# 改进yolo11-HSFPN等200+全套创新点大全：液体泼溅图像分割系统源码＆数据集全套

## 3.背景

研究背景与意义  
  
液体泼溅现象在许多领域中都具有重要的实际意义，例如环境监测、工业生产和安全管理等。液体泼溅不仅可能导致环境污染，还可能对设备和人员造成潜在的安全隐患。因此，准确识别和分割液体泼溅图像，对于及时采取措施、降低风险具有重要的现实意义。近年来，随着计算机视觉技术的快速发展，深度学习尤其是卷积神经网络（CNN）在图像分割任务中表现出了卓越的性能。YOLO（You Only Look Once）系列模型因其高效的实时检测能力而广泛应用于目标检测和图像分割任务中。  
  
本研究旨在基于改进的YOLOv11模型，构建一个高效的液体泼溅图像分割系统。该系统将利用一个包含2485张图像的数据集，专注于液体泼溅的检测与分割。该数据集仅包含一个类别，即“spillage”，并采用YOLOv8格式进行标注，确保了数据的规范性和一致性。通过对数据集的深入分析与处理，研究将探讨如何通过改进YOLOv11模型的结构和算法，提升液体泼溅图像的分割精度和实时性。  
  
此外，液体泼溅图像分割系统的研究不仅可以为工业和环境领域提供技术支持，还能为智能监控、自动化生产等应用场景提供重要的技术基础。通过实现高效的液体泼溅检测与分割，能够为相关领域的决策提供数据支持，进而推动相关技术的进步与应用。因此，本研究具有重要的理论价值和实际应用前景，为未来的研究提供了新的思路和方向。

## 2.视频效果

## 3. 图片效果展示

注意：本项目提供训练的数据集和训练教程,由于版本持续更新,暂不提供权重文件（best.pt）,请按照6.训练教程进行训练后实现上图演示的效果。

## 4. 数据集信息

本项目数据集信息介绍  
  
本项目旨在开发一个改进版的YOLOv11液体泼溅图像分割系统，为此我们构建了一个专门的数据集，命名为“spillage-1-”。该数据集专注于液体泼溅现象的图像识别与分割，涵盖了多种液体在不同环境下的泼溅场景。数据集中包含了丰富的图像样本，旨在为模型提供足够的训练数据，以提高其在液体泼溅检测和分割任务中的准确性和鲁棒性。  
  
数据集的类别数量为1，主要针对“spillage”这一主题进行深入研究。该类别包含了多种液体泼溅的情况，如水、油、果汁等，涉及不同的泼溅角度、速度和环境条件。这种多样性使得模型能够在不同的实际应用场景中表现出色。数据集中所收集的图像均经过精心标注，确保每个泼溅现象都能被准确识别和分割，进而为后续的深度学习模型训练提供高质量的基础。  
  
在数据集的构建过程中，我们注重图像的多样性和代表性，力求涵盖各种可能的液体泼溅情况，以便模型能够学习到液体在不同条件下的动态特征。此外，数据集还包括了不同光照条件、背景环境和液体性质的图像，以增强模型的泛化能力。通过对“spillage”类别的深入挖掘和细致标注，我们希望能够为液体泼溅图像分割领域的研究提供一个坚实的数据基础，推动相关技术的进步与应用。

## 5. 全套项目环境部署教程（零基础手把手教学）

5.1 所需软件PyCharm和Anaconda安装教程（第一步）

https://www.bilibili.com/video/BV1BoC1YCEKi/?spm\_id\_from=333.999.0.0&vd\_source=bc9aec86d164b67a7004b996143742dc

5.2 安装Python虚拟环境创建和依赖库安装视频教程（第二步）

https://www.bilibili.com/video/BV1ZoC1YCEBw?spm\_id\_from=333.788.videopod.sections&vd\_source=bc9aec86d164b67a7004b996143742dc

