UniverSeg: Universal Medical Image Segmentation

URL

接收会议：**CVPR2023**(计算机视觉顶级会议)

单位：MIT(Adrian Dalca等大佬), 康奈尔大学

主页：https://universeg.csail.mit.edu/

代码：https://github.com/JJGO/UniverSeg

demo：https://colab.research.google.com

论文下载链接：https://arxiv.org/abs/2304.0613

Introduction

> 理解和阐述问题的重要性（20分）：读者能够理解文献中所讨论的问题的重要性，并能够清晰地解释为什么这个问题值得研究。

虽然深度学习模型已经成为医学图像分割的主要方法，但它们通常无法泛化到涉及新解剖结构、图像模态或标签不可见的分割任务。给定一个新的分割任务，研究人员通常必须训练或微调模型，这很耗时，并对临床研究人员构成了巨大的障碍，因为他们往往缺乏训练神经网络的资源和专业知识。

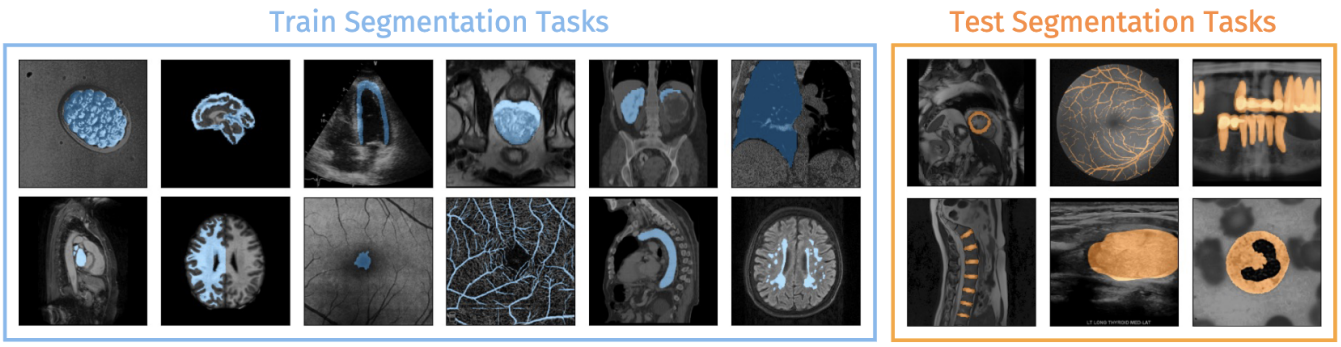


Figure 1: Medical segmentation involves many imaging types, biomedical domains, and target labels. We employ a large diverse set of training tasks **(blue)** to build a model that can segment unseen tasks **(orange)** without additional training.

文章提出了UniverSeg，这是一种在没有额外训练的情况下解决标签不可见的医学分割任务的方法。给定定义新分割任务的查询图像和图像标签对的示例集，UniverSeg采用新的交叉块机制来生成准确的分割图，而不需要额外的训练。为了实现对新任务的泛化，收集并标准化了53个开放访问的医学分割数据集，这些数据集具有超过22000次扫描，我们称之为MegaMedical。我们使用这个集合对UniverSeg进行了各种解剖学和成像模式的训练。我们证明了UniverSeg在标签不可见的任务上显著优于几种相关方法，并对所提出的系统的重要方面进行了彻底的分析和见解。

这篇文章本质上就是医学影像领域的**少样本学习**。

Method

> 方法的理解（20分）：读者能够理解并准确解释文献中使用的技术或方法，包括其原理、优点和缺点。

原理

模型部分

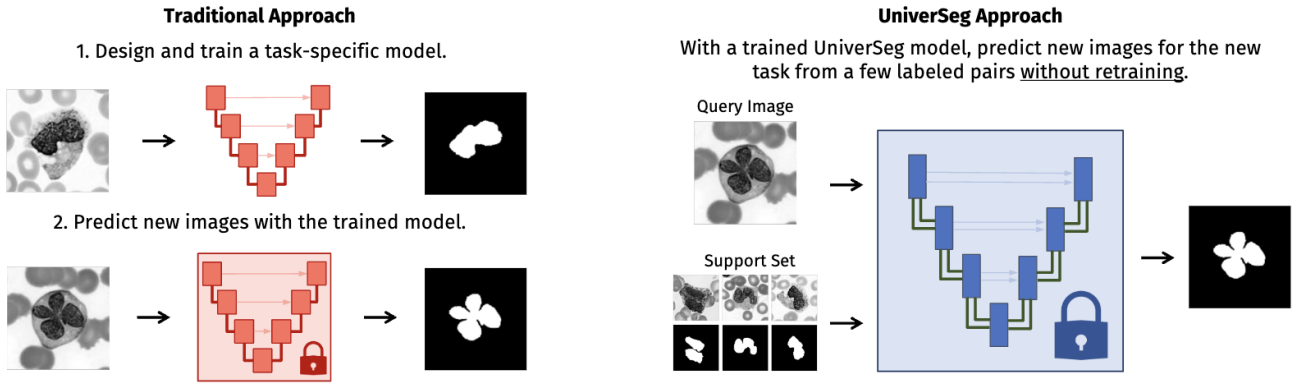


Figure 2: **Workflow for inference on a new task, from an unseen dataset.** Given a new task, traditional models **(left)** are trained before making predictions. UniverSeg **(right)** employs a *single* trained model which can make predictions for images (queries) from the new task with a few labeled examples as input (support set), without additional fine-tuning.

