**基于无监督的显著物体分割算法**

陶渝 1221001033

摘要

本文档聚焦计算机视觉领域无监督显著物体分割算法的研究与实践。首先搭建 PyTorch 开发环境，完成一个针对传统方法依赖手工特征、复杂场景表现不足的问题，采用两阶段框架，通过渐进式课程学习的显著性蒸馏和自校正伪标签优化的算法的复现。在此基础上，通过消融实验验证了损失函数组合的有效性，分析了算法不足并展望未来方向。最后，将通道与空间注意力机制结合构建卷积块注意力模块（CBAM），实验表明该改进在多个数据集上均有效。

关键词：无监督学习；显著物体分割；注意力机制

# 一、引言​

计算机视觉作为人工智能领域的核心分支，在当前科技发展中占据重要地位，其在自动驾驶、医疗影像诊断、安防监控等诸多领域的广泛应用，深刻改变着社会发展模式与人们生活方式。这一技术通过模拟人类视觉感知机制，实现对图像和视频的理解与分析，不仅推动了相关产业的智能化升级，也为解决复杂现实问题提供了创新路径，其研究价值与应用前景备受关注。

本次大作业旨在通过对计算机视觉领域中显著物体分割的探究与实践，深入掌握算法原理的理解和并对算法进行复现、对比和改进，解决实际应用问题并探索技术边界。为系统呈现研究内容，文档后续将依次展开环境搭建、显著目标检测问题定义与研究现状分析、算法原理介绍、实验设计与结果讨论等内容，通过系统化学习相关领域为学生学习计算机视觉与实践提供完整的知识体系。

# 二、PyTorch 与环境管理

深度学习项目的高效开发依赖于稳定且可复现的运行环境，PyTorch 作为主流框架之一，其环境搭建与管理是开展计算机视觉等任务的基础。本章系统阐述 PyTorch 开发环境的全流程构建体系，从硬件驱动配置、框架安装到虚拟环境隔离，再到包依赖管理与代码版本控制。通过分析 Conda 与 pip 的包管理策略、虚拟环境的实践应用以及 Git 代码托管流程，梳理学生在学习过程中的知识和经验。

## 2.1 PyTorch 环境搭建

### 2.1.1 安装前的准备工作

在搭建 PyTorch 开发环境前，需完成硬件驱动与基础工具的配置。

首先，通过命令行输入nvidia-smi.exe查询显卡支持的 CUDA 版本，以此为依据安装对应的 CUDA 工具包及 CuDNN 加速库，确保 GPU 能够正常被使用。

其次，需安装 VSCode 等集成开发环境，推荐配置 Chinese 语言包、Python 插件及 Pylance 代码分析工具，以提升开发效率。

此外，需从官网下载 Conda 环境管理工具，完成安装后配置系统环境变量，并将 Conda 终端集成至 VSCode，为后续虚拟环境创建奠定基础。

利用 Conda 工具创建独立的虚拟开发环境，建议选择不低于 Python 3.10 的版本，以确保对 PyTorch 最新特性的支持。利用Conda工具创建的虚拟环境可实现项目依赖的隔离，避免不同项目间的版本冲突，提升环境的稳定性与可复现性。

### 2.1.2 PyTorch 安装方法

Anaconda 与 pip 是两种主流的安装方式。

Anaconda 安装流程需先创建新环境并指定 Python 版本（一般直接从官网复制相关命令），通过conda install命令添加 PyTorch，可按需配置 GPU 支持；该方式适合复杂依赖管理，能自动解决包依赖冲突。

pip 安装则需确保已安装 Python 和 pip，直接运行pip install torch torchvision torchaudio命令，可通过添加 CUDA 版本参数（如+cu118）实现 GPU 加速支持（一般也直接从官网复制命令），其优势在于灵活性高，适合已有环境的扩展。

安装完成后，需验证环境的正确性：通过torch.\_\_version\_\_命令查询 PyTorch 版本；在 Python 环境中尝试导入torch库，确保无报错；运行print(torch.cuda.is\_available())检查 GPU 是否可用。若出现依赖冲突等问题，可通过更新包版本或重新配置环境解决，必要时参考官方文档、社区资源或询问大预言模型。