## 6. 改进YOLOv11训练教程和Web\_UI前端加载模型视频教程（零基础手把手教学）

https://www.bilibili.com/video/BV1BoC1YCEhR?spm\_id\_from=333.788.videopod.sections&vd\_source=bc9aec86d164b67a7004b996143742dc

## 7.原始YOLOv11算法讲解

## 8. 70+全套YOLOV8创新点原理讲解（非科班也可以轻松写刊发刊，V12版本正在科研待更新）

由于篇幅限制，每个创新点的具体原理讲解就不全部展开，具体见下列网址中的改进模块对应项目的技术原理博客网址【Blog】（创新点均为模块化搭建，原理适配YOLOv5~YOLOv11等各种版本）

https://gitee.com/qunmasj/good

## 9.系统功能展示（检测对象为举例，实际内容以本项目数据集为准）

图9.1.系统支持检测结果表格显示  
  
 图9.2.系统支持置信度和IOU阈值手动调节  
  
 图9.3.系统支持自定义加载权重文件best.pt(需要你通过步骤5中训练获得)  
  
 图9.4.系统支持摄像头实时识别  
  
 图9.5.系统支持图片识别  
  
 图9.6.系统支持视频识别  
  
 图9.7.系统支持识别结果文件自动保存  
  
 图9.8.系统支持Excel导出检测结果数据

## 10. YOLOv11核心改进模块源码讲解

```以下是对给定代码的核心部分进行分析和详细注释的结果：  
  
```python  
try:  
 # 尝试导入swattention模块和TransNext\_cuda中的所有内容  
 import swattention  
 from ultralytics.nn.backbone.TransNeXt.TransNext\_cuda import \*  
except ImportError as e:  
 # 如果导入失败（通常是因为没有安装CUDA版本），则导入TransNext\_native中的所有内容  
 from ultralytics.nn.backbone.TransNeXt.TransNext\_native import \*  
 pass  
```  
  
### 代码分析与注释  
  
1. \*\*导入模块\*\*：  
 - `try`语句块用于尝试执行可能会引发异常的代码。在这里，首先尝试导入`swattention`模块以及`TransNext\_cuda`中的所有内容。  
 - `import swattention`：这是一个自定义或第三方库，可能用于实现某种注意力机制（Attention Mechanism）。  
 - `from ultralytics.nn.backbone.TransNeXt.TransNext\_cuda import \*`：尝试从`TransNext\_cuda`模块导入所有定义的内容。`TransNext\_cuda`可能是一个基于CUDA的高效实现，适用于具有GPU加速的深度学习任务。  
  
2. \*\*异常处理\*\*：  
 - `except ImportError as e`：如果在`try`块中发生`ImportError`，则执行此块的代码。`ImportError`通常表示所需的模块未找到或无法导入。  
 - `from ultralytics.nn.backbone.TransNeXt.TransNext\_native import \*`：在导入CUDA版本失败后，导入`TransNext\_native`模块的所有内容。`TransNext\_native`可能是一个纯CPU实现，适用于没有CUDA支持的环境。  
  
3. \*\*`pass`语句\*\*：  
 - `pass`是一个空语句，表示在异常处理块中不执行任何操作。它的存在是为了保持代码结构的完整性。  
  
### 总结  
这段代码的核心功能是根据系统的环境条件选择合适的模块进行导入。如果CUDA支持可用，则使用GPU加速的实现；如果不可用，则回退到CPU实现。这种设计使得代码在不同的硬件环境下具有更好的兼容性和灵活性。```

这个文件名为 `TransNext.py`，主要功能是导入与 `TransNeXt` 相关的模块。代码的第一部分尝试导入 `swattention` 模块以及 `TransNext\_cuda` 中的内容，这通常意味着该模块可能是一个使用 CUDA 加速的实现，适用于支持 GPU 的环境。  
  
如果在导入过程中遇到 `ImportError`（即无法找到或加载指定的模块），代码会进入 `except` 块。在这个块中，它会尝试导入 `TransNext\_native` 模块。这通常表示该模块是一个不依赖于 CUDA 的实现，适用于不支持 GPU 的环境。  
  
通过这种方式，程序能够根据运行环境的不同，灵活选择合适的模块进行导入，从而确保在不同的硬件条件下都能正常运行。这种设计提高了代码的兼容性和灵活性，使得用户在不同的计算环境中都能使用 `TransNeXt` 的功能。