用于从未见过的数据集推断新任务的工作流。给定一个新的任务，传统模型(左)在做出预测之前要经过训练。UniverSeg (右)使用一个经过训练的模型，该模型可以预测来自新任务的图像(查询)，只有几个带标签的示例作为输入(支持集)，而**不需要进行额外的微调**。

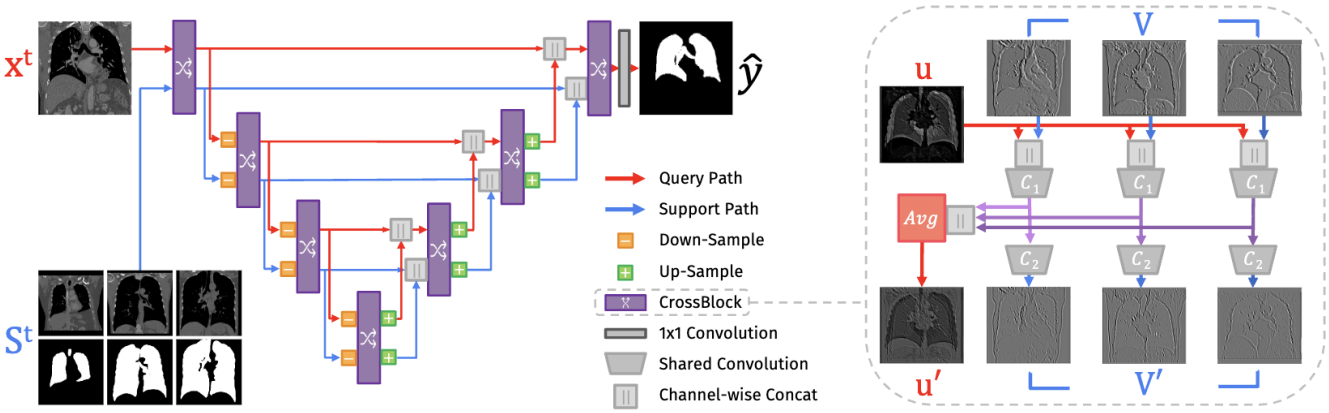


Figure 3: A UniverSeg network **(left)** takes as input a query image and a support set of image and label-maps (pairwise concatenated in the channel dimension) and employs multi-scale CrossBlock features. A CrossBlock **(right)** takes as input representations of the query u and support set $V = \{v_i\}$, and interacts u with each support entry v_i to produce u' and V'.

UniverSeg 网络(左)以一个查询图像和一组图像和标签映射(在通道维度中concat)作为输入，并使用多尺度 CrossBlock 特性。CrossBlock (右)将查询 u 和支持集 V = { vi }作为输入表示，并将 u 与每个支持条目 vi 交互以生成 u ′和 V ′

$$\begin{aligned} \text{CrossConv}(u, V; \theta_z) &= \{z_i\}_{i=1}^{n}, \\ \text{for } z_i &= \text{Conv}(u || v_i; \theta_z), \end{aligned} \tag{1}$$

交叉卷积，就是一个u和每个v进行一个通道维度的拼接，再过一个普通的2d卷积。

$$\text{CrossBlock}(u, V; \theta_z, \theta_v) = (u', V'), \text{where:} \quad (2)$$
$$z_i = A(\text{CrossConv}(u, v_i; \theta_z)) \quad \text{for } i = 1, 2, \ldots, n$$
$$u' = 1/n \sum_{i=1}^{n} z_i$$
$$v'_i = A(\text{Conv}(z_i; \theta_v)) \quad \text{for } i = 1, 2, \ldots, n,$$

卷积之后的输出n个z_i，过一个激活函数，取平均作为新的target。过一个卷积+激活，作为新的support.

下面是核心代码区域：

模型的前向过程 Python 复制代码

```python
def forward(self, target_image, support_images, support_labels):

    target = E.rearrange(target_image, "B 1 H W -> B 1 1 H W")
    support = torch.cat([support_images, support_labels], dim=2)

    pass_through = []

    for i, encoder_block in enumerate(self.enc_blocks):
        target, support = encoder_block(target, support) # 需要查看如果encoder
        if i == len(self.encoder_blocks) - 1:
            break
        pass_through.append((target, support))
        target = vmap(self.downsample, target)
        support = vmap(self.downsample, support)

    for decoder_block in self.dec_blocks:
        target_skip, support_skip = pass_through.pop()
        target = torch.cat([vmap(self.upsample, target), target_skip], dim=2)
        support = torch.cat([vmap(self.upsample, support), support_skip], dim=2)
        target, support = decoder_block(target, support)

    target = E.rearrange(target, "B 1 C H W -> B C H W")
    target = self.out_conv(target)

    return target
```

交叉模块 Python 复制代码

```python
@validate_arguments_init
@dataclass(eq=False, repr=False)
class CrossBlock(nn.Module):

    in_channels: size2t
    cross_features: int
    conv_features: Optional[int] = None
    cross_kws: Optional[Dict[str, Any]] = None
    conv_kws: Optional[Dict[str, Any]] = None

    def __post_init__(self):
        super().__init__()

        conv_features = self.conv_features or self.cross_features
        cross_kws = self.cross_kws or {}
        conv_kws = self.conv_kws or {}

        self.cross = CrossOp(self.in_channels, self.cross_features, **cross_kws)
        self.target = Vmap(ConvOp(self.cross_features, conv_features, **conv_kws))
        self.support = Vmap(ConvOp(self.cross_features, conv_features, **conv_kws))

    def forward(self, target, support):
        target, support = self.cross(target, support) # 需要查看交叉算子
        target = self.target(target)
        support = self.support(support)
        return target, support
```