## 2.2 包管理工具

### 2.2.1 Conda 与 pip 的功能对比

Conda 作为 Anaconda 的核心组件，具备多语言包管理能力，可自动处理依赖冲突，适合科学计算领域的复杂项目；其内置的虚拟环境管理功能可实现项目隔离。

pip 则专为 Python 设计，聚焦 Python 生态，资源更新迅速，处理 Python 包效率更高，但需配合virtualenv等工具实现环境隔离，依赖冲突需手动解决。

### 2.2.2 requirements.txt 依赖管理

requirements.txt文件用于记录项目所需的所有 Python 包及其版本，确保环境一致性。通过pip freeze > requirements.txt命令可自动导出当前环境依赖；手动编辑可更新包版本或添加新依赖；部署时执行pip install -r requirements.txt即可快速重建环境，提升团队协作与项目迁移效率。

### 2.2.3 高级功能：自定义包构建与发布

构建自定义包时，需设计包含\_\_init\_\_.py的清晰目录结构，通过setup.py文件定义包元数据（如名称、版本、依赖项）。完成开发后，在虚拟环境中测试安装，确保功能正常，最终通过twine工具上传至 PyPI（Python Package Index），实现包的公开共享。

## 2.3 虚拟环境：隔离项目依赖

### 2.3.1 为什么需要虚拟环境

虚拟环境通过隔离项目依赖，可有效避免不同项目间的版本冲突，确保本地开发与生产环境的一致性。其简化了部署流程，通过精确复制环境配置，新成员可快速接入项目，同时提升了团队协作效率，降低了环境配置的沟通成本。

### 2.3.2 虚拟环境基础操作

创建虚拟环境可使用 Python 内置的venv模块或 Conda 工具；激活环境时，Windows 系统执行venv\Scripts\activate，Linux/Mac 系统执行source venv/bin/activate；项目完成后，直接删除虚拟环境目录即可释放资源，更多的命令可以查看下表。

表 1 包管理命令参考表

|  |  |  |
| --- | --- | --- |
| **操作场景** | **venv** | **Conda** |
| 创建虚拟环境 | python -m venv venv\_name | conda create -n env\_name python=X.X |
| 激活环境 | Windows: venv\Scripts\activate Linux/Mac: source venv/bin/activate | Windows: conda activate env\_name  Linux/Mac: conda activate env\_name |
| 查看已激活环境 | echo %VIRTUAL\_ENV%（Windows） echo $VIRTUAL\_ENV（Linux/Mac） | conda info --envs |
| 安装包 | pip install package\_name | conda install package\_name pip install package\_name |
| 导出依赖 | pip freeze > requirements.txt | conda list --explicit > environment.txt |
| 删除环境 | 直接删除环境目录（如rm -rf venv\_name） | conda remove -n env\_name --all |
| 更新环境 | pip install --upgrade package\_name | conda update package\_name conda update --all |
| 克隆环境 | 不直接支持，需通过 requirements.txt 重建 | conda create -n new\_env --clone old\_env |

## 2.4 代码仓库的使用

### 2.4.1 GitHub 托管流程

以 GitHub 为例，代码托管需先注册账户并创建新项目，通过git clone <repository-url>命令将远程仓库克隆至本地。本地修改代码后，使用git add和git commit提交更改，再通过git push同步至远程仓库。为管理不同开发功能，可创建分支并通过权限设置实现团队协作。

### 2.4.2 Git 基础操作

分支管理是 Git 的核心功能之一，通过git branch创建新分支，git checkout切换分支，确保主分支的稳定性。版本控制流程需遵循 “克隆 - 修改 - 提交 - 推送” 的标准工作流，确保代码变更可追溯，团队协作高效有序。更多相关命令可以参考下表

表 2 git命令参考表

|  |  |
| --- | --- |
| **操作场景** | **Git 命令** |
| 创建新分支 | git branch <branch-name> |
| 切换分支 | git checkout <branch-name> |
| 创建并切换分支 | git checkout -b <branch-name> |
| 删除分支 | git branch -d <branch-name> |
| 重命名分支 | git branch -m <old-name> <new-name> |
| 查看所有分支 | git branch -a |
| 拉取远程分支到本地 | git checkout -b <local-name> origin/<remote-name> |
| 推送本地分支到远程 | git push -u origin <branch-name> |
| 删除远程分支 | git push origin --delete <branch-name> |
| 跟踪远程分支 | git branch --set-upstream-to=origin/<remote-name> <local-name> |
| 拉取远程分支到本地 | git checkout -b <local-name> origin/<remote-name> |
| 推送本地分支到远程 | git push -u origin <branch-name> |
| 删除远程分支 | git push origin --delete <branch-name> |
| 跟踪远程分支 | git branch --set-upstream-to=origin/<remote-name> <local-name> |
| 克隆远程仓库 | git clone <repository-url> |
| 查看工作区状态 | git status |
| 添加文件到暂存区 | git add <file> 或 git add . |
| 提交到本地仓库 | git commit -m "提交说明" |
| 推送到远程仓库 | git push origin <branch-name> |
| 从远程拉取并合并 | git pull origin <branch-name> |

