实验 8 视频背景替换

8.1 实验概述

人像视频背景替换实验是借助语义分割算法与计算机视觉技术，实现视频序列中人物主体与原始背景的像素级分离，并融合新场景以构建自然视觉效果的实践研究。其核心在于通过语义分割模型达成高精度、高实时性的人像提取，为背景替换提供可靠的技术支撑。实验的关键环节是语义分割模型对人像区域的精准识别：模型需逐帧分析视频内容，通过多层次特征提取定位人物整体区域，并精细勾勒轮廓边缘，尤其要处理发丝、透明饰品等细节部位，解决传统方法中边缘模糊、细节丢失的问题，这对算法的特征捕捉能力和分割精度提出了极高要求。为实现实用化效果，实验需平衡分割质量与处理速度：在实时场景（如在线会议、实时直播）中，系统需在极短时间内完成单帧分割与图像合成，这不仅依赖模型的结构优化（如轻量化设计、推理加速），还与硬件算力密切相关。最终，实验先由模型输出人像区域的二值掩码，再利用图像融合技术将人像前景与新背景叠加，通过调整光影强度、色彩色调的一致性，使前景与新背景自然融合，最终生成视觉协调、边缘过渡平滑的视频效果。

图 8-1 视频背景替换示例

8.2 实验目的

- 了解视频背景替换的常用方法；
- 学习利用 Segformer 模型实现视频背景替换；
- 理解物体分割数据集和评价标准。

8.3 实验内容

8.3.1 视频背景替换常用方法

视频背景替换是计算机视觉领域的一个重要应用。随着深度学习技术的发展，视频背景替换取得了显著的进步。常用的视频背景替换方法包括传统方法和基于深度学习的方法。

（1）传统视频背景替换方法

传统方法以“像素特征差异”为核心，无需复杂模型训练，替换速度快，但对动态场景（如光照变化、前景遮挡）鲁棒性较差，适用于简单静态背景场景。

表 7-1 传统视频背景替换方法

方法	介绍
帧差法	基于相邻帧像素灰度值差异分割前景：计算连续 2~3 帧的像素差值，超过阈值的像素判定为前景（运动区域），其余为背景。优点是计算速度快、实现简单；缺点是易受光照波动影响，易产生 “空洞” 或 “拖影”，无法处理静态前景（如静止的人物）。
背景减法	先构建并动态更新“背景模型”，再将当前帧与背景模型对比，差异像素判定为前景。相比帧差法能适应缓慢光照变化，但对快速运动目标易出现分割不完整，且复杂背景（如抖动的树叶）易产生大量误检。
光流法	基于像素运动矢量分割前景：通过计算相邻帧中每个像素的运动方向与速度，将运动矢量显著的像素归为前景，静止像素归为背景。可捕捉目标运动轨迹，但计算复杂度高、实时性差，且纹理单一区域（如纯色墙面）易出现光流估计失效。

（2）基于深度学习的视频背景替换方法

基于深度学习的方法以语义分割为核心，能从像素级理解图像内容，区分人物、背景等语义类别，分割精度与鲁棒性远超传统方法，其中 Segformer 凭借 “高精度 + 轻量化” 特性，成为视频背景替换（尤其是中小目标、边缘精细场景）的优选方案之一。

表 7-2 基于深度学习的视频背景替换方法

方法	介绍
Segformer	基于 Transformer 的轻量级语义分割模型，采用 “分层 Transformer 编码器+轻量 MLP 解码器”架构，无复杂卷积操作，参数量低。Transformer 通过提取多尺度特征，输出像素级语义掩码，标记“person”等目标为前景，实现视频背景替换；其边缘分割精准，利于捕捉发丝、衣物褶皱，并且轻量化，实时性高，抗光照变化、遮挡能力强，帧间结果连贯无闪烁。适用于直播背景替换、会议虚拟背景、短视频人像换景等场景。
DeepLabV3	基于 CNN 的经典语义分割模型，以 ResNet 为骨干，核心模块为“空洞卷积+空间金字塔池化（ASPP）”；DeepLabV3 利用空洞卷积扩大感受野，ASPP 融合多尺度特征，强化全局上下文建模，精准区分前景与复杂背景；其复杂背景分割能力强，抗局部噪声（飘动树叶）。适用于户外背景替换、监控视频目标提取、复杂场景短视频后期等场景。
UNet	基于“编码器-解码器”对称架构的经典语义分割模型，核心设计是“跳跃连接”——将编码器提取的高分辨率局部特征（含边缘、纹理细节）与解码器的低分辨率全局特征融合，实现精准的像素级分割。在视频背景替换中，U-Net 能快速定位前景目标（如人物），通过对称结构平衡特征提取与细节恢复，边缘分割精度较高（可捕捉手部、发丝等精细区域）；其变体 U-Net++通过嵌套式密集连接进一步优化特征融合，减少梯度消失问题，提升复杂场景（如人物与背景颜色相近）的分割鲁棒性。

	但相比 Segformer，U-Net 参数量略高，实时性稍弱，更适用于离线视频背景替换（如短视频后期剪辑）、低动态场景。
Mask R-CNN	基于“目标检测+实例分割”双任务架构的模型，先通过 Faster R-CNN 检测图像中的目标，再在检测框内进行像素级实例分割，输出前景目标的精确掩码。在视频背景替换中，Mask R-CNN 的优势在于 “同时实现目标定位与分割”，能有效避免背景中相似物体（如玩偶、雕塑）的误分割，尤其适合多目标场景（如视频中多人同时出现）；其分割掩码为实例级，可单独对每个前景目标进行背景替换，灵活性高。但模型复杂度较高，推理速度较慢，更适用于对精度要求高、实时性要求较低的场景。