```以下是代码中最核心的部分，并附上详细的中文注释：  
  
```python  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
  
class GRNwithNHWC(nn.Module):  
 """ GRN (全局响应归一化) 层  
 该层用于对输入进行归一化处理，提升模型的性能。  
 输入假设为 (N, H, W, C) 格式。  
 """  
 def \_\_init\_\_(self, dim, use\_bias=True):  
 super().\_\_init\_\_()  
 self.use\_bias = use\_bias  
 # gamma 和 beta 是可学习的参数  
 self.gamma = nn.Parameter(torch.zeros(1, 1, 1, dim))  
 if self.use\_bias:  
 self.beta = nn.Parameter(torch.zeros(1, 1, 1, dim))  
  
 def forward(self, x):  
 # 计算输入的 L2 范数  
 Gx = torch.norm(x, p=2, dim=(1, 2), keepdim=True)  
 # 计算归一化因子  
 Nx = Gx / (Gx.mean(dim=-1, keepdim=True) + 1e-6)  
 if self.use\_bias:  
 return (self.gamma \* Nx + 1) \* x + self.beta  
 else:  
 return (self.gamma \* Nx + 1) \* x  
  
class UniRepLKNetBlock(nn.Module):  
 """ UniRepLKNet 的基本模块  
 该模块包含深度卷积、归一化、激活函数等。  
 """  
 def \_\_init\_\_(self, dim, kernel\_size, drop\_path=0., deploy=False, attempt\_use\_lk\_impl=True):  
 super().\_\_init\_\_()  
 # 深度卷积层  
 self.dwconv = nn.Conv2d(dim, dim, kernel\_size=kernel\_size, stride=1, padding=kernel\_size // 2, groups=dim, bias=True)  
 # 归一化层  
 self.norm = nn.BatchNorm2d(dim)  
 # Squeeze-and-Excitation (SE) 模块  
 self.se = SEBlock(dim, dim // 4)  
 # 前馈网络  
 ffn\_dim = dim \* 4  
 self.pwconv1 = nn.Linear(dim, ffn\_dim)  
 self.act = nn.GELU()  
 self.pwconv2 = nn.Linear(ffn\_dim, dim)  
  
 def forward(self, inputs):  
 # 前向传播  
 y = self.dwconv(inputs) # 深度卷积  
 y = self.norm(y) # 归一化  
 y = self.se(y) # SE 模块  
 y = self.pwconv1(y) # 前馈网络第一层  
 y = self.act(y) # 激活函数  
 y = self.pwconv2(y) # 前馈网络第二层  
 return y + inputs # 残差连接  
  
class UniRepLKNet(nn.Module):  
 """ UniRepLKNet 模型  
 该模型由多个 UniRepLKNetBlock 组成，进行特征提取和分类。  
 """  
 def \_\_init\_\_(self, in\_chans=3, num\_classes=1000, depths=(3, 3, 27, 3), dims=(96, 192, 384, 768)):  
 super().\_\_init\_\_()  
 self.stages = nn.ModuleList()  
 for i in range(len(depths)):  
 # 每个阶段由多个 UniRepLKNetBlock 组成  
 stage = nn.Sequential(\*[UniRepLKNetBlock(dim=dims[i], kernel\_size=3) for \_ in range(depths[i])])  
 self.stages.append(stage)  
  
 def forward(self, x):  
 # 前向传播，经过每个阶段  
 for stage in self.stages:  
 x = stage(x)  
 return x  
  
# 示例：创建模型并进行前向传播  
if \_\_name\_\_ == '\_\_main\_\_':  
 inputs = torch.randn((1, 3, 640, 640)) # 输入示例  
 model = UniRepLKNet() # 创建模型  
 res = model(inputs) # 前向传播  
 print(res.shape) # 输出结果的形状  
```  
  