## 2.5 小结

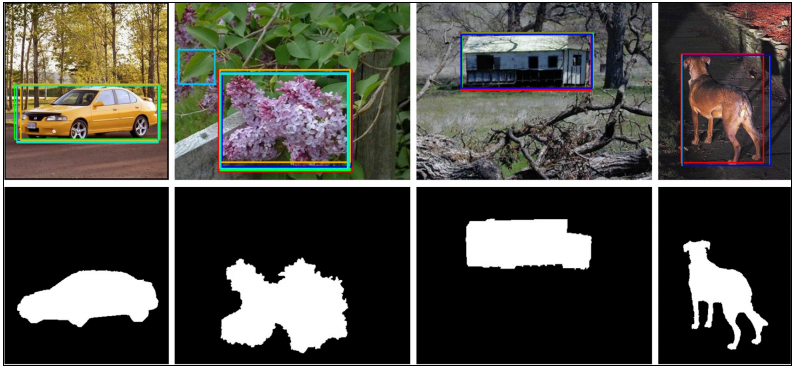
本章围绕 PyTorch 开发环境的核心要素，构建了 “环境准备 - 框架部署 - 依赖管理 - 代码托管” 的闭环知识体系。合理配置 CUDA 与驱动、选择适配的安装方式（Anaconda 或 pip）是构建下一步项目的前提；虚拟环境通过隔离项目依赖，有效解决了多项目并行开发的版本冲突问题；requirements.txt 与自定义包发布机制则强化了环境的可复现性与团队协作效率；而 Git 与 GitHub 的版本控制流程，为代码的迭代与协作提供了标准化方案。整体而言，本章内容既覆盖了开发环境搭建的基础操作，又渗透了工程化实践的核心思想，为后续计算机视觉算法的研究与应用奠定了坚实的工程基础。

# 三、问题定义和研究现状​

显著目标检测作为计算机视觉的基础任务，其核心在于解析人类视觉注意力机制并实现图像显著区域的自动化识别。本章系统阐述显著目标检测的问题本质与研究现状，首先明确任务的输入输出定义及核心目标，继而从传统方法演进至深度学习框架下全监督、弱监督与自监督模型的现状。

## 3.1 问题定义

显著目标检测（Salient Object Detection, SOD）旨在识别图像中吸引人类注意力的关键区域，**通过模拟人类视觉系统对场景中醒目部分的感知机制，提取图像中的突出区域。**这一任务的核心目标是为后续高级视觉任务（如图像分割、目标识别等）提供预处理支持，通过定位显著性区域提升任务效率与资源管理能力。输入为原始图像，输出为标注显著区域的二值掩码或概率图，其本质是对图像中语义重要性的像素级预测。



## 3.2 研究现状

### 3.2.1 传统方法演进

早期显著目标检测研究基于低级视觉特征构建模型：基于局部对比度的方法通过像素级中心定位，利用颜色、方向等特征计算元素与周围环境的对比度差异；基于扩散的模型借助图结构扩散矩阵，在图像区域内传播显著性值；基于贝叶斯方法的模型则通过兴趣点凸包估计显著性先验，结合聚类技术生成边界簇。此外，基于目标先验和经典机器学习算法的模型也在早期研究中得到应用，这些方法依赖手工设计特征，在复杂场景下泛化能力有限。

### 3.2.2 深度学习方法发展

全监督 SOD 模型以像素级标注数据为基础，通过深度神经网络提取多尺度特征。Zhao 等人（2015）采用双 CNN 架构分别建模超像素的全局与局部上下文，通过多层感知机回归生成显著性分数；He 等人（2017）将子化任务作为辅助训练目标，基于 U-Net 架构引入自适应权重层实现特征融合；Islam 等人（2018）通过跳跃连接策略促进编码器特征的渐进细化，结合子化任务监督提升边界检测精度。此类模型依赖大规模标注数据，在受控场景下表现优异，但数据采集成本较高。

