

Task 2: Artwork Similarity Detection

In []:

```
import os
import numpy as np
import cv2
import torch
from PIL import Image
from torchvision import models, transforms
from facenet_pytorch import MTCNN, InceptionResnetV1
from scipy.spatial.distance import cosine, euclidean
import mediapipe as mp
from sklearn.metrics.pairwise import cosine_similarity
from PIL import Image
import matplotlib.pyplot as plt
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
    from .autonotebook import tqdm as notebook_tqdm
```

In []:

```
# =====
# 1. 加载数据集
# =====

# 设定数据集路径
train_dir = 'ImgNationalGalleryOfArt' #存放国家美术馆数据集的目录路径
image_size = (224, 224) # 统一尺寸, 因为 ResNet50 和 FaceNet 预期的输入
batch_size = 32 #批处理大小

# 获取所有图片路径
image_paths = [os.path.join(train_dir, fname) for fname in os.listdir]
```

In []:

```
# =====
# 1.1 加载样例
# =====

# 图片示例
sample_image_path = "ImgNationalGalleryOfArt/1024px-Angelo_Marullo,_"

# 读取图像
img = cv2.imread(sample_image_path) # 读取图像 (格式为 BGR)

# OpenCV 默认是 BGR 格式, 转换为 RGB, (Matplotlib 需要 RGB 才能正确显示)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 显示原始图像
plt.figure(figsize=(6, 6))
plt.imshow(img_rgb)
plt.axis("off")
plt.title("Original img")
plt.show()
```

```
# 转换为 PIL 格式
img_pil = Image.fromarray(cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB))
```

Original img



```
In [ ]: # -----
# 2. 预处理
# -----



# 预处理: 归一化 & 颜色调整
preprocess = transforms.Compose([
    # transforms.ToPILImage(), # 把 NumPy 数组转换为 PIL 格式 #####
    transforms.Resize(image_size), # 调整所有图片大小, 确保输入网络的尺寸-
    # 绘画作品的风格可能不同, 所以颜色抖动适应不同风格
    transforms.ColorJitter(
        brightness=0.2, # 在 ±20% 范围内随机调整亮度
        contrast=0.2, # 在 ±20% 范围内随机调整对比度
        saturation=0.2, # 在 ±20% 范围内随机调整颜色饱和
        hue=0.1 # 在 ±10% 范围内随机调整色调
    ),
    transforms.ToTensor(), # 将 PIL.Image 转换为 PyTorch 张量, 并将像素值
    transforms.Normalize( # ImageNet 预训练模型的标准化参数
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    ),
])
```

In []:

```
# =====
# 2.1 样例预处理
# =====

# 图片可能会有轻微的颜色调整 (归一化影响不大)
# 在颜色对比度上可能有细微变化, 但人眼难以察觉

img_preprocess = preprocess(img_pil)
# 转回 numpy 格式, 方便显示
img_np = img_preprocess.permute(1, 2, 0).numpy()
img_np = img_np * np.array([0.229, 0.224, 0.225]) + np.array([0.485
img_np = np.clip(img_np, 0, 1)

# 显示预处理后的图像
plt.figure(figsize=(6, 6))
plt.imshow(img_np)
plt.axis("off")
plt.title("Image after preprocess")
plt.show()
```

Image after preprocess



In []:

```
# =====
# 3. 加载模型 (MTCNN) 人脸检测
# =====

# 用于检测图片中的人脸 (返回多个检测到的面部)
```

```
# MTCNN 适用于检测绘画中的人脸, 比 YOLO 更适合非真实照片
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
mtcnn = MTCNN(keep_all=True, device=device) # 允许检测多个面部
```

```
In [ ]: # =====
# 4. 加载模型 (FaceNet) 人脸特征提取
# =====
```

```
# FaceNet, 计算人脸特征向量
# VGGFace2, 大规模人脸数据集, 预训练模型能够提取高质量的面部特征
facenet = InceptionResnetV1(pretrained='vggface2').eval()
```

```
In [ ]: # =====
# 3.1 & 4.1 对样例进行人脸检测 & 人脸特征提取
# =====
```

```
# 人脸检测
faces = mtcnn(img_pil)

# 显示检测到的人脸
if faces is not None:
    face = faces[0].unsqueeze(0).to(device) # 取第一张人脸, 并增加 batch_size
    with torch.no_grad():
        face_embedding = facenet(face) # 提取特征向量

    print("Face Embedding Shape:", face_embedding.shape) # 这个 512

    # 画出检测到的人脸区域
    img_face = img_rgb.copy()
    boxes, _ = mtcnn.detect(img_pil)
    for box in boxes:
        x1, y1, x2, y2 = map(int, box)
        cv2.rectangle(img_face, (x1, y1), (x2, y2), (255, 0, 0), 2)

    # 显示检测到的人脸
    plt.figure(figsize=(6, 6))
    plt.imshow(img_face)
    plt.axis("off")
    plt.title("Face Detection Result")
    plt.show()
else:
    print("未检测到人脸")
```

```
Face Embedding Shape: torch.Size([1, 512])
```