8.3.2 Segformer 模型

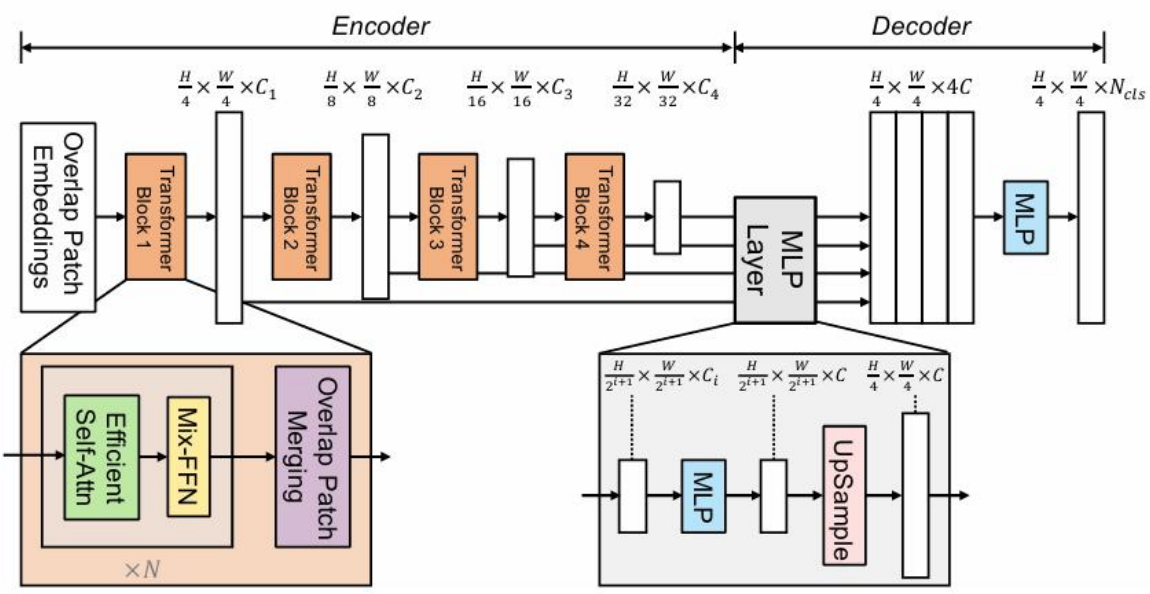
SegFormer 是一款融合无位置编码的 Transformer 编码器与轻量级全连接多层感知机 MLP 解码器的语义分割框架，核心优势在于 结构简洁、效率高、精度优。

（1）总体结构

Segformer 框架主要由两个模块组成。一个分层 Transformer 编码器用于提取粗糙和细致的特征；以及一个轻量级的 All-MLP 解码器，用于直接融合这些多级特征并预测语义分割掩码。具体流程如下：

首先，输入一张分辨率为 H×W×3 的 RGB 图像，将其划分为 4×4 的细粒度 Patch；随后，将这些 4×4 大小的 Patch 输入至分层的 Transformer 编码器。编码器包含 4 个递进的 Stage 阶段，每个 Stage 通过重叠 Patch 合并 Overlapped Patch Merging 模块与高效自注意力 Efficient Self-Attention 模块，逐步输出多尺度特征图 F1~F4。

随后，将编码器输出的 4 个尺度特征图 F1~F4 送入轻量级全 MLP 解码器完成特征融合与分割预测生成分辨率为 H/4×W/4×Ncls 的分割特征图，其中 Ncls 为语义分割任务的目标类别数。该特征图通过简单的上采样恢复至与输入图像一致的分辨率后，每个像素对应一个类别概率分布，取概率最大的类别作为该像素的最终分割结果，完成语义分割任务。



（2）分层的 Transformer 编码器

Segformer 设计了一系列具有相同架构但不同大小的混合 Transformer 编码器 MiT-B0~MiT-B5，其中 MiT-B0 是轻量级预测模型，MiT-B5 是性能最好最大的模型。MiT 共有以下四个特点：

分层特征表示。区别于 ViT 只能获得单一分辨率特征图，MiT 可以输入一张图，产生多尺度特征图，这些多尺度特征图提供高分辨率全局特征到低分辨率局部特征，有助于提高语义分割性能。

高效自注意力。编码器的计算瓶颈在自注意力层。原自注意力过程如公式（8-1）所示，计算复杂度为 $\mathrm{O}\left(\mathrm{N}^2\right)$。

$$\mathrm{Attention}(Q,K,V)=Soft\max\left(\frac{QK^T}{\sqrt{d_{head}}}\right)V \tag{8-1}$$

Segformer 引入缩放因子 R 来降低计算的复杂度，如公式（8-2）所示，其中 K 是要缩减的序列，$\mathrm{Reshape}\left(\frac{\mathrm{N}}{\mathrm{R}},\ \mathrm{C}\cdot\mathrm{R}\right)$ 表示将 K 的维度 (N,C) 重塑成 $\left(\frac{\mathrm{N}}{\mathrm{R}},\ \mathrm{C}\cdot\mathrm{R}\right)$，$\mathrm{Linear}$ 表示一个卷积，将 $\hat{\mathrm{K}}$ 的维度从 $\left(\frac{\mathrm{N}}{\mathrm{R}},\ \mathrm{C}\cdot\mathrm{R}\right)$ 变成 $\left(\frac{\mathrm{N}}{\mathrm{R}},\ \mathrm{C}\right)$

$$\begin{aligned}\hat{\mathrm{K}}&=\mathrm{Reshape}\left(\frac{\mathrm{N}}{\mathrm{R}},\ \mathrm{C}\cdot\mathrm{R}\right)(\mathrm{K})\\ \mathrm{K}&=\mathrm{Linear}(\mathrm{C}\cdot\mathrm{R},\ \mathrm{C})\left(\hat{\mathrm{K}}\right)\end{aligned} \tag{8-2}$$

重叠补丁合并。学习 ViT 中的补丁合并模块可以很容易的将特征图的分辨率缩小两倍，然后将层次结构中的任何特征图进行迭代。但原 ViT 补丁合并模块是设计用于合并非重叠特征或图像补丁，无法保证这些补丁周围的局部连续性。Segformer 为了让该补丁合并模块适用于重叠补丁合并任务，设计了 K、S 和 P，K 是补丁大小，S 是相邻补丁之间的步长，P 是填充大小，以生成与非重叠合并补丁模块相同大小的特征。

Mix-FFN。ViT 使用位置编码来引入位置信息，其要求固定的分辨率。这会导致训练图像和测试图像的分辨率不同，需要对位置编码进行插值操作，导致精度下降。为解决该问题，Segformer 引入了一个 Mix-FFN，它考虑了零填充过程对泄露位置信息的影响，故直接在前馈网络 FFN 中使用了一个 3×3 卷积。Mix-FFN 可以表示如下：

$$x_{out}=MLP\left(GELU\left(Conv_{3\times3}\left(MLP\left(x_{in}\right)\right)\right)\right)+x_{in}$$