弱监督模型通过稀疏标注（如边框、图像级标签）降低数据依赖。Wang 等人（2017）联合训练分类网络与前景特征推理网络（FIN），通过迭代条件随机场（CRF）自训练细化预测；Zhu 等人（2018）在 GAN 框架的判别器中引入相关层，强化显著性图与真实掩码的局部块对比，提升边界捕捉能力。该类方法在减少标注工作量的同时，仍需平衡监督信号稀疏性与预测精度的矛盾。

自监督学习为 SOD 提供了无标注数据下的特征学习范式。Jidong 等人（2025）将对比学习引入 SOD 框架，利用 Token to Token Vision Transformer（T2T）和 Vision Graph Neural Network（ViG）实现图像级与像素级对比学习，其方法可作为插件集成至各类模型；Lina 等人（2025）设计多模态自监督预训练模型，通过外观表示学习、深度增强自监督及对比多模态表示辅助任务，构建 RGB-D SOD 模型，在八个数据集上实现自监督性能突破，将注释工作量减少 53% 以上。

### 3.2.3 现存问题与挑战

当前 SOD 领域面临诸多技术挑战：

尽管预训练 CNN 具备多尺度特征提取能力，模型在特征聚合与显著性区域精确表征方面仍存在瓶颈；

大规模无偏数据集的缺失影响模型对复杂场景的泛化能力，尤其在物体与背景外观相似或场景杂乱时检测性能显著下降；

此外，针对移动与嵌入式设备的轻量级模型设计，尚未形成成熟的解决方案，限制了 SOD 技术的实际应用拓展。

## 3.3小结

本章围绕显著目标检测的问题定义与研究现状进行分析，明确了该任务在视觉感知与高级任务预处理中的关键作用。研究现状表明，传统方法基于低级特征构建显著性模型，而深度学习通过监督范式创新（全监督、弱监督、自监督）推动了性能突破，尤其 2025 年自监督对比学习与多模态融合技术的引入，为无标注数据场景下的模型训练提供了新范式。然而，领域仍面临特征高效聚合、大规模无偏数据集构建、复杂场景鲁棒性及轻量级部署等挑战。

# 四、算法介绍​

在本章节中，学生主要介绍一种无监督显著物体分割算法的结构、原理和功能，并且在原有算法的基础上尝试做出部分改进。相关的论文链接为<https://arxiv.org/html/2404.14759>

## 4.1算法背景

传统的显著性目标检测（SOD）方法严重依赖手工设计的特征（如颜色和对比度）来提取显著性。尽管这些方法在简单场景中表现有效，但由于缺乏高级语义信息，在复杂场景中会面临挑战。现有的基于深度学习的无监督 SOD 方法大多将传统 SOD 方法生成的预测结果作为显著性线索，并结合语义信息来生成更精细的显著性预测。最近，基于在大规模数据上预训练的卷积神经网络（CNN）通常会在某些主要对象上产生高激活值的观察，A2S 方法开发了一种从深度网络的激活图中提取显著性并生成高质量伪标签的方法。然而，我们发现在初始训练阶段，复杂场景或物体边界处的难例样本会导致不可修复的误差积累。