交叉算子 Python 复制代码

```python
@validate_arguments_init
@dataclass(eq=False, repr=False)
class CrossOp(nn.Module):

    in_channels: size2t
    out_channels: int
    kernel_size: size2t = 3
    nonlinearity: Optional[str] = "LeakyReLU"
    init_distribution: Optional[str] = "kaiming_normal"
    init_bias: Union[None, float, int] = 0.0

    def __post_init__(self):
        super().__init__()

        self.cross_conv = CrossConv2d(
            in_channels=as_2tuple(self.in_channels),
            out_channels=self.out_channels,
            kernel_size=self.kernel_size,
            padding=self.kernel_size // 2,
        )

        if self.nonlinearity is not None:
            self.nonlin = get_nonlinearity(self.nonlinearity)

        reset_conv2d_parameters(
            self, self.init_distribution, self.init_bias, self.nonlinearity
        )

    def forward(self, target, support):
        interaction = self.cross_conv(target, support).squeeze(dim=1)

        if self.nonlinearity is not None:
            interaction = vmap(self.nonlin, interaction)

        new_target = interaction.mean(dim=1, keepdims=True)

        return new_target, interaction
```

CrossConv2d Python 复制代码

```python
from typing import Optional, Tuple, Union

import einops as E
import torch
import torch.nn as nn
from pydantic import validate_arguments

size2t = Union[int, Tuple[int, int]]


class CrossConv2d(nn.Conv2d):
    """
    Compute pairwise convolution between all element of x and all elements of y.
    x, y are tensors of size B,_,C,H,W where _ could be different number of elements in x and y
    essentially, we do a meshgrid of the elements to get B,Sx,Sy,C,H,W tensors, and then
    pairwise conv.
    Args:
        x (tensor): B,Sx,Cx,H,W
        y (tensor): B,Sy,Cy,H,W
    Returns:
        tensor: B,Sx,Sy,Cout,H,W
    """
    """
    CrossConv2d is a convolutional layer that performs pairwise convolutions between elements of two input tensors.

    Parameters
    ----------
    in_channels : int or tuple of ints
        Number of channels in the input tensor(s).
        If the tensors have different number of channels, in_channels must be a tuple
    out_channels : int
        Number of output channels.
    kernel_size : int or tuple of ints
        Size of the convolutional kernel.
    stride : int or tuple of ints, optional
        Stride of the convolution. Default is 1.
    padding : int or tuple of ints, optional
        Zero-padding added to both sides of the input. Default is 0.
    dilation : int or tuple of ints, optional
        Spacing between kernel elements. Default is 1.
```

```
    groups : int, optional
        Number of blocked connections from input channels to output channels. Default is 1.
    bias : bool, optional
        If True, adds a learnable bias to the output. Default is True.
    padding_mode : str, optional
        Padding mode. Default is "zeros".
    device : str, optional
        Device on which to allocate the tensor. Default is None.
    dtype : torch.dtype, optional
        Data type assigned to the tensor. Default is None.

    Returns
    -------
    torch.Tensor
        Tensor resulting from the pairwise convolution between the elements of x and y.

    Notes
    -----
    x and y are tensors of size (B, Sx, Cx, H, W) and (B, Sy, Cy, H, W), respectively,
    The function does the cartesian product of the elements of x and y to obtain a tensor
    of size (B, Sx, Sy, Cx + Cy, H, W), and then performs the same convolution for all
    (B, Sx, Sy) in the batch dimension. Runtime and memory are O(Sx * Sy).

    Examples
    --------
    >>> x = torch.randn(2, 3, 4, 32, 32)
    >>> y = torch.randn(2, 5, 6, 32, 32)
    >>> conv = CrossConv2d(in_channels=(4, 6), out_channels=7, kernel_size=3, padding=1)
    >>> output = conv(x, y)
    >>> output.shape  #(2, 3, 5, 7, 32, 32)
    """

    @validate_arguments
    def __init__(
        self,
        in_channels: size2t,
        out_channels: int,
        kernel_size: size2t,
        stride: size2t = 1,
        padding: size2t = 0,
        dilation: size2t = 1,
```

```
        groups: int = 1,
        bias: bool = True,
        padding_mode: str = "zeros",
        device=None,
        dtype=None,
    ) -> None:

        if isinstance(in_channels, (list, tuple)):
            concat_channels = sum(in_channels)
        else:
            concat_channels = 2 * in_channels

        super().__init__(
            in_channels=concat_channels,
            out_channels=out_channels,
            kernel_size=kernel_size,
            stride=stride,
            padding=padding,
            dilation=dilation,
            groups=groups,
            bias=bias,
            padding_mode=padding_mode,
            device=device,
            dtype=dtype,
        )

    def forward(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
        """
        Compute pairwise convolution between all elements of x and all el
ements of y.

        Parameters
        ----------
        x : torch.Tensor
            Input tensor of size (B, Sx, Cx, H, W).
        y : torch.Tensor
            Input tensor of size (B, Sy, Cy, H, W).

        Returns
        -------
        torch.Tensor
            Tensor resulting from the cross-convolution between the eleme
nts of x and y.
            Has size (B, Sx, Sy, Co, H, W), where Co is the number of out
put channels.
        """
        B, Sx, *_ = x.shape
        _, Sy, *_ = y.shape
```

```
        xs = E.repeat(x, "B Sx Cx H W -> B Sx Sy Cx H W", Sy=Sy)
        ys = E.repeat(y, "B Sy Cy H W -> B Sx Sy Cy H W", Sx=Sx)

        xy = torch.cat([xs, ys], dim=3,)

        batched_xy = E.rearrange(xy, "B Sx Sy C2 H W -> (B Sx Sy) C2 H W")
        batched_output = super().forward(batched_xy)

        output = E.rearrange(
            batched_output, "(B Sx Sy) Co H W -> B Sx Sy Co H W", B=B, Sx=Sx, Sy=Sy
        )
        return output
```