（3）轻量级的全 MLP 解码器

Segformer 使用了一个轻量级的解码器，该解码器仅由 MLP 层组成，避免了其他方法中通常使用的手工设计和计算量较大的组件。这种编码器的核心是多级 Transformer Encoder 架构，该架构比传统的 CNN Encoder 可以获得更大的感受野。

ALL-MLP 由四步组成。第一，从 MIT 中获取多层次 feature，记作 F_i，通过 MLP 层进行通道维度统一。第二，F_i 上采样到 1/4 大小，并进行拼接。第三，；利用 MLP 层对拼接之后的特征进行融合。最后，另一个 MLP 层对融合的特征进行预测，输出分辨率为 $\frac{H}{4} \times \frac{W}{4} \times N_{cls}$。

（4）损失函数

交叉熵损失。它是是分类任务中常用的损失函数，基于预测概率与真实标签的分布差异计算损失。对于语义分割任务，其通过衡量每个像素点的预测概率分布与真实类别标签（通常为独热编码形式）之间的交叉熵来优化模型，公式如下，其中 y_i 为真实标签，p_i 为预测标签。它能有效引导模型学习类别间的判别特征，但在类别不平衡时可能偏向多数类。

$$Loss = -\sum_i y_i \log(p_i)$$

Dice 损失。Dice 损失基于 Dice 系数（衡量两个集合相似度的指标）设计，公式如下。其中 X 为预测结果，Y 为真实标签。它对边缘细节敏感，适用于解决类别不平衡问题（如小目标分割），能有效提升模型对前景目标的分割精度，但在类别较多且 batch_size 较小时可能存在训练不稳定的问题。

$$Loss = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$$

（5）速度与精度

SegFormer 的特性与该视频人像背景替换项目需求高度契合，成为理想的语义分割核心模块。视频人像背景替换的核心需求是：实时或近实时的推理速度（保证视频流畅播放，通常需≥24 FPS）、精准的人像分割精度（避免人像边缘模糊、漏分割或误分割，确保背景替换自然）、较低的硬件资源占用（适配普通消费级显卡）。SegFormer 在这些方面均能提供有效支撑。

在权威数据集上，SegFormer-B5（最大模型）表现尤为突出：在 Cityscapes 验证集上达到 84.0%的 mIoU，较此前的 SETR 模型提升 1.8%，且在 ADE20K 数据集上以 51.8%的 mIoU 刷新当时 SOTA，同时在 COCO-Stuff 数据集上也保持领先精度。即使是轻量级模型 SegFormer-B0，在 ADE20K 也能实现 37.4%的 mIoU，远超同量级的 DeepLabV3+等模型，充分证明其在不同复杂度需求下的精度优势。

实验数据显示，在无 TensorRT 等加速工具的情况下，轻量级模型 SegFormer-B0 在 Cityscapes 数据集上表现亮眼：当输入图像短边为 512 时，推理速度达 47.6 FPS，较经典实时分割模型 ICNet 快 17.3 FPS，同时 mIoU 提升 4.2%；即使是最大模型 SegFormer-B5，推理速度也比 SETR 快 5 倍，参数规模仅为 SETR 的 1/4，在保证高精度的同时，有效降低了硬件资源占用。

8.3.3 数据集

该项目数据集基于 VOC2012 数据集。VOC2012 支持目标检测、语义分割、实例分割等多类计算机视觉任务。其核心标注类别共 21 类，包含一个背景类和 20 个目标物体类，覆盖日常场景中常见物体，如人、汽车、猫等。其因标注完整、场景多样，成为计算机视觉算法开发的“基准测试库”。