## 4.2 算法框架

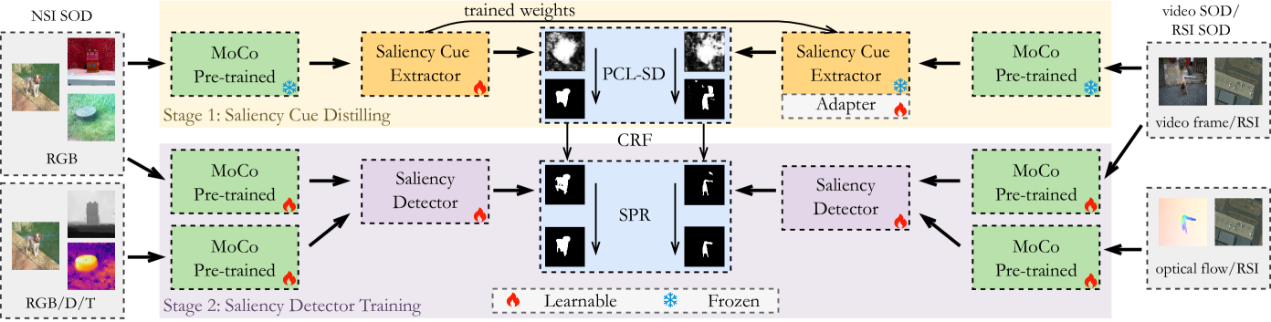


图 1 算法框架

学生复现算法框架如上，该算法分为两阶段，在阶段一，论文训练一个显著性线索提取器（SCE），以从预训练的深度网络中迁移显著性知识。论文采用提出的基于渐进式课程学习的显著性蒸馏方法，来缓解训练初期的误差积累问题，并确保训练过程的稳定性和鲁棒性。在阶段二，论文将获得的显著性线索作为初始伪标签，用于训练显著性检测器（SD）。论文采用 CR来增强初始伪标签，并使用提出的自修正伪标签优化机制，在训练过程中逐步提升伪标签的质量。

原算法同样对RGBD SOD、视频 SOD 和 RSI SOD进行了优化，考虑到引入这三类数据将会显著增加该模型的训练成本，且本文档用于学生学习并非研究，所以本文档只讨论算法对RGB数据的分割。

### 4.2.1 显著性蒸馏过程

显著性蒸馏的过程可以表示为：

其中，表示像素数量，表示由显著性线索提取器（SCE）输出的显著性预测*S*中的第*i*个像素。直观来说，将每个像素的预测值向 0 或 1 两个极端拉动。然而在训练初期，可能会将预测值接近0.5的难例样本向错误方向拉动，论文将此称为误差累积问题。所提出的 PCL-SD 策略聚焦于两个核心要点：（1）如何定义难例样本；（2）如何逐步引入难例样本。首先，显著性预测*S*中某个像素是否属于难例样本，取决于其预测值。具体而言，当像素*S*(*i*)满足以下条件时，会被归类为难例样本：

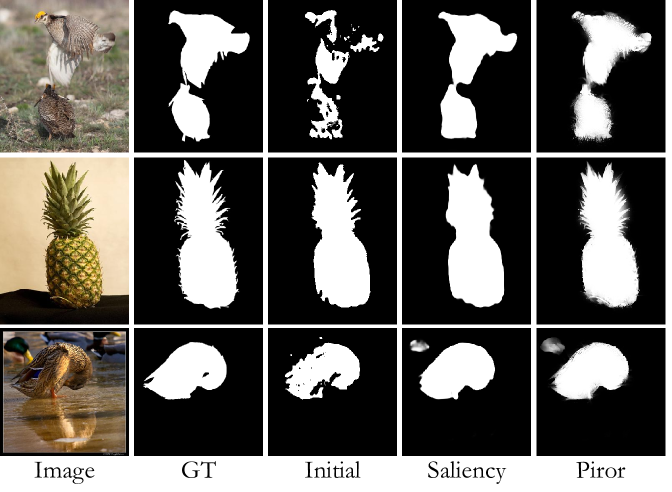
其中，p是划分难例样本的阈值，p越大意味着划分出的难例样本越多。其次，p的初始值设为 0.2，并在训练过程中逐渐减小，直至包含所有样本。这种递减过程由以下公式控制：

其中 Ec 和 Et 分别表示当前周期和总周期。最后，我们定义 PCL-SD 为：

其中 ⊙ 表示矩阵的 Hadamard 乘积。

### 4.2.2 自校正伪标签优化

获取高质量的伪标签对于训练显著性检测器（SD）至关重要。另一方面，如图所示，SD 输出的显著性预测结果 *S* 能够部分修正伪标签中的误差。我们将显著性预测定义为后验修正：。然而，尽管这种后验修正可以纠正初始伪标签中的错误，但也会引入模型过度自信并陷入停滞的风险。为了克服这一问题，我们引入输入图像的先验信息来优化显著性预测，以避免模型陷入自我满足的陷阱。



以往的方法主要依赖条件随机场（CRF）进行先验修正，这会带来巨大的计算成本。论文采用了一种实时像素细化器，基于输入图像提供高效的先验修正。首先，我们将像素之间的特征距离 和位置距离 ​ 定义如下：