训练部分

模型在多任务上进行训练

Algorithm 1 UniverSeg Training Loop using SGD with learning rate η over tasks $\mathcal{T}$, main architecture f_θ, in-task augmentations Aug_t and task augmentations $\text{Aug}_\mathcal{T}$

for $k = 1, \ldots, \text{NumTrainSteps}$ **do**
- $t \sim \mathcal{T}$ ▷ Sample Task
- $(x_i^t, y_i^t) \sim t$ ▷ Sample Query
- $S^t \leftarrow \{(x_j^t, y_j^t)\}_{j \neq i}^n$ ▷ Sample Support
- $x_i^t, y_i^t \leftarrow \text{Aug}_t(x_i^t, y_i^t)$ ▷ Augment Query
- $S^t \leftarrow \{\text{Aug}_t(x_j^t, y_j^t)\}_j^n$ ▷ Augment Support
- $x_i^t, y_i^t, S^t \leftarrow \text{Aug}_\mathcal{T}(x_i^t, y_i^t, S^t)$ ▷ Task Aug
- $\hat{y}_i \leftarrow f_\theta(x_i^t, S^t)$ ▷ Predict label map
- $\ell \leftarrow \mathcal{L}_{\text{seg}}(\hat{y}_i, y_i^t)$ ▷ Compute loss
- $\theta \leftarrow \theta - \eta \nabla_\theta \ell$ ▷ Gradient step

end for

任务增强(AugT (x，y，S))。类似于标准的数据增强，可以减少对训练样本的过度拟合，增强训练任务对于推广到新任务是有用的，特别是那些远离训练任务分布的任务。我们引入了任务增强–修改所有查询和支持图像，和/或所有分段映射，与相同类型的任务改变转换。任务增强示例包括分割映射的边缘检测或对所有图像和标签的水平翻转。我们提供了一个所有增强的列表和参数，我们在补充部分 C 中使用。

推理部分

对于给定的查询图像 xt，UniverSeg 预测分割 y = fθ (xt，St)，给定一个支持集 St，其中预测质量取决于支持集 St 的选择。为了减少这种依赖性，并在内存约束限制支持集大小时利用更多的数据进行推断，我们将 K 独立采样的支持集合的预测结合起来{ Sti } Ki = 1作为它们的像素级平均值，得到预测 $\hat{y} = \frac{1}{K} \sum_{k=1}^{K} f_\theta(x, S_k^t)$.

Python | 复制代码

```python
from example_data.wbc import WBCDataset

d_support = WBCDataset('JTSC', split='support', label='cytoplasm')
d_test = WBCDataset('JTSC', split='test', label='cytoplasm')

n_support = 64

support_images, support_labels = zip(*itertools.islice(d_support, n_support))
support_images = torch.stack(support_images).to(device)
support_labels = torch.stack(support_labels).to(device)

n_viz = 16
visualize_tensors({
    'Support Image': support_images[:n_viz],
    'Support Label': support_labels[:n_viz],
}, 8, title='Support Set Examples')

from collections import defaultdict

n_predictions = 10

results = defaultdict(list)
idxs = np.random.permutation(len(d_test))[:n_predictions]

for i in tqdm(idxs):
    image, label = d_test[i]
    vals = inference(model, image, label, support_images, support_labels)
    for k, v in vals.items():
        results[k].append(v)

scores = results.pop('score')
visualize_tensors(results, col_names=[f'Dice = {100*s:.1f}' for s in scores], title='Test Predictions', col_wrap=n_predictions)
```

优点和缺点

> 读者结合后面的实验部分，理解这些可能更深刻，特别是消融实验，
>
> 能看出哪些是真涨点，哪些是故事。

优点：

- 模型的泛化能力强。
- 只需要少量的support set，就能得到比较好的效果。
- 不需要拿到目标域的数据进行训练，直接推理即可。
- 小白也能使用

缺点：

- 模型必须跨任务，多数据集训练
- 最终效果也是做了ensemble才有的
- 非常依赖数据增强，模型本身的架构并不能带来很高的收益

experiments

> 结果的解读（20分）：读者能够理解并准确解释文献中的实验结果，包括数据、图表和其他视觉展示的内容。

数据集

为了训练我们的通用模型 fθ，我们采用了一组大而多样的分段任务，以便能够推广到新的任务。我们汇编了 MegaMedical——一个广泛的开放获取的医学分割数据集，包括不同的解剖学、成像模式和标签。它由53个数据集构成，包括26个医学领域和16种成像模式。