该项目数据集扫描 VOC2012，筛选 VOC2012 数据集中包含“person”（类别 ID 为 15）的样本。根据支持的 VOC 格式创建“VOC2012_Segmentation_Person”目录。遍历所有样本的分割标注，将其转为 numpy 数组，检测标注中是否存在“person”像素（值为 15），若存在，则将该样本图像和分割标注复制到输出目录。最后统一生成 train.txt 等划分文件。VOC2012 语义分割任务原始样本总数 2913，筛选保留含“person”的样本数 888，保留比例 30.48%。下面是数据集筛选方法 filterperson.py 实现：

```
import os
import numpy as np
import shutil
from PIL import Image

def filter_voc2012_segmentation_with_person(voc_root, output_root=None):
    """
    筛选 VOC2012 语义分割数据集中包含"person"（类别 ID=15）的样本
    :param voc_root: VOC2012 数据集根目录（如 "VOCdevkit/VOC2012"）
    :param output_root:  筛选后数据集的输出目录，默认在原目录同级创建 "VOC2012_Segmentation_Person"
    """
    # -------------------------- 1. 定义原始数据集路径（语义分割核心目录）--------------------------
    # 原始图像目录（JPEG 格式）
    img_dir = os.path.join(voc_root, "JPEGImages")
    # 语义分割标注目录（PNG 格式，像素值对应类别 ID）
    seg_annot_dir = os.path.join(voc_root, "SegmentationClass")
    # 语义分割任务的数据集划分文件目录（train.txt/val.txt/trainval.txt）
    seg_set_dir = os.path.join(voc_root, "ImageSets", "Segmentation")

    # 检查原始目录是否存在
    required_dirs = [img_dir, seg_annot_dir, seg_set_dir]
    for dir_path in required_dirs:
        if not os.path.exists(dir_path):
            raise FileNotFoundError(f"原始目录不存在：{dir_path}\n 请确认 VOC2012 数据集路径正确")

    # -------------------------- 2. 创建输出目录（避免修改原始数据）--------------------------
    if output_root is None:
        # 默认在原数据集同级目录创建新文件夹
        parent_dir = os.path.dirname(voc_root)
        output_root = os.path.join(parent_dir, "VOC2012_Segmentation_Person")

    # 输出目录结构与原始 VOC 保持一致
    output_img = os.path.join(output_root, "JPEGImages")  # 筛选后的图像
    output_seg_annot = os.path.join(output_root, "SegmentationClass")  # 筛选后的分割标注
    output_seg_set = os.path.join(output_root, "ImageSets", "Segmentation")  # 筛选后的划分文件
    # 创建目录（已存在则跳过）
    for dir_path in [output_img, output_seg_annot, output_seg_set]:
        os.makedirs(dir_path, exist_ok=True)

    # -------------------------- 3. 定义关键参数--------------------------
```

```
person_class_id = 15  # VOC2012 语义分割中"person"的类别 ID（固定为 15，背景为 0）
kept_image_ids = []  # 存储包含"person"的样本 ID（无后缀）
total_samples = 0  # 原始样本总数
kept_samples = 0  # 保留的样本数

# -------------------------- 4. 读取语义分割的数据集划分文件（train/val/trainval）--------------------------
# 先获取所有分割任务的样本 ID（避免遍历全部图像，只处理划分文件中的样本）
seg_set_files = ["train.txt", "val.txt", "trainval.txt"]
all_seg_image_ids = set()
for set_file in seg_set_files:
    set_path = os.path.join(seg_set_dir, set_file)
    with open(set_path, "r", encoding="utf-8") as f:
        for line in f:
            img_id = line.strip()
            if img_id:
                all_seg_image_ids.add(img_id)
total_samples = len(all_seg_image_ids)
print(f"VOC2012 语义分割任务原始样本总数：{total_samples}")

# -------------------------- 5. 筛选包含"person"的样本--------------------------
print("\n 开始筛选包含'person'的样本...")
for img_id in all_seg_image_ids:
    # 拼接图像和分割标注的完整路径
    img_path = os.path.join(img_dir, f"{img_id}.jpg")  # 原始图像（JPEG）
    seg_annot_path = os.path.join(seg_annot_dir, f"{img_id}.png")  # 分割标注（PNG）

    # 检查文件是否存在（避免数据集损坏导致的缺失）
    if not os.path.exists(img_path):
        print(f"警告：图像文件缺失 → {img_path}，跳过该样本")
        continue
    if not os.path.exists(seg_annot_path):
        print(f"警告：分割标注文件缺失 → {seg_annot_path}，跳过该样本")
        continue

    # 读取分割标注，判断是否包含"person"（类别 ID=15）
    # VOC 分割标注为单通道 PNG，像素值直接对应类别 ID（0=背景，1=aeroplane，...，15=person）
    seg_annot = Image.open(seg_annot_path)
    seg_annot_np = np.array(seg_annot)  # 转换为 numpy 数组（shape: [H, W]）

    # 检查标注中是否存在 person 的像素（值为 15）
    if np.any(seg_annot_np == person_class_id):
        # 保留该样本：复制图像和分割标注到输出目录
        shutil.copy2(img_path, os.path.join(output_img, f"{img_id}.jpg"))
        shutil.copy2(seg_annot_path, os.path.join(output_seg_annot, f"{img_id}.png"))
        kept_image_ids.append(img_id)
        kept_samples += 1

        # 每处理 100 个样本显示进度
        if kept_samples % 100 == 0:
            print(f"已筛选 {kept_samples} 个含'person'的样本")

# -------------------------- 6. 生成筛选后的分割任务划分文件（train/val/trainval）--------------------------
print("\n 生成筛选后的数据集划分文件...")
for set_file in seg_set_files:
```

```
            input_set_path = os.path.join(seg_set_dir, set_file)    # 原始划分文件
            output_set_path = os.path.join(output_seg_set, set_file)    # 输出划分文件

            # 只保留包含在 kept_image_ids 中的样本 ID
            with open(input_set_path, "r", encoding="utf-8") as f_in, \
                    open(output_set_path, "w", encoding="utf-8") as f_out:
                for line in f_in:
                    img_id = line.strip()
                    if img_id in kept_image_ids:
                        f_out.write(f"{img_id}\n")    # 写入保留的样本 ID

    # -------------------------- 7. 输出筛选结果统计--------------------------
    print("\n" + "=" * 60)
    print("VOC2012 语义分割数据集筛选完成！")
    print(f"原始样本总数：{total_samples}")
    print(f"保留含'person'的样本数：{kept_samples}")
    print(f"保留比例：{kept_samples / total_samples:.2%}")
    print(f"\n 筛选后数据集路径：")
    print(f"- 图像目录：{output_img}")
    print(f"- 分割标注目录：{output_seg_annot}")
    print(f"- 划分文件目录：{output_seg_set}")
    print("=" * 60)


if __name__ == "__main__":
    # -------------------------- 原 VOC2012 数据集根目录 --------------------------
    VOC2012_ROOT = "F:\VOCdevkit\VOC2012"

    # 执行筛选
    filter_voc2012_segmentation_with_person(VOC2012_ROOT)
```