其中，I和P分别表示输入图像和位置信息，而​和​分别表示特征值和位置差异的标准差。参数​和​控制平滑程度。然后，细化器R(⋅)定义为：

其中， 𝒩(⋅) 表示以8方向方式相邻的像素集合。最后，先验校正可以定义为：

其中，⊙ 表示矩阵的哈达玛积（逐元素相乘）。最终，经细化的伪标签定义为：

其中，指细化后的伪标签， 指之前的伪标签。引入​ 旨在提升细化过程的稳定性。、、 经经验赋值，分别为 0.2、0.6 和 0.2 。如图 4 所示，先验修正有效弥补了局部细节的大量丢失问题。论文所提出的 SPR 机制结合了先验修正与后验修正，在训练过程中逐步提升伪标签质量，展现出强大的自监督性能。

### 4.2.3 监督策略

论文首先在第一阶段训练显著性线索提取器（SCE），随后在第二阶段训练显著性检测器（SD）。在第一阶段的训练中，论文引入了边界感知纹理匹配（BTM）来补充额外的结构线索，其形式化为：

这里，​ 表示显著性纹理向量，表示输入图像的纹理向量，而  表示显著性预测的二值边界掩码。此外，采用结构一致性损失来实现变换不变性预测，其形式化为：

其中,表示经过变换后的显著性预测结果。为确保训练稳定性，仅采用随机缩放变换。训练 SCE 的总损失可定义为：

*γ* 经经验赋值为 0.05。论文使用 IoU 损失来训练显著性检测器（SD），其定义为：

其中 *G* 指伪标签，训练 SD 的总损失可定义为：

## 4.3 算法改进

学生在理解上述算法基本原理的基础上对上述算法进行了一些试探性的改进。具体而言，学生在a2s算法框架中，将通道注意力机制和空间注意力机制相结合，构建卷积块注意力模块。

在第二阶段的训练过程中，原算法框架分别在特征转换模块、全局信息模块和全局特征融合模块中均集成了通道注意力机制（CA），其形式化为：

其中，是网络中间层的特征图，为全局平均池化，为全局最大池化，表示多层感知机，是 Sigmoid 函数。

在通道注意力的基础上，引入了空间注意力机制，通过通道注意力和空间注意力的结合，构建卷积块注意力模块（CBAM）。其中，空间注意力机制的形式化为：

其中，*F* 是网络中间层的特征图为通道维度拼接，是用 7×7 卷积对拼接后的特征做空间卷积。

卷积块注意力模块的形式化定义为：

其中，F表示输入特征图，⊗表示点乘运算。卷积块注意力模块通过顺序应用通道注意力和空间注意力模块，对中间特征图进行自适应细化。

## 4.4 小结​

本章主要介绍了一种无监督显著物体分割算法，该算法分为显著性线索提取器训练和显著性检测器训练两个阶段，通过渐进式课程学习的显著性蒸馏方法和自修正伪标签优化机制来提升算法性能。此外学生在原有算法基础上尝试对注意力机制进行改进，尝试为提高这种无监督显著目标检测的性能指标。

# 五、实验结果​

## 5.1实验设置

批处理大小设置为 8，输入图像被调整为 320×320。我们采用水平翻转作为数据增强方法。使用 SGD 优化器，初始学习率为 0.1，该学习率线性衰减，我们训练显著性线索提取器 20 个 epoch。使用学习率为 0.005 的 SGD 优化器，我们训练显著性检测器 10 个 epoch。所有实验均在单个 RTX 3090 GPU （Autodl）上实现。

我们使用 DUTS 数据集的训练子集来训练，评估时采用了 ECSSD、PASCALS、HKU-IS、DUTS-TE和 DUT-O数据集，这种数据集的选择和原论文相同。