我们将原始数据集、处理过的图像和标签地图的各种格式的数据标准化。利用合成分割任务扩展训练数据，进一步提高训练任务的多样性。

MegaMedical 具有广泛的生物医学领域，如眼睛[37,58,66,80,95]，肺[85,89,92]，脊椎[107]，白细胞[108]，腹部[9,11,32,40,46,54,55,57,60,64,65,81,92]和脑[4,25,33,52,53,67,68,69,92]等。补充表3提供了 MegaMedical 数据集的详细列表。每个数据集的采集细节、受试者年龄范围和健康状况都不同。

医学图像任务的创建。虽然 MegaMedical 中的数据集具有多种成像任务和标签协议，但本文主要研究二维二值分割的一般问题。对于具有3D 数据的数据集，对于每个主题，我们沿着所有主轴提取体积的

2D 中间切片。当存在多种模式时，我们将每种模式作为一个新的任务来包括。对于包含多个分割标签的数据集，我们创建与可用标签一样多的二进制分割任务。将所有图像调整为128 × 128像素，并将亮度归一化为范围[0,1]。

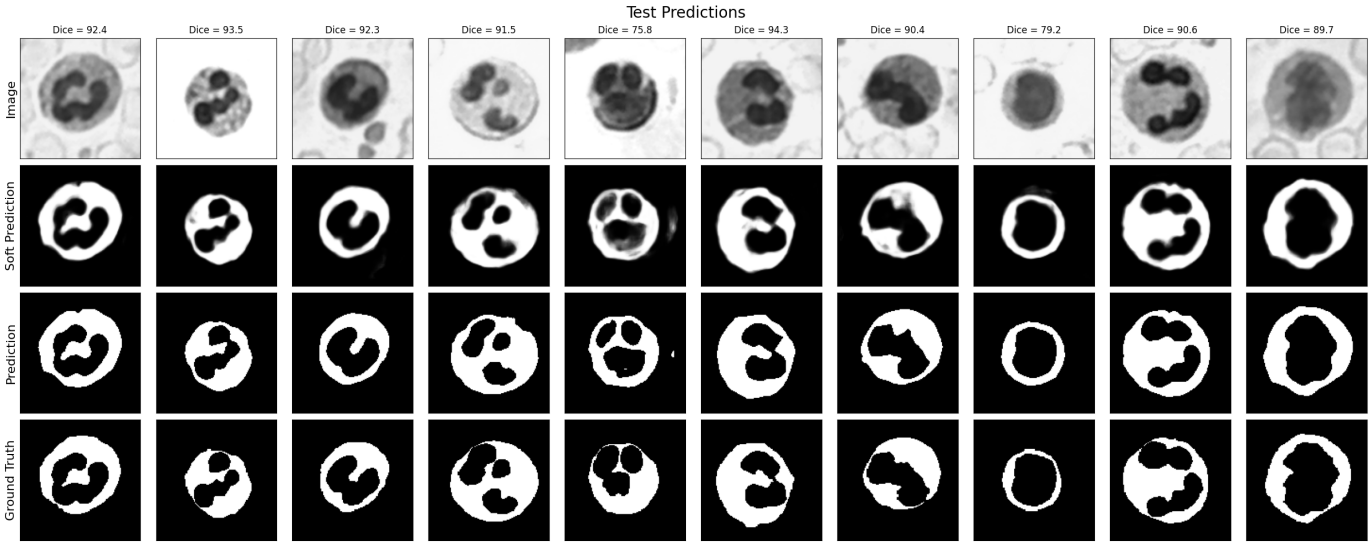
下面是一个非常简单的数据集准备代码：

wbc.py Python 复制代码

```python
import pathlib
import subprocess
from dataclasses import dataclass
from typing import Literal, Optional, Tuple

import numpy as np
import PIL
import torch
from torch.utils.data import Dataset


def process_img(path: pathlib.Path, size: Tuple[int, int]):
    img = PIL.Image.open(path)
    img = img.resize(size, resample=PIL.Image.BILINEAR)
    img = img.convert("L")
    img = np.array(img)
    img = img.astype(np.float32)
    return img


def process_seg(path: pathlib.Path, size: Tuple[int, int]):
    seg = PIL.Image.open(path)
    seg = seg.resize(size, resample=PIL.Image.NEAREST)
    seg = np.array(seg)
    seg = np.stack([seg == 0, seg == 128, seg == 255])
    seg = seg.astype(np.float32)
    return seg


def load_folder(path: pathlib.Path, size: Tuple[int, int] = (128, 128)):
    data = []
    for file in sorted(path.glob("*.bmp")):
        img = process_img(file, size=size)
        seg_file = file.with_suffix(".png")
        seg = process_seg(seg_file, size=size)
        data.append((img / 255.0, seg))
    return data


def require_download_wbc():
    dest_folder = pathlib.Path("/tmp/universeg_wbc/")

    if not dest_folder.exists():
        repo_url = "https://github.com/zxaoyou/segmentation_WBC.git"
        subprocess.run(
```

```
            ["git", "clone", repo_url, str(dest_folder),],
            stderr=subprocess.DEVNULL,
            check=True,
        )

    return dest_folder


@dataclass
class WBCDataset(Dataset):
    dataset: Literal["JTSC", "CV"]
    split: Literal["support", "test"]
    label: Optional[Literal["nucleus", "cytoplasm", "background"]] = None
    support_frac: float = 0.7

    def __post_init__(self):
        root = require_download_wbc()
        path = root / {"JTSC": "Dataset 1", "CV": "Dataset 2"}[self.dataset]
        T = torch.from_numpy
        self._data = [(T(x)[None], T(y)) for x, y in load_folder(path)]
        if self.label is not None:
            self._ilabel = {"cytoplasm": 1, "nucleus": 2, "background": 0}[self.label]
        self._idxs = self._split_indexes()

    def _split_indexes(self):
        rng = np.random.default_rng(42)
        N = len(self._data)
        p = rng.permutation(N)
        i = int(np.floor(self.support_frac * N))
        return {"support": p[:i], "test": p[i:]}[self.split]

    def __len__(self):
        return len(self._idxs)

    def __getitem__(self, idx):
        img, seg = self._data[self._idxs[idx]]
        if self.label is not None:
            seg = seg[self._ilabel][None]
        return img, seg
```