图 8-2 VOC2012_Segmentation_Person 数据集概览

8.3.4 评价标准

交并比（mIoU）：语义分割任务核心指标，计算所有类别“交集-并集比”的平均值，数值越高（0-100%）表示分割精度越高。

平均像素准确率（mPA）：计算每个类别像素准确率的平均值，避免类别不平衡对整体准确率的干扰。

8.4 实验步骤

8.4.1 环境配置

Python 3.9
Pytorch 2.8.2
Opencv-python 4.8.0
Numpy 1.21.5
Pillow 9.0.1

8.4.2 数据预处理

下面是 SegmentationDataset 类的实现，该类继承自 PyTorch 的 Dataset，核心功能是从 VOC 格式的语义分割数据集中读取单个样本，并完成图像与标注的同步预处理了，包括格式统一、尺寸调整、数据增强等，为后续模型训练提供合规输入。

```
class SegmentationDataset(Dataset):
    def __init__(self, annotation_lines, input_shape, num_classes, train, dataset_path):
        super(SegmentationDataset, self).__init__()
```

```
        self.annotation_lines   = annotation_lines
        self.length             = len(annotation_lines)
        self.input_shape        = input_shape
        self.num_classes        = num_classes
        self.train              = train
        self.dataset_path       = dataset_path

    def __len__(self):
        return self.length

    def __getitem__(self, index):
        annotation_line = self.annotation_lines[index]
        name            = annotation_line.split()[0]

        #-------------------------------#
        #   从文件中读取图像
        #-------------------------------#
        jpg             = Image.open(os.path.join(os.path.join(self.dataset_path, "VOC2007/JPEGImages"), name + ".jpg"))
        png             = Image.open(os.path.join(os.path.join(self.dataset_path, "VOC2007/SegmentationClass"), name + ".png"))
        #-------------------------------#
        #   数据增强
        #-------------------------------#
        jpg, png        = self.get_random_data(jpg, png, self.input_shape, random = self.train)

        jpg             = np.transpose(preprocess_input(np.array(jpg, np.float64)), [2,0,1])
        png             = np.array(png)
        png[png >= self.num_classes] = self.num_classes
        #-------------------------------------------------------#
        #   转化成 one_hot 的形式
        #   在这里需要+1 是因为 voc 数据集有些标签具有白边部分
        #   我们需要将白边部分进行忽略，+1 的目的是方便忽略。
        #-------------------------------------------------------#
        seg_labels  = np.eye(self.num_classes + 1)[png.reshape([-1])]
        seg_labels  = seg_labels.reshape((int(self.input_shape[0]), int(self.input_shape[1]), self.num_classes + 1))

        return jpg, png, seg_labels

    def rand(self, a=0, b=1):
        return np.random.rand() * (b - a) + a

    def get_random_data(self, image, label, input_shape, jitter=.3, hue=.1, sat=0.7, val=0.3, random=True):
        image   = cvtColor(image)
        label   = Image.fromarray(np.array(label))
        #-------------------------------#
        #   获得图像的高宽与目标高宽
        #-------------------------------#
        iw, ih  = image.size
        h, w    = input_shape

        if not random:
            iw, ih  = image.size
            scale   = min(w/iw, h/ih)
            nw      = int(iw*scale)
            nh      = int(ih*scale)

            image       = image.resize((nw,nh), Image.BICUBIC)
```

```
            new_image      = Image.new('RGB', [w, h], (128,128,128))
            new_image.paste(image, ((w-nw)//2, (h-nh)//2))

            label              = label.resize((nw,nh), Image.NEAREST)
            new_label      = Image.new('L', [w, h], (0))
            new_label.paste(label, ((w-nw)//2, (h-nh)//2))
            return new_image, new_label

        #------------------------------------------#
        #     对图像进行缩放并且进行长和宽的扭曲
        #------------------------------------------#
        new_ar = iw/ih * self.rand(1-jitter,1+jitter) / self.rand(1-jitter,1+jitter)
        scale = self.rand(0.5, 2)
        if new_ar < 1:
            nh = int(scale*h)
            nw = int(nh*new_ar)
        else:
            nw = int(scale*w)
            nh = int(nw/new_ar)
        image = image.resize((nw,nh), Image.BICUBIC)
        label = label.resize((nw,nh), Image.NEAREST)

        #------------------------------------------#
        #     翻转图像
        #------------------------------------------#
        flip = self.rand()<.5
        if flip:
            image = image.transpose(Image.FLIP_LEFT_RIGHT)
            label = label.transpose(Image.FLIP_LEFT_RIGHT)

        #------------------------------------------#
        #     将图像多余的部分加上灰条
        #------------------------------------------#
        dx = int(self.rand(0, w-nw))
        dy = int(self.rand(0, h-nh))
        new_image = Image.new('RGB', (w,h), (128,128,128))
        new_label = Image.new('L', (w,h), (0))
        new_image.paste(image, (dx, dy))
        new_label.paste(label, (dx, dy))
        image = new_image
        label = new_label

        image_data          = np.array(image, np.uint8)
        #------------------------------------------#
        #     高斯模糊
        #------------------------------------------#
        blur = self.rand() < 0.25
        if blur:
            image_data = cv2.GaussianBlur(image_data, (5, 5), 0)

        #------------------------------------------#
        #     旋转
        #------------------------------------------#
        rotate = self.rand() < 0.25
        if rotate:
            center          = (w // 2, h // 2)
```

```
        rotation    = np.random.randint(-10, 11)
        M           = cv2.getRotationMatrix2D(center, -rotation, scale=1)
        image_data  = cv2.warpAffine(image_data, M, (w, h), flags=cv2.INTER_CUBIC, borderValue=(128,128,128))
        label       = cv2.warpAffine(np.array(label, np.uint8), M, (w, h), flags=cv2.INTER_NEAREST, borderValue=(0))

    #---------------------------------#
    #   对图像进行色域变换
    #   计算色域变换的参数
    #---------------------------------#
    r                   = np.random.uniform(-1, 1, 3) * [hue, sat, val] + 1
    #---------------------------------#
    #   将图像转到 HSV 上
    #---------------------------------#
    hue, sat, val   = cv2.split(cv2.cvtColor(image_data, cv2.COLOR_RGB2HSV))
    dtype           = image_data.dtype
    #---------------------------------#
    #   应用变换
    #---------------------------------#
    x           = np.arange(0, 256, dtype=r.dtype)
    lut_hue = ((x * r[0]) % 180).astype(dtype)
    lut_sat = np.clip(x * r[1], 0, 255).astype(dtype)
    lut_val = np.clip(x * r[2], 0, 255).astype(dtype)

    image_data = cv2.merge((cv2.LUT(hue, lut_hue), cv2.LUT(sat, lut_sat), cv2.LUT(val, lut_val)))
    image_data = cv2.cvtColor(image_data, cv2.COLOR_HSV2RGB)

    return image_data, label
```