我们采用三种指标来评估我们的模型和现有最先进的方法，包括平均绝对误差（M）、平均 F-measure（Fβ和 E-measure（Eξ）。具体的计算公式如下：

平均绝对误差（M）衡量预测图 P 与真实标签 G 之间的逐像素平均差异，计算公式为：

指标同时考虑预测图的精确率（Precision）和召回率（Recall），计算公式为：

其中 β² 设为 0.3。

Eξ 指标综合考虑局部像素值与图像级均值，定义为：

其中表示增强对齐矩阵。

## 5.2 实验结果

学生将复现的算法和原论文中的效果进行展示，具体如下：

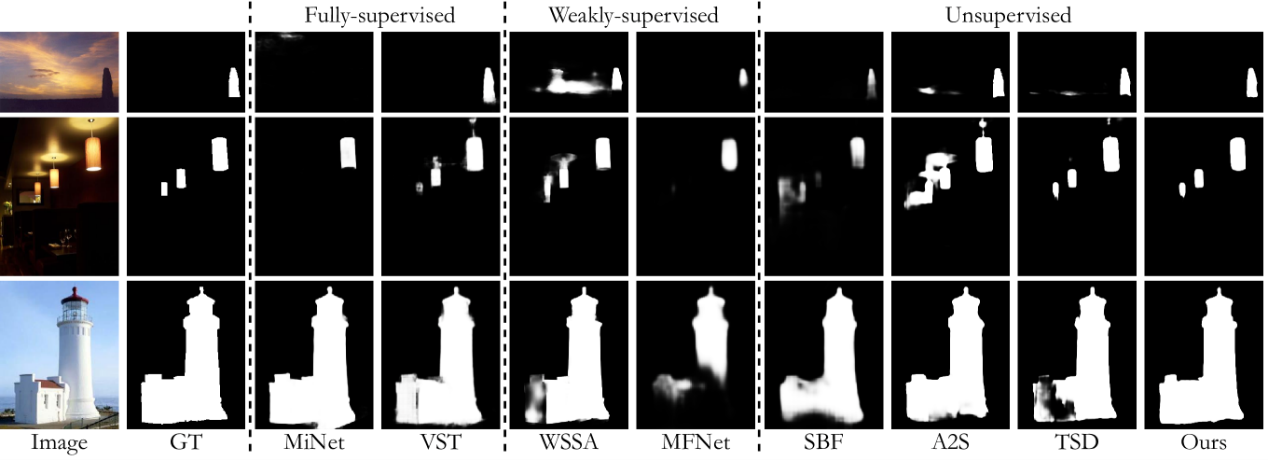


图 2 原论文中对比效果



图 3 原图

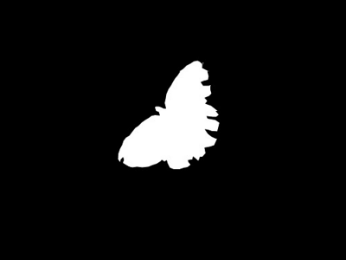
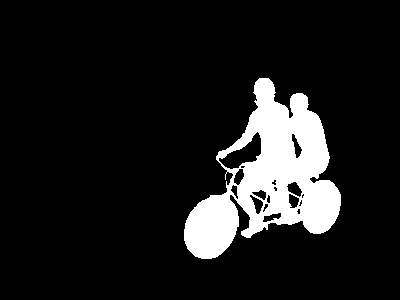
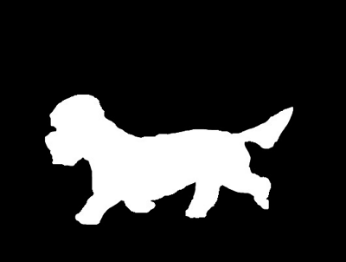


图 4 真值图像

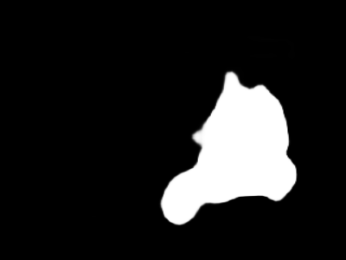


图 5 第一阶段产生的伪标签

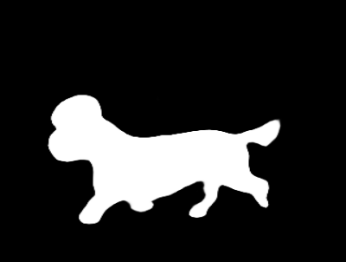


图 6 第二阶段残生的真值效果

学生基于 DUTS 训练集复现算法后，在 DUT-O、DUTS-TE、ECSSD、HKU-IS 和 PASCAL-S 五个测试集上进行评估。从表中数据可见

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Method | DUT-O | | | DUTS-TE | | | ECSSD | | | HKU-IS | | | PASCAL-S | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| paper | .062 | .759 | .868 | .047 | .816 | .906 | .038 | .923 | .951 | .033 | .908 | .954 | .069 | .844 | .899 |
| ours | .069 | .724 | .841 | .053 | .791 | .885 | .044 | .916 | .940 | .039 | .897 | .943 | .074 | .832 | .887 |