Face Detection Result



In []:

```
# -----
# 5. 加载模型 (ResNet) 整体风格特征提取
# -----
# ResNet50, 深度残差网络,
# 提取图片的整体视觉特征 (风格、颜色、结构。。。
resnet_model = models.resnet50(pretrained=True)
resnet_model.eval() # 设置为评估模式
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
```

```
warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMGNET1K_V1`. You can also use `weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
```

```
warnings.warn(msg)
```

Out[]: 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): Bottleneck(
              (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
              (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
              (relu): ReLU(inplace=True)
              (downsample): Sequential(
                  (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
              )
          )
      (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
In [ ]: # =====
# 5.1 对样例进行整体风格特征提取
# =====

# 将图像预处理并输入 ResNet50
img_tensor = preprocess(img_pil)
img_tensor = img_tensor.unsqueeze(0) # 添加一个批次维度

# 获取某一层的特征图（例如，第一层卷积层的输出）
def get_intermediate_layers(model, layer_name, input_tensor):
    # 注册一个 hook，获取中间层的输出
    def hook_fn(module, input, output):
        intermediate_output.append(output)

    intermediate_output = []
    hook = dict([*model.named_modules()])[layer_name].register_forward_hook(hook_fn)

    # 执行一次前向传播
    with torch.no_grad():
        model(input_tensor)

    hook.remove() # 移除 hook，避免影响后续的操作
    return intermediate_output[0] # 返回层的输出（特征图）

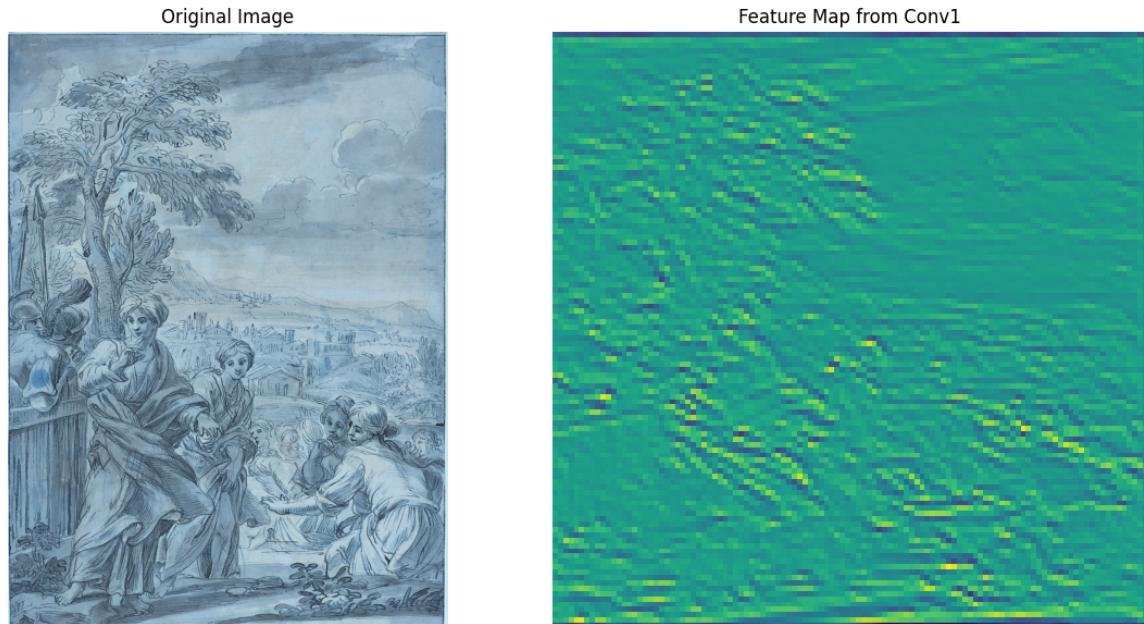
# 提取中间层的特征图（例如，第一层卷积层）
layer_name = 'conv1'
feature_map = get_intermediate_layers(resnet_model, layer_name, img)