8.4.3 模型训练

模型训练文件 train.py。主要包含训练参数配置、模型初始化、数据加载与预处理关联、训练流程控制等核心内容。该文件中详细设置了冻结 / 解冻训练策略（包括不同阶段的 epoch 数、batch size）、优化器及学习率相关参数，指定了实验数据集路径和输入图像尺寸，通过导入的 SegmentationDataset 类和 seg_dataset_collate 函数关联数据预处理与加载逻辑，并支持日志记录、模型保存等功能，整体实现了从参数配置到模型训练循环的完整流程控制。

```
    # ---------------------------------------#
    #   开始模型训练
    # ---------------------------------------#
    for epoch in range(Init_Epoch, UnFreeze_Epoch):
        # ---------------------------------------#
        #   如果模型有冻结学习部分
        #   则解冻，并设置参数
        # ---------------------------------------#
        if epoch >= Freeze_Epoch and not UnFreeze_flag and Freeze_Train:
            batch_size = Unfreeze_batch_size

            # -------------------------------------------------------------------#
            #   判断当前 batch_size，自适应调整学习率
            # -------------------------------------------------------------------#
            nbs = 16
            lr_limit_max = 1e-4 if optimizer_type in ['adam', 'adamw'] else 5e-2
            lr_limit_min = 3e-5 if optimizer_type in ['adam', 'adamw'] else 5e-4
            Init_lr_fit = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
```

```
                Min_lr_fit = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)
                # ---------------------------------------#
                #   获得学习率下降的公式
                # ---------------------------------------#
                lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit, UnFreeze_Epoch)

                for param in model.backbone.parameters():
                    param.requires_grad = True

                epoch_step = num_train // batch_size
                epoch_step_val = num_val // batch_size

                if epoch_step == 0 or epoch_step_val == 0:
                    raise ValueError("数据集过小，无法继续进行训练，请扩充数据集。")

                gen = DataLoader(train_dataset, shuffle=shuffle, batch_size=batch_size, num_workers=num_workers,
                                 pin_memory=True,
                                 drop_last=True, collate_fn=seg_dataset_collate, sampler=train_sampler,
                                 worker_init_fn=partial(worker_init_fn, rank=rank, seed=seed))
                gen_val = DataLoader(val_dataset, shuffle=shuffle, batch_size=batch_size, num_workers=num_workers,
                                     pin_memory=True,
                                     drop_last=True, collate_fn=seg_dataset_collate, sampler=val_sampler,
                                     worker_init_fn=partial(worker_init_fn, rank=rank, seed=seed))

                UnFreeze_flag = True

            if distributed:
                train_sampler.set_epoch(epoch)

            set_optimizer_lr(optimizer, lr_scheduler_func, epoch)

            fit_one_epoch(model_train, model, loss_history, eval_callback, optimizer, epoch, epoch_step, epoch_step_val,
                          gen, gen_val, UnFreeze_Epoch, Cuda, \
                          dice_loss, focal_loss, cls_weights, num_classes, fp16, scaler, save_period, save_dir,
                          local_rank)

            if distributed:
                dist.barrier()

        if local_rank == 0:
            loss_history.writer.close()
```

8.4.4 模型测试

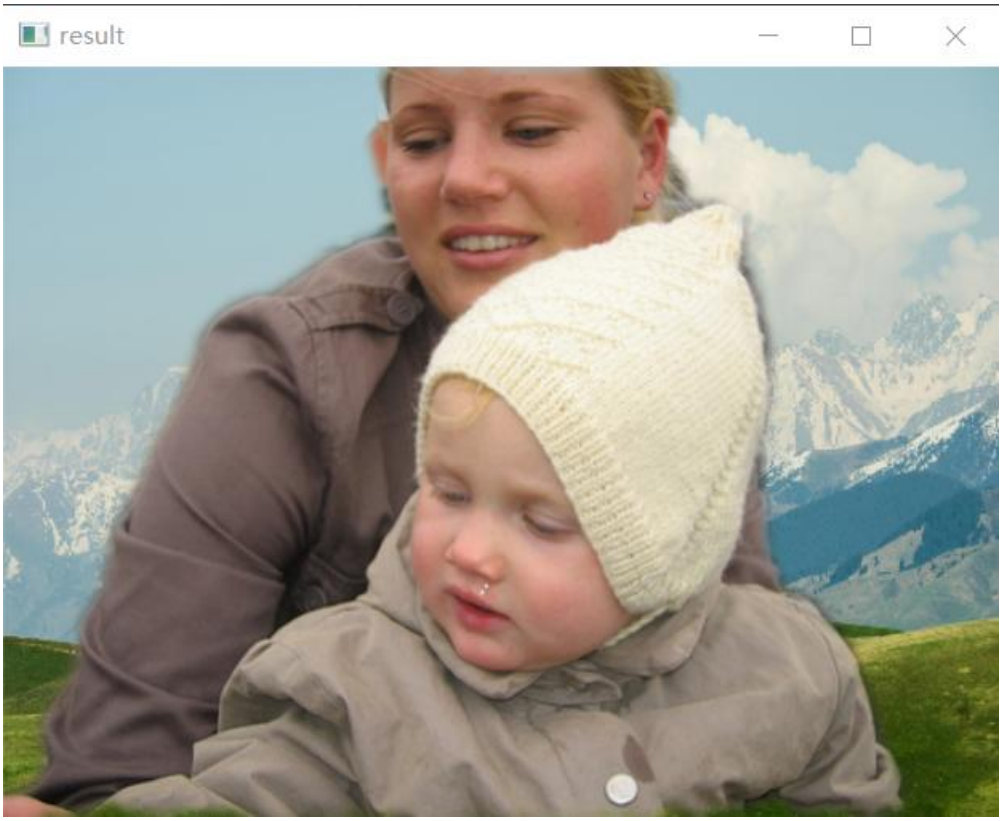
模型测试时，将 mode_path 保存为训练好的权重文件。

8.4.4.1 单张图片测试模式

对单张图片进行前景分割并替换背景，需要修改代码中 mode 参数，令 mode="image"，设置为图片模式。

运行测试代码，按提示输入背景图片路径和待处理人像图片路径后，显示处理后的人像背景替换图片，结果会保存至 image_save_path 指定路径。

```
vediopeople
D:\Users\ASUS\anaconda3\python.exe F:\vediopeople\segformer-pytorch-master\segformer-pytorch-master\vediopeople.py
model_data/segformer_b0_weights_voc.pth model, and classes loaded.
Configurations:
----------------------------------------------------------------------
|                     keys |                                  values|
----------------------------------------------------------------------
|               model_path |  model_data/segformer_b0_weights_voc.pth|
|              num_classes |                                      21|
|                      phi |                                      b0|
|              input_shape |                              [512, 512]|
|                 mix_type |                                       0|
|                     cuda |                                    True|
----------------------------------------------------------------------
Input background filename:img/2.jpg
成功加载背景图片: img/2.jpg
Input image filename:img/6.jpg
图片处理完成，已保存至: result.jpg
```