Paper表示原论文中的指标，ours表示学生复现的指标情况，和原论文具体指标相比，复现的算法指标浮动均在合理范围内。

学生进行了消融实验，具体而言，我们采取不同监督策略并进行对比，结果如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Method | DUT-O | | | DUTS-TE | | | ECSSD | | | HKU-IS | | | PASCAL-S | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| iou+bce+mse | .069 | .724 | .841 | .053 | .791 | .885 | .044 | .916 | .940 | .039 | .897 | .943 | .074 | .832 | .887 |
| iou | .071 | .724 | .839 | .060 | .789 | .871 | .045 | .912 | .932 | .039 | .892 | .944 | .076 | .826 | .881 |
| mse | .502 | .000 | .250 | .502 | .000 | .250 | .502 | .000 | .250 | .502 | .000 | .250 | .502 | .000 | .250 |
| iou+bce | .075 | .724 | .822 | .058 | .787 | .871 | .044 | .916 | .940 | .039 | .898 | .944 | .081 | .831 | .881 |
| bce+mse | .487 | .000 | .25 | .487 | .000 | .250 | .490 | .000 | .250 | .488 | .000 | .250 | .490 | .000 | .250 |

其中，采取iou+bce+mse的联合损失达到的综合效果最优，该结论与原论文一致。

最后学生将原算法的通道注意力（CA）替换为卷积块注意力模块（CBAM）后，关键指标显著改善，最终结果如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Method | DUT-O | | | DUTS-TE | | | ECSSD | | | HKU-IS | | | PASCAL-S | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CA | .069 | .724 | .841 | .053 | .791 | .885 | .044 | .916 | .940 | .039 | .897 | .943 | .074 | .832 | .887 |
| CBAM | .061 | .749 | .861 | .048 | .812 | .899 | .044 | .917 | .940 | .038 | .902 | .944 | .071 | .838 | .891 |

学生使用CBAM模块替换CA模块后，在以上四个数据集中，M（平均绝对误差），Fβ（平均 F - measure ）和Eξ（E - measure ）均有提升，提升幅度分别为5.73%，1.39%和0.87%。

# 八、结论​

本研究作为计算机视觉领域的实践练习，全面展现了学生在算法复现、原理探究及创新设计中的知识积累与实践能力。在工程实践层面，学生通过搭建 PyTorch 开发环境、管理虚拟依赖及运用 Git 进行版本控制，系统掌握了深度学习项目的全流程工程化能力，尤其是对 Conda 与 pip 包管理策略的对比应用，以及 requirements.txt 文件的依赖导出与重建机制，有效解决了多项目开发中的环境冲突问题。在算法原理理解上，学生深入剖析了无监督显著目标分割的两阶段框架，明确了渐进式课程学习（PCL-SD）对训练初期误差累积的缓解机制，以及自修正伪标签优化（SPR）中先验修正与后验修正的协同作用，通过数学公式推导与可视化分析，验证了边界感知纹理匹配（BTM）和结构一致性损失（L\_sc）对特征聚合的强化效果。

在创新实践方面，学生将通道注意力与空间注意力融合为卷积块注意力模块（CBAM），通过实验证明该模块使4个测试数据集的平均绝对误差降低 5.73%、Fβ-measure 提升 1.39%，且实现了结构一致性的显著优化。消融实验中，学生通过对比 IoU+BCE+MSE 联合损失与单一损失的性能差异，验证了多损失函数协同优化的必要性，进一步巩固了对监督策略设计的认知。

然而，实践过程中仍暴露出若干不足：学生未引入 RGBD 等多模态数据，限制了模型在三维场景中的适应性；CBAM 模块的计算开销未进行轻量化优化，影响了算法在边缘设备的部署潜力。后续可从三方面改进：一是结合预训练语言模型增强语义理解，通过跨模态特征融合提升复杂场景鲁棒性；二是探索动态难例挖掘策略，优化伪标签生成过程中的边界定位精度；三是引入轻量级注意力结构（如 MobileNetV3 中的注意力模块），在保持性能的同时降低计算成本，推动算法向实际应用场景拓展。