# 显示原图
plt.figure(figsize=(12, 6))

# 显示原图
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(np.array(img_rgb), cv2.COLOR_BGR2RGB)) # 显示原图
plt.title("Original Image")
plt.axis('off')

# 显示提取的特征图（假设我们展示第一个特征图）
plt.subplot(1, 2, 2)
# 将特征图从 tensor 转换为 NumPy 格式并绘制
feature_map = feature_map.squeeze().detach().cpu().numpy() # 去掉批次维
plt.imshow(feature_map[0], cmap='viridis') # 显示特征图的第一个通道
plt.title("Feature Map from Conv1")
plt.axis('off')

plt.tight_layout()
plt.show()
```



In []:

```
# =====
# 6. 加载模型 (Mediapipe) 姿态估计
# =====

# Pose() 检测人体姿态, 提取关键点用于计算姿势相似度
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()
```

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1742074743.877992 6960511 gl_context.cc:369] GL version: 2.1 (2.1 Metal - 89.3), renderer: Apple M3 Pro

In []:

```
# =====
# 6.1 对样例进行姿态估计
# =====

# 进行姿态检测
results = pose.process(img_rgb) # 计算 17 个关键点 (肩膀、胳膊、腿等)

# 绘制人体关键点
if results.pose_landmarks:
    img_pose = img_rgb.copy()
    for landmark in results.pose_landmarks.landmark:
        h, w, _ = img_pose.shape
        x, y = int(landmark.x * w), int(landmark.y * h)
        cv2.circle(img_pose, (x, y), 5, (255, 0, 0), -1) # 红点表示

# 显示带关键点的图片
plt.figure(figsize=(6, 6))
plt.imshow(img_pose)
plt.axis("off")
plt.title("Pose Detection Result")
plt.show()

else:
    print("未检测到人体关键点")
```

```
# 输出 17 个关节点的坐标
if results.pose_landmarks:
    keypoints = np.array([[lm.x, lm.y] for lm in results.pose_landmarks])
    print("Extracted Keypoints:", keypoints.shape)
```

未检测到人体关键点

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1742074743.950232 6965135 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inference. Disabling support for feedback tensors.
W0000 00:00:1742074743.962185 6965135 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inference. Disabling support for feedback tensors.
```

```
In [ ]: # 连接关键点的骨架索引
POSE_CONNECTIONS = [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6),
                     (6, 8), (7, 9), (8, 10), (9, 11), (10, 12), (11,
                     (12, 13), (13, 14), (14, 15), (15, 16))

if results.pose_landmarks:
    img_skeleton = img_rgb.copy()
    h, w, _ = img_skeleton.shape

    # 画出人体骨架
    for connection in POSE_CONNECTIONS:
        x1, y1 = int(results.pose_landmarks.landmark[connection[0]].x * w), int(results.pose_landmarks.landmark[connection[0]].y * h)
        x2, y2 = int(results.pose_landmarks.landmark[connection[1]].x * w), int(results.pose_landmarks.landmark[connection[1]].y * h)
        cv2.line(img_skeleton, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # 显示带骨架的图片
    plt.figure(figsize=(6, 6))
    plt.imshow(img_skeleton)
    plt.axis("off")
    plt.title("Human Skeleton")
    plt.show()
```

```
In [ ]: # =====
# 7. 特征提取函数: 人脸特征提取
# =====

def extract_face_features(image):
    """ 提取人脸特征 """
    img_cvt = Image.fromarray(
        cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    )
    faces = mtcnn(img_cvt) # 返回一个检测到的脸部裁剪图像 (可能是多个)
    if faces is None:
        return None # 没检测到人脸, 返回空

    # 取第一张脸 (如果有多个), 忽略多张脸的情况
    face = faces[0].unsqueeze(0)

    # 提取特征
    with torch.no_grad():
        face_embedding = facenet(face) # 提取人脸特征向量, 计算一个 512 维
```

```
    return face_embedding.numpy().flatten() #展平为一维数组, 便于计算相似度
```

```
In [ ]: # =====
# 8. 特征提取函数: 姿态特征提取
# =====

def extract_pose_features(image):
    """ 提取人体关键点特征 """
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #OpenCV 读取的图片是BGR格式
    results = pose.process(img_rgb) #计算人体关键点 (17 个) (肩膀、胳膊、手等)

    if results.pose_landmarks:
        keypoints = np.array([[lm.x, lm.y] for lm in results.pose_landmarks])
        return keypoints.flatten() #展平成一维数组, 用于相似度计算

    return None # 没检测到人体, 返回空
```

```
In [ ]: extract_pose_features(img_rgb)
```

```
In [ ]: # =====
# 8. 特征提取函数: 图片风格和内容
# =====

# 计算一个 2048 维的向量, 表示图片风格和内容
def extract_image_features(image):
    """ 提取整张图片的视觉特征 """
    img_resized = preprocess(Image.fromarray(image)).unsqueeze(0)
    #img_resized = preprocess_image(image)
    with torch.no_grad():
        features = resnet_model(img_resized) # 提取图片特征
    return features.numpy().flatten() # 展平为一维数组
```

```
In [ ]: # =====
# 4. 计算相似性
# =====

def compute_similarity(feature1, feature2, method='cosine'):
    """ 计算特征相似度 """
    if feature1 is None or feature2 is None:
        return 0 # 没有匹配特征时返回 0 相似度
    # 余弦相似度
    if method == 'cosine':
        return 1 - cosine(feature1, feature2) # 两个向量的夹角, 数值在0~1之间
    # 欧几里得距离
    elif method == 'euclidean':
        return 1 / (1 + euclidean(feature1, feature2)) # 两个向量的距离
    else:
        raise ValueError("Unsupported similarity method!")
```

```
In [ ]: # =====
# 5. 计算数据集中的相似度
# =====

# 计算所有图片的特征
```

```

image_features = []
face_features = []
pose_features = []

for path in image_paths:
    img = cv2.imread(path)

    # 提取特征
    image_feat = extract_image_features(img)
    face_feat = extract_face_features(img)
    pose_feat = extract_pose_features(img)

    image_features.append(image_feat)
    face_features.append(face_feat)
    pose_features.append(pose_feat)

# 转换为 NumPy 数组
image_features = np.array(image_features)
face_features = np.array(face_features, dtype=object) # 可能有 None
pose_features = np.array(pose_features, dtype=object)