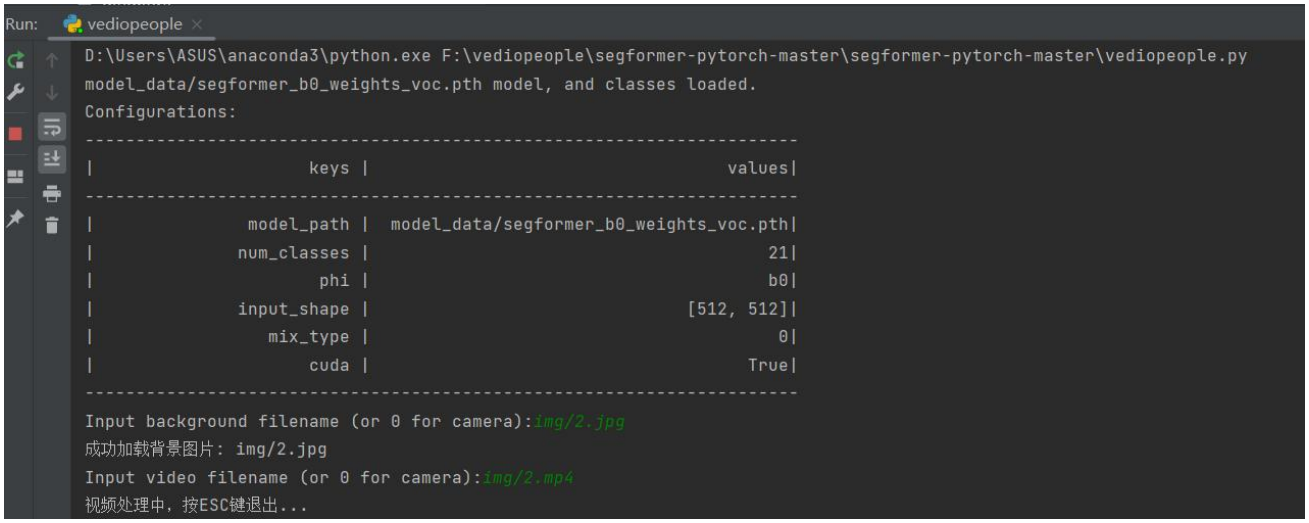

```
if mode == "image":
        # 加载背景图片
        if replace_background:
            try:
                background_image_path = input('Input background filename:')
                background_img = cv2.imread(background_image_path)
                if background_img is None:
                    raise FileNotFoundError(f"无法加载背景图片: {background_image_path}")
                print(f"成功加载背景图片: {background_image_path}")
            except Exception as e:
                print(f"加载背景图片失败: {str(e)}")
                replace_background = False  # 禁用背景替换
```

```
        # 输入图片路径
        image_path = input('Input image filename:')
        try:
            # 读取图片
            frame = cv2.imread(image_path)
            if frame is None:
                raise FileNotFoundError(f"无法加载图片: {image_path}")

            # 格式转换：BGR→RGB（适配 PIL）
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_pil = Image.fromarray(np.uint8(frame_rgb))

            # 模型推理：获取掩码
            _, mask = segformer.detect_image(frame_pil, count=False)
            processed_frame = frame_rgb  # 使用原始图像作为前景

            # 背景替换处理
            if replace_background and background_img is not None:
                # 调整背景图片尺寸以匹配输入图片
                background_img = cv2.resize(background_img, (frame.shape[1], frame.shape[0]))

                # 创建前景掩码
                foreground_mask = np.zeros(mask.shape[:2], dtype=np.uint8)
                for cls in foreground_classes:
                    foreground_mask[mask == cls] = 255

                # 对掩码进行模糊处理，使边缘更自然
                foreground_mask = cv2.GaussianBlur(foreground_mask, (15, 15), 0) / 255.0
                foreground_mask = np.expand_dims(foreground_mask, axis=-1)

                # 格式转换：RGB→BGR（适配 OpenCV）
                processed_frame_bgr = cv2.cvtColor(processed_frame, cv2.COLOR_RGB2BGR)

                # 融合前景和新背景
                result_frame = (processed_frame_bgr * foreground_mask +
                                background_img * (1 - foreground_mask)).astype(np.uint8)
            else:
                # 不替换背景时直接转换格式
                result_frame = cv2.cvtColor(processed_frame, cv2.COLOR_RGB2BGR)

            # 保存结果图片
            cv2.imwrite(image_save_path, result_frame)
            print(f"图片处理完成，已保存至: {image_save_path}")

            # 显示结果图片
            cv2.imshow("result", result_frame)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        except Exception as e:
            print(f'处理图片时出错: {str(e)}')
```

8.4.4.2 视频测试模式

对人像进行前景分割并替换背景，需要修改代码中 mode 参数，令 mode="video"，设置为视频模式。

运行测试代码，按提示输入背景图片路径和待处理人像视频路径后，实时显示处理结果，并把结果保存至 video_save_path 指定路径。



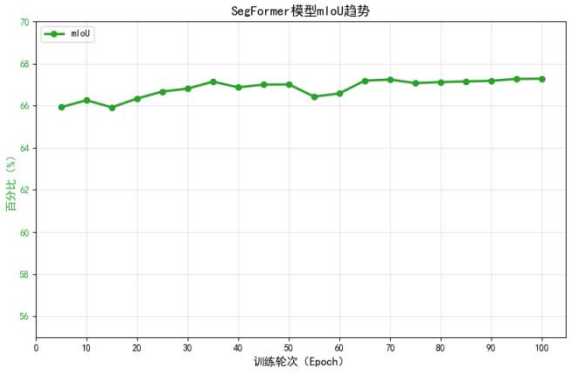

```
elif mode == "video":
    # 原有视频处理逻辑保持不变
    while True:
        # 加载背景图片
        if replace_background:
            try:
                background_image_path = input('Input background filename (or 0 for camera):')
                background_img = cv2.imread(background_image_path)
                if background_img is None:
                    raise FileNotFoundError(f"无法加载背景图片: {background_image_path}")
                print(f"成功加载背景图片: {background_image_path}")
            except Exception as e:
                print(f"加载背景图片失败: {str(e)}")
                replace_background = False  # 禁用背景替换

        # 循环输入视频路径
        video_path = input('Input video filename (or 0 for camera):')
        # 处理摄像头输入（0）
        if video_path.strip() == '0':
            video_path = 0
        try:
            # 尝试打开视频
            capture = cv2.VideoCapture(video_path)
            if not capture.isOpened():
                raise ValueError("无法打开视频文件或摄像头")
        except Exception as e:
            print(f'Open Error! {str(e)} Try again!')
            continue

        # 替换原视频保存部分的编码设置
        if video_save_path != "":
            # 使用 MP4 兼容的编码格式
```

