道亮度远高于 G 和 B 的平均值。



2. 蓝色图片 (blue)

- RGB 分类：蓝色通道亮度最高，平均亮度值分别为 121、204、246，B 通道亮度远超 R 和 G，判断为蓝色。
- HSV 分类：确认为蓝色。
- K-Means 主导颜色：聚类结果主导颜色为蓝色，RGB 值接近（119、203、246）。
- 颜色偏向：蓝色偏向明显，B 通道亮度远高于 R 和 G 的平均值。



3. 绿色图片 (green)

- RGB 分类：绿色通道亮度最高，平均亮度值分别为 101、203、101，G 通道亮度远超 R 和 B，判断为绿色。
- HSV 分类：确认为绿色。

- K-Means 主导颜色：聚类结果主导颜色为绿色，RGB 值接近 (102, 205, 102)。
- 颜色偏向：绿色偏向明显，G 通道亮度远高于 R 和 B 的平均值。



4. 漐变色图片 (image3)

- RGB 分类：根据亮度分布，判断为多色图像。
- HSV 分类：色调范围广，最终结果判断为绿色。
- K-Means 主导颜色：颜色偏向图像的颜色，RGB 值接近 (180,196,217)。
- 颜色偏向：结果为蓝色偏向。



5. 复杂图案图片 (mix3)

- RGB 分类：根据亮度分布，判断为多色图像。
- HSV 分类：为绿色图像。
- K-Means 主导颜色：颜色偏向图像的颜色，RGB 值接近 (31,41,103)。
- 颜色偏向：结果为蓝色偏向。



实验现象：

1. RGB 颜色分类

a. 优点：

- 直接基于图像的 RGB 通道亮度值进行分类，计算简单，效率高。
- 能够有效区分基本颜色，如红色、绿色、蓝色等。

b. 缺点：

- 对于复杂颜色或颜色分布不均的图像，分类准确性可能较低。
- 对光照条件敏感，不同光照下同一种颜色可能被误判。
- 适用场景：适用于颜色分布较为单一、光照条件稳定的图像分类任务。

2. HSV 颜色分类

a. 优点：

- 基于色调 (Hue) 进行分类，更符合人类对颜色的直观感受。
- 对光照变化具有一定的鲁棒性，能够更好地处理不同光照条件下的图像。

b. 缺点：

- i. 颜色范围的划分具有一定的主观性，不同应用场景可能需要调整阈值。
 - ii. 对于颜色混合复杂的图像，分类结果可能不够准确。
- c. 适用场景：适用于需要处理不同光照条件、颜色较为鲜明的图像分类任务。

3. K-Means 聚类提取主导颜色

- a. 优点：
 - i. 能够自动从图像中提取出主要颜色，无需事先定义颜色范围。
 - ii. 对于多色图像，能够有效找出占据主要部分的颜色。
 - b. 缺点：
 - i. 聚类结果受初始中心点和聚类数目影响，可能需要多次实验调整参数。
 - ii. 计算复杂度较高，处理大量图像时速度较慢。
- c. 适用场景：适用于需要提取图像主要颜色特征的场景，如颜色直方图分析、颜色主题提取等。

4. 颜色偏向分析

- a. 优点：
 - i. 能够细致地分析图像在红、绿、蓝三个通道上的偏向情况，提供更丰富的颜色信息。
 - ii. 对于颜色偏向明显的图像，能够准确判断偏向方向。
 - b. 缺点：
 - i. 对于颜色分布均衡的图像，可能无法有效判断偏向。
 - ii. 需要结合其他颜色分析方法，单独使用时信息有限。
- c. 适用场景：适用于需要对颜色进行细致分析，判断颜色倾向性的图像处理任务。