```

W0000 00:00:1742074749.305913 landmark_projection_calculate
r.cc:186] Using NORM_RECT without IMAGE_DIMENSIONS is only supported
for the square ROI. Provide IMAGE_DIMENSIONS or use PROJECTION_MATRIX.

In []: # =====
6. 评估相似性
=====

```

def find_most_similar(target_idx, top_k=5):
    """ 在数据集中找到与目标图片最相似的前 K 张 """
    target_img_feat = image_features[target_idx]
    target_face_feat = face_features[target_idx]
    target_pose_feat = pose_features[target_idx]

    similarities = []

    for i in range(len(image_paths)):
        if i == target_idx:
            continue

        img_sim = compute_similarity(target_img_feat, image_feature)
        face_sim = compute_similarity(target_face_feat, face_feature)
        pose_sim = compute_similarity(target_pose_feat, pose_feature)

        # 计算综合相似度 (加权平均)
        total_sim = (0.5 * img_sim) + (0.3 * face_sim) + (0.2 * pose_sim)
        similarities.append((image_paths[i], total_sim))

    # 排序并返回最相似的 top_k 图片
    similarities.sort(key=lambda x: x[1], reverse=True)
    return similarities[:top_k]

```

In []: # 设定查询图片的数量
num_queries = 3 # 展示10副图像

```

fig, axes = plt.subplots(num_queries, 6, figsize=(20, num_queries * 12))

# 遍历不同的查询图片
for idx in range(num_queries):
    query_idx = idx # 选择查询图片的索引
    similar_images = find_most_similar(query_idx)

    # 获取目标图片路径
    query_img_path = image_paths[query_idx]
    query_img = cv2.imread(query_img_path) # 读取目标图片

    # 显示目标图片
    axes[idx, 0].imshow(cv2.cvtColor(query_img, cv2.COLOR_BGR2RGB))
    axes[idx, 0].axis('off') # 关闭坐标轴
    axes[idx, 0].set_title(f"Target Img {query_idx + 1}")

    # 显示与目标图片最相似的 5 张图片
    for i, (img_path, sim_score) in enumerate(similar_images[:5]):
        img = cv2.imread(img_path) # 读取相似图片
        axes[idx, i + 1].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        axes[idx, i + 1].axis('off') # 关闭坐标轴
        axes[idx, i + 1].set_title(f"Sim: {sim_score:.4f}")

    # 调整图像显示
    plt.tight_layout()
    plt.show()

```