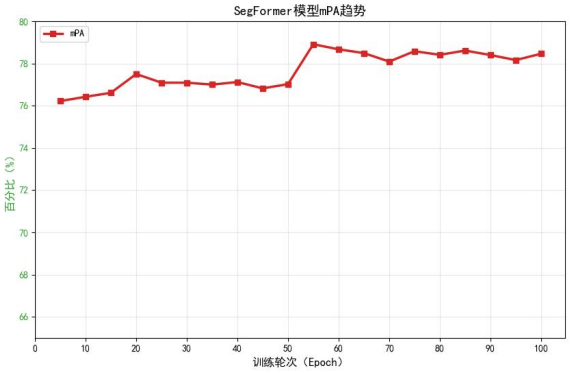
```
                    fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # 替换 'XVID' 为 'mp4v'
                    size = (int(capture.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT)))
                    out = cv2.VideoWriter(video_save_path, fourcc, video_fps, size)

                    # 调整背景图片尺寸以匹配视频
                    if replace_background and background_img is not None:
                        background_img = cv2.resize(background_img, size)

                ref, frame = capture.read()
                if not ref:
                    raise ValueError("未能正确读取摄像头（视频），请注意是否正确安装摄像头（是否正确填写视频路径）。
")

                fps = 0.0
                print("视频处理中，按 ESC 键退出...")
                while (True):
                    t1 = time.time()
                    ref, frame = capture.read()
                    if not ref:
                        break
                    # 格式转换：BGR→RGB（适配 PIL）
                    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    frame_pil = Image.fromarray(np.uint8(frame_rgb))

                    # 模型推理：只需要掩码（mask），不需要带粉色的 processed_frame
                    _, mask = segformer.detect_image(frame_pil, count=False)  # 只取掩码
                    processed_frame = frame_rgb  # 直接用原始 RGB 图像

                    # 背景替换处理
                    if replace_background and background_img is not None:
                        # 创建前景掩码（只保留指定类别的区域）
                        foreground_mask = np.zeros(mask.shape[:2], dtype=np.uint8)
                        for cls in foreground_classes:
                            foreground_mask[mask == cls] = 255

                        # 对掩码进行模糊处理，使边缘更自然
                        foreground_mask = cv2.GaussianBlur(foreground_mask, (15, 15), 0) / 255.0
                        foreground_mask = np.expand_dims(foreground_mask, axis=-1)

                        # 格式转换：RGB→BGR（适配 OpenCV 显示）
                        processed_frame_bgr = cv2.cvtColor(processed_frame, cv2.COLOR_RGB2BGR)

                        # 融合前景和新背景
                        frame = (processed_frame_bgr * foreground_mask +
                                 background_img * (1 - foreground_mask)).astype(np.uint8)
                    else:
                        # 不替换背景时直接转换格式
                        frame = cv2.cvtColor(processed_frame, cv2.COLOR_RGB2BGR)

                    # 计算 FPS
                    fps = (fps + (1. / (time.time() - t1))) / 2
                    print("fps= %.2f" % (fps))
                    frame = cv2.putText(frame, "fps= %.2f" % (fps), (0, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0),
                                        2)
```

```
            # 显示视频
            cv2.imshow("video", frame)
            c = cv2.waitKey(1) & 0xff
            # 保存视频
            if video_save_path != "":
                out.write(frame)
            # 按 ESC 退出
            if c == 27:
                capture.release()
                break

    print("Video Detection Done!")
    capture.release()
    if video_save_path != "":
        print("Save processed video to the path :" + video_save_path)
        out.release()
    cv2.destroyAllWindows()
```

8.4.5 性能评估

SegFormer 模型在训练过程中展现出良好的稳定性与逐步收敛的特性，核心分割指标 mIoU 从初始阶段逐步上升，于 30-40 轮后稳定在 63%-65%，平均像素准确率（mPA）同步上升至 78% 左右且波动极小，适用于视频人像背景替换项目。





8.5 实验总结

本次视频背景替换实验以 SegFormer 语义分割模型为核心，围绕高精度人像分割与自然背景融合展开，完成了从数据集优化、模型训练到多场景测试验证的完整实践流程，有效验证了深度学习方法在视频背景替换任务中的优势与实用价值。

实验先从 VOC2012 数据集筛选含“person”类的样本，保留比例 30.48%，构建专用数据集；再通过自定义数据预处理类完成数据处理与增强，配合“冻结-解冻”训练策略优化模型，最终 SegFormer 模型 mIoU 稳定在 63%-65%、mPA 达 78%。实验成功实现单图与视频两种模式的视频人像背景替换，验证了 SegFormer 语义分割模型在在线会议、直播、短视频后期等场景的适用性。

8.6 资源下载

实验报告下载地址：https://github.com/stroberry5/Segformer_person

数据集下载地址 https://github.com/stroberry5/Segformer_person
实验代码下载地址：https://github.com/stroberry5/Segformer_person
视频演示下载地址：https://github.com/stroberry5/Segformer_person

参考文献

[1] Xie E, Wang W, Yu Z, et al. SegFormer: Simple and efficient design for semantic segmentation with transformers[J]. Advances in neural information processing systems, 2021, 34: 12077-12090.

[2] Chen L C, Papandreou G, Kokkinos I, et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs[J]. IEEE transactions on pattern analysis and machine intelligence, 2017, 40(4): 834-848.

[3] Zheng S, Lu J, Zhao H, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021: 6881-6890.

[4] Cordts M, Omran M, Ramos S, et al. The cityscapes dataset for semantic urban scene understanding[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 3213-3223.

[5] Zhou B, Zhao H, Puig X, et al. Scene parsing through ade20k dataset[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 633-641.