3.3.2 深度学习分类结果

模型	优点	缺点	适用场景
AlexNet	模型结构简单，计算效率高；在 ImageNet 数据集上表现出色，能准确识别大量物体类别	网络深度浅，特征提取能力有限，对复杂图像细节捕捉不足；对小尺寸图像分类效果不佳	适用于对计算资源有限制、需要快速分类的场景，以及对大量常见物体进行分类的任务
VGG16	网络结构更深，提取丰富特征，对图像细节捕捉能力强；在多种分类任务中表现优异，泛化能力强	参数量大，计算复杂度高，训练和推理速度慢；对计算资源要求高，部署在受限设备上困难	适用于需要高精度分类、对图像细节要求较高的场景，以及在计算资源充足的情况下进行物体识别任务

1. 动物图片 (animal1、animal2、animal3)

- AlexNet 分类：准确识别出动物类别，如 animal1 为小熊猫，animal2 为狐狸，animal3 为猫。
- VGG16 分类：与 AlexNet 结果一致，进一步验证分类准确性。

颜色亮度分析

选择文件 animal1.jpg



分析图片 使用标准化

显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 多色图像

颜色分类 (OpenCV): 多色图像

PIL计算的平均亮度:

红色: 114.181
绿色: 101.119
蓝色: 85.414

OpenCV计算的平均亮度:

红色: 114.183
绿色: 101.117
蓝色: 85.413

图像分类结果:

AlexNet 分类结果: lesser_panda
VGG16 分类结果: lesser_panda

OK ⌂

颜色亮度分析

选择文件 animal2.jpg



分析图片 使用标准化

显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 多色图像

颜色分类 (OpenCV): 多色图像

PIL计算的平均亮度:

红色: 107.991
绿色: 134.320
蓝色: 143.105

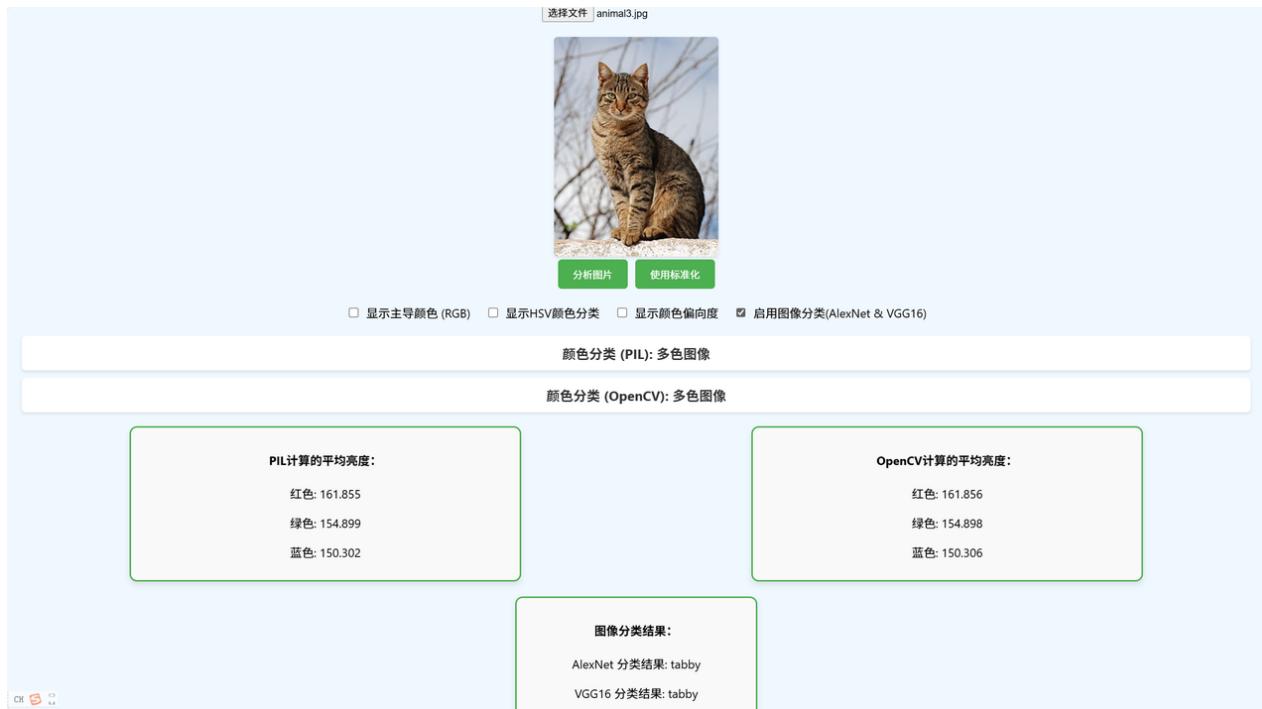
OpenCV计算的平均亮度:

红色: 107.990
绿色: 134.321
蓝色: 143.104

图像分类结果:

AlexNet 分类结果: red_fox
VGG16 分类结果: red_fox

OK ⌂



2. 植物图片 (plant1、plant2、plant3)

- AlexNet 分类：不能准确识别出植物类别，plant1 嫩芽识别错误，plant2 向日葵成功，plant3 的蓝色花朵识别为动物，也错误了。
- VGG16 分类：与 AlexNet 结果不一致，验证分类准确性也很低，plant1 嫩芽识别错误，plant2 向日葵成功，plant3 的蓝色花朵识别为动物，也错误了。



颜色亮度分析

plant2.jpg



显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 红色

颜色分类 (OpenCV): 红色

PIL计算的平均亮度:
 红色: 196.441
 绿色: 169.461
 蓝色: 27.444

OpenCV计算的平均亮度:
 红色: 196.443
 绿色: 169.459
 蓝色: 27.293

图像分类结果:
 AlexNet 分类结果: daisy
 VGG16 分类结果: daisy

plant3.jpg



显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 多色图像

颜色分类 (OpenCV): 多色图像

PIL计算的平均亮度:
 红色: 101.661
 绿色: 122.216
 蓝色: 118.476

OpenCV计算的平均亮度:
 红色: 101.658
 绿色: 122.220
 蓝色: 118.518

图像分类结果:
 AlexNet 分类结果: cabbage_butterfly
 VGG16 分类结果: cabbage_butterfly

3. 自然风景图片 (sea、tree)

- AlexNet 分类：不能准确识别出自然风景类别，如 sea 为大海，tree 为树木。
- VGG16 分类：与 AlexNet 结果一致，但是都不能准确识别出。

颜色亮度分析

选择文件 sea.jpg

显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 蓝色
颜色分类 (OpenCV): 蓝色

PIL计算的平均亮度:

红色: 89.774
绿色: 133.454
蓝色: 178.379

OpenCV计算的平均亮度:

红色: 89.769
绿色: 133.458
蓝色: 178.375

图像分类结果:

AlexNet 分类结果: great_white_shark
VGG16 分类结果: great_white_shark

选择文件 tree.jpg

显示主导颜色 (RGB) 显示HSV颜色分类 显示颜色偏向度 启用图像分类(AlexNet & VGG16)

颜色分类 (PIL): 绿色
颜色分类 (OpenCV): 绿色

PIL计算的平均亮度:

红色: 110.295
绿色: 156.297
蓝色: 56.083

OpenCV计算的平均亮度:

红色: 110.295
绿色: 156.297
蓝色: 56.083

图像分类结果:

AlexNet 分类结果: golf_ball
VGG16 分类结果: golf_ball

实验现象：

1. AlexNet

a. 优点：

- i. 模型结构相对简单，计算效率较高。
- ii. 在 ImageNet 数据集上表现出色，能够准确识别大量物体类别。

b. 缺点：

- i. 网络深度较浅，特征提取能力有限，对于复杂图像的细节特征捕捉不足。
- ii. 对于小尺寸图像的分类效果可能不如更深层次的网络。

- c. 适用场景：适用于对计算资源有限制、需要快速分类的场景，以及对大量常见物体进行分类的任务。

2. VGG16

- a. 优点：

- i. 网络结构 deeper，能够提取更丰富的特征，对图像细节的捕捉能力更强。
 - ii. 在多种图像分类任务中表现出色，泛化能力较强。

- b. 缺点：

- i. 模型参数量大，计算复杂度高，训练和推理速度较慢。
 - ii. 对于计算资源要求较高，部署在资源受限的设备上可能面临挑战。

- c. 适用场景：适用于需要高精度分类、对图像细节要求较高的场景，以及在计算资源充足的情况下进行物体识别任务。

3.3.3 实验结果总结

- RGB 颜色分类 可识别红、绿、蓝等基础色，但遇到紫色、青色等混合色时，容易误判；
- HSV 颜色分类 利用 Hue 值，对黄色、橙色、紫色等颜色的分类准确率明显高于 RGB 方法；
- K-Means 聚类 对纯色图像分类效果很好，但当背景颜色占比较高时，可能误判背景颜色为主导颜色；
- 颜色偏向分析 可用于衡量 RGB 三通道的主导颜色趋势，在色彩单一的图像中较为准确；
- AlexNet 和 VGG16 均能正确分类**大多数动物图像**，但是**对植物和风景的识别比较差**；
- VGG16 由于深度更大（16 层），在部分细粒度分类上更准确；
- 在背景复杂的图像中，分类模型可能受到干扰，导致错误分类

四、总结与展望

本实验综合运用了传统图像处理方法（RGB 亮度、HSV 分类、K-Means 聚类）和深度学习方法（AlexNet/VGG16），对图像颜色和物体类别进行了分析和分类。

4.1 实验总结

本实验通过计算图像的平均亮度，成功利用 OpenCV 和 PIL 对红色、蓝色、绿色物品的图像进行了主导颜色的判断，并使用预训练的深度学习模型对图像中的物体进行了分类。实验结果表明，OpenCV 和 PIL 在图像读取和处理后计算得到的平均亮度值一致，能够准确地判断图像的主导颜色。同时，基

于 HSV 颜色空间的颜色分类方法和 K-Means 计算主导颜色的方法也有效地辅助了颜色分析。预训练模型在图像分类任务中表现良好，能够识别出图像中物体的类别。

4.2 实验创新点

- 1. 多方法颜色分析：**结合了基于平均亮度的颜色判断、HSV 颜色分类以及 K-Means 主导颜色计算等多种方法，从不同角度对图像颜色进行分析，提高了颜色判断的准确性和可靠性。
- 2. 颜色偏向分析：**通过计算各通道亮度与另外两通道亮度平均值的差值，进一步分析了图像颜色的偏向情况，为颜色描述提供了更细致的信息。
- 3. 深度学习模型集成：**集成了 PyTorch 中的预训练模型 AlexNet 和 VGG16，利用其强大的特征提取和分类能力，实现了对图像中物体的准确分类，拓展了图像分析的应用范围。
- 4. 前后端展示：**开发了基于 React 的前端界面和基于 Flask 的后端 API，实现了用户友好的交互式展示，用户可以实时上传图像并获取分析结果。

4.3 未来展望

- 1. 优化颜色分类算法：**可以尝试引入深度学习的颜色分类模型，以提高对复杂颜色的分类能力。例如，训练一个专门用于颜色分类的卷积神经网络，能够自动学习颜色特征并进行分类。
- 2. 扩展数据集：**增加更丰富的颜色类别、不同光照条件下的样本，以及更多种类的物体图像，以提高模型的泛化能力和适应性。
- 3. 结合多模型融合：**结合多个深度学习模型，如 ResNet、Inception 等，通过模型融合提高图像分类的鲁棒性和准确性。
- 4. 模型优化与部署：**对深度学习模型进行量化、剪枝等优化操作，减少模型大小和计算量，提高推理速度，以便在移动设备或嵌入式系统上部署。
- 5. 功能拓展：**可以进一步拓展系统的功能，如添加目标检测、图像分割等功能，为用户提供更全面的图像分析服务。

在未来的实验和研究中，可以进一步优化颜色分析算法，提高对复杂图像场景中颜色判断的准确性。例如，考虑引入更先进的颜色特征提取方法或结合深度学习技术进行颜色分类。此外，可以扩展实验数据集，增加更多种类和更具挑战性的图像，以全面评估系统的性能。同时，对图像分类模型进行微调或使用更强大的模型架构，有望进一步提高分类的准确率和鲁棒性。