实验分析



从图像中可以看出，二值分割做的还是挺准确的。

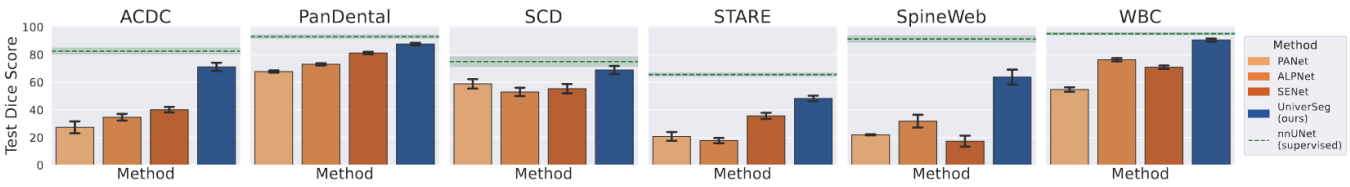


Figure 4: **Average Dice score per each held out dataset**. Performance of UniverSeg and several few-shot baselines, and the upper bound of each dataset determined by the individual fully-trained networks. For each of the unseen datasets, we average across tasks and subjects, and show the bootstrap variability in the error bars.

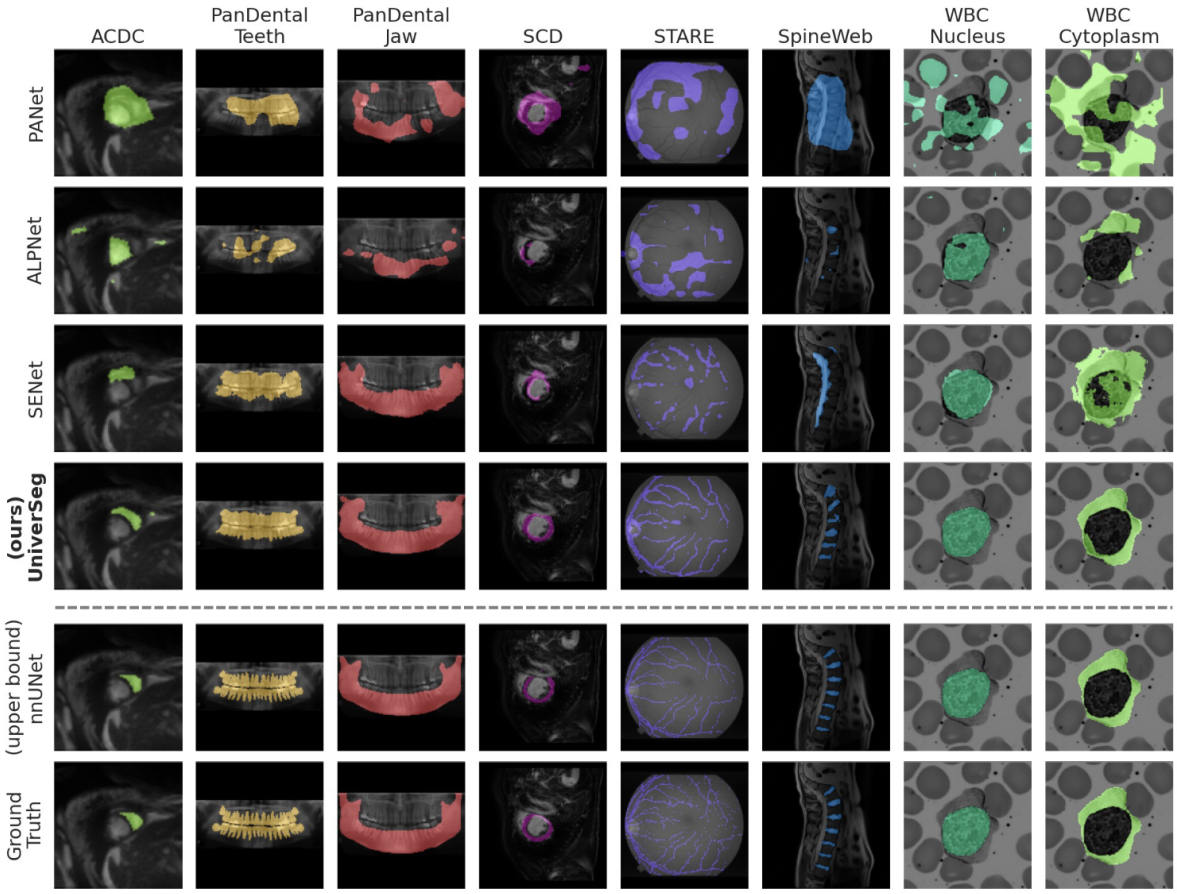


Figure 5: **Example model predictions for unseen tasks**. For a randomly sampled image per held-out task, we visualize the predictions of UniverSeg, few-shot baselines, and individually trained nnUNet models, along with ground truth maps.

Model	#Params	Runtime ms	Dice Score
PANet	14.71	240.0 ± 1.8	41.8 ± 1.3
ALPNet	43.02	527.7 ± 8.7	47.8 ± 1.1
SENet	0.92	4.1 ± 0.8	50.1 ± 1.3
UniverSeg (ours)	1.18	142.0 ± 0.4	**71.8** ± **0.9**
nnUNet (sup.)	17× 1.87	17× $1.4 \cdot 10^7$	84.4 ± 1.0

Table 1: **Performance Summary**. For UniverSeg and each FS baseline we report model size (in millions), inference run-time, and average held-out Dice score (with bootstrapping standard deviation) . As an upper bound, we include the set of 17 individually trained task-specific nnUNets for the 6 held-out datasets, where their run-time is their cumulative required training time.

上面也体现了，这个模型的优越性。模型参数量少，推理时间快，效果也好。

消融实验

Synth	Medical	In-Task	Task	ACDC	PanDental	SCD	STARE	SpineWeb	WBC
✓				55.4 ± 3.4	80.6 ± 1.3	55.7 ± 2.4	42.6 ± 2.5	50.1 ± 6.5	86.0 ± 1.4
	✓			44.9 ± 1.8	85.3 ± 0.9	59.9 ± 1.9	**63.8 ± 0.9**	40.3 ± 6.0	82.0 ± 1.6
✓	✓			50.6 ± 2.9	85.7 ± 0.9	59.0 ± 1.9	61.9 ± 1.6	45.6 ± 4.8	84.2 ± 1.4
	✓	✓		52.3 ± 4.3	86.5 ± 0.9	64.9 ± 2.7	56.0 ± 2.3	57.2 ± 3.7	85.1 ± 1.4
	✓		✓	68.0 ± 3.0	**87.5 ± 1.0**	63.5 ± 2.3	56.6 ± 2.1	57.8 ± 6.6	89.2 ± 1.3
	✓	✓	✓	**70.0 ± 2.8**	**88.0 ± 0.9**	**71.2 ± 3.1**	42.2 ± 2.1	58.4 ± 8.5	**90.3 ± 1.2**
✓	✓	✓	✓	**70.9 ± 2.9**	**87.5 ± 0.9**	**69.0 ± 2.9**	48.1 ± 2.0	**64.6 ± 5.4**	**90.6 ± 1.1**

消融训练策略。每个数据集提供了 UniverSeg 的模型，通过不同的组合训练提出了增加任务多样性的技术: 任务内增强，任务增强和合成任务。

Table 7: **Model Support Size**. Comparison of predictions for models trained with various of support sizes N and evaluated with and without ensembling $K = 10$ predictions. We report results on each held-out dataset as well as the global average. Standard deviation is computed by bootstrapping subjects before hierarchically averaging the data. For all datasets, we find that increasing the support size leads to better predictions, with diminishing returns after $N > 16$. Ensembling predictions significantly improve performance in the majority of settings (paired t-test).

N	K	ACDC	PanDental	SCD	STARE	SpineWeb	WBC	All (avg.)
1	1	41.3 ± 1.3	76.3 ± 0.9	60.2 ± 1.8	37.4 ± 3.8	30.4 ± 5.5	74.0 ± 1.2	53.3 ± 1.0
	10	44.5 ± 2.4	79.1 ± 1.0	60.0 ± 1.9	38.5 ± 4.0	32.4 ± 6.6	79.4 ± 1.4	55.7 ± 1.1
2	1	41.3 ± 2.6	80.0 ± 1.0	63.5 ± 2.0	40.4 ± 2.1	38.0 ± 4.3	77.6 ± 1.1	56.8 ± 1.0
	10	42.8 ± 3.2	82.4 ± 1.1	68.0 ± 2.5	40.7 ± 2.3	43.4 ± 4.1	82.3 ± 1.4	60.0 ± 1.2
4	1	53.9 ± 1.9	83.9 ± 1.0	64.7 ± 1.7	47.9 ± 2.9	45.5 ± 4.0	82.7 ± 1.4	63.1 ± 0.8
	10	57.0 ± 2.6	84.6 ± 1.1	66.4 ± 2.8	48.6 ± 2.9	50.8 ± 4.1	85.7 ± 1.5	65.5 ± 0.8
8	1	57.0 ± 2.5	85.0 ± 0.9	66.9 ± 3.2	45.9 ± 3.5	57.3 ± 6.5	83.7 ± 1.5	66.0 ± 1.3
	10	61.6 ± 3.3	86.1 ± 0.9	69.0 ± 4.1	47.1 ± 3.5	62.3 ± 6.0	85.9 ± 1.5	68.6 ± 1.3
16	1	64.1 ± 2.4	86.1 ± 0.9	69.1 ± 3.1	48.8 ± 3.0	64.4 ± 5.8	86.9 ± 1.4	69.9 ± 1.0
	10	66.8 ± 2.5	86.7 ± 0.9	68.7 ± 3.5	**49.7 ± 2.8**	**66.8 ± 5.7**	88.3 ± 1.5	71.2 ± 1.0
32	1	65.6 ± 3.0	87.1 ± 0.9	69.0 ± 2.0	45.7 ± 2.2	65.8 ± 4.6	87.6 ± 1.3	70.1 ± 0.9
	10	**69.3 ± 2.9**	**87.6 ± 0.9**	**69.5 ± 1.9**	**46.4 ± 2.1**	**66.4 ± 4.3**	88.9 ± 1.4	**71.4 ± 0.8**
64	1	69.0 ± 2.9	87.2 ± 0.9	68.7 ± 2.9	47.2 ± 2.2	64.2 ± 5.5	89.7 ± 1.1	71.0 ± 1.0
	10	**70.9 ± 2.9**	**87.5 ± 0.9**	**69.0 ± 2.9**	**48.1 ± 2.0**	**64.6 ± 5.4**	**90.6 ± 1.1**	**71.8 ± 0.9**

支撑集越多，ensemble越多，效果越好。

Table 4: **List of augmentations used in model training.**

Augmentation	Aug Type	Parameter Details
Flip Intensities	Task	$\mathbf{p} = 0.50$
Flip Labels	Task	$\mathbf{p} = 0.50$
Horizontal/Vertical Flip	Task	$\mathbf{p} = 0.50$
Sobel-Edge Label	Task	$\mathbf{p} = 0.50$
Task Affine Shift	Task	$\mathbf{p} = 0.50$ **degrees** $= [0, 360]$ **translate** $= [0, 0.2]$ **scale** $= [0.8, 1.1]$
Task Brightness Contrast Change	Task	$\mathbf{p} = 0.50$ **brightness** $= [-0.1, 0.1]$ **contrast** $= [0.8, 1.2]$
Task Elastic Warp	Task	$\mathbf{p} = 0.25$ $\boldsymbol{\alpha} = [1, 2]$ $\boldsymbol{\sigma} = [6, 8]$
Task Gaussian Blur	Task	$\mathbf{p} = 0.50$ **k-size** $= 5$ $\boldsymbol{\sigma} = [0.1, 1.1]$
Task Gaussian Noise	Task	$\mathbf{p} = 0.50$ $\boldsymbol{\mu} = [0, 0.05]$ $\boldsymbol{\sigma^2} = [0, 0.05]$
Task Sharpness Change	Task	$\mathbf{p} = 0.50$ **sharpness** $= 5$
Example Affine Shift	In-Task	$\mathbf{p} = 0.50$ **degrees** $= [0, 360]$ **translate** $= [0, 0.2]$ **scale** $= [0.8, 1.1]$
Example Brightness Contrast Change	In-Task	$\mathbf{p} = 0.25$ **brightness** $= [-0.1, 0.1]$ **contrast** $= [0.5, 1.5]$
Example Gaussian Blur	In-Task	$\mathbf{p} = 0.25$ **k-size** $= 5$ $\boldsymbol{\sigma} = [0.1, 1.1]$
Example Gaussian Noise	In-Task	$\mathbf{p} = 0.25$ $\boldsymbol{\mu} = [0, 0.05]$ $\boldsymbol{\sigma^2} = [0, 0.05]$
Example Sharpness Change	In-Task	$\mathbf{p} = 0.25$ **sharpness** $= 5$
Example Variable Elastic Warp	In-Task	$\mathbf{p} = 0.80$ $\boldsymbol{\alpha} = [1, 2.5]$ $\boldsymbol{\sigma} = [7, 8]$

任务相关的数据增强。

conclusion

> 对文献贡献的评估（20分）：读者能够评估文献对其研究领域的贡献，包括新的观点、技术、数据或理论。

- 文章提出了 UniverSeg，一种学习医疗图像分割单一任务标签不可见模型的方法。
- 文章使用大量和多样化的开放医学分割数据集来训练 UniverSeg，它能够泛化到未知的解剖和任务。
- 文章提出了一种新的交叉卷积运算，多尺度的交互查询和支持表示。

实验**非常硬**，UniverSeg 在所有公开的数据集中大大优于现有的少样本学习方法。通过广泛的消融研究，可以得出 UniverSeg 的性能在训练过程中强烈依赖于任务多样性，在推理过程中强烈依赖于支持集多样性。这突出了 UniverSeg 的实用性，可使用可变大小的支持集，潜在用户的数据集具有灵活性。

thoughts

> 批判性思考（20分）：读者能够批判性地思考文献的内容，包括对方法的选择、实验设计、数据分析、结论的合理性等方面的评价。

局限

在这项工作中，文章重点论证和深入分析 UniverSeg 的核心思想，使用二维数据和单一标签。

UniverSeg 能够很容易地适应科学家和临床研究人员确定的新的分割任务，而不需要对他们训练新的模型（通常来说，他们不太会写AI的code,调参数）

但现在很多医学影像领域都是3d的，该模型不适用3D。

未来可以扩展到3D分割领域，使用2.5D或3D的模型以及多标签，进一步缩小与上限的差距。

潜在的可以**发顶会的方向**。