### 代码核心部分解释：  
1. \*\*GRNwithNHWC\*\*：实现了全局响应归一化层，主要用于增强特征的表达能力。  
2. \*\*UniRepLKNetBlock\*\*：构建了一个基本的模块，包含深度卷积、归一化、SE模块和前馈网络，支持残差连接。  
3. \*\*UniRepLKNet\*\*：整体模型，由多个 `UniRepLKNetBlock` 组成，负责特征提取和分类任务。  
  
### 使用方法：  
在 `\_\_main\_\_` 部分创建了一个输入张量，并通过模型进行前向传播，输出结果的形状。```

这个程序文件 `UniRepLKNet.py` 实现了一个名为 UniRepLKNet 的深度学习模型，主要用于音频、视频、点云、时间序列和图像识别等任务。该模型的设计灵感来源于多个现有的网络架构，包括 RepLKNet、ConvNeXt、DINO 和 DeiT，并在这些基础上进行了改进。  
  
程序的开头部分包含了一些版权和许可证信息，随后导入了必要的库，包括 PyTorch 和一些自定义的模块。接下来，定义了一些基础组件，如 GRN（全局响应归一化）层、NCHW 和 NHWC 格式的转换层，以及卷积层的获取函数。通过这些基础组件，模型能够在不同的输入格式之间进行转换，以适应不同的计算需求。  
  
在模型的核心部分，定义了多个模块，包括 Squeeze-and-Excitation Block（SEBlock）和 Dilated Reparam Block。这些模块通过不同的卷积和归一化操作来提取特征，并在此基础上实现了更复杂的网络结构。Dilated Reparam Block 允许使用扩张卷积来捕捉更大范围的上下文信息，从而提高模型的表现。  
  
UniRepLKNet 的结构由多个 UniRepLKNetBlock 组成，每个块中包含卷积、归一化、激活和残差连接等操作。模型的设计允许在训练和推理阶段使用不同的结构，以提高效率和性能。  
  
此外，程序中还定义了一些辅助函数，如 `update\_weight` 用于加载预训练权重，`unireplknet\_a` 等函数用于创建不同配置的 UniRepLKNet 模型。每个函数根据指定的深度和维度初始化模型，并在需要时加载权重。  
  
在文件的最后部分，提供了一个简单的测试代码，创建了一个输入张量并通过模型进行前向传播，展示了模型的基本使用方法。  
  
总的来说，这个程序文件提供了一个灵活且高效的深度学习模型实现，适用于多种视觉和听觉任务，具备较强的可扩展性和可配置性。

```以下是经过简化和注释的核心代码部分，主要包括 `ConvBN`、`Block` 和 `StarNet` 类的实现。  
  
```python  
import torch  
import torch.nn as nn  
from timm.models.layers import DropPath, trunc\_normal\_  
  
class ConvBN(torch.nn.Sequential):  
 """  
 定义一个卷积层，后接批归一化层（可选）。  
 """  
 def \_\_init\_\_(self, in\_planes, out\_planes, kernel\_size=1, stride=1, padding=0, dilation=1, groups=1, with\_bn=True):  
 super().\_\_init\_\_()  
 # 添加卷积层  
 self.add\_module('conv', torch.nn.Conv2d(in\_planes, out\_planes, kernel\_size, stride, padding, dilation, groups))  
 # 如果需要，添加批归一化层  
 if with\_bn:  
 self.add\_module('bn', torch.nn.BatchNorm2d(out\_planes))  
 # 初始化批归一化层的权重和偏置  
 torch.nn.init.constant\_(self.bn.weight, 1)  
 torch.nn.init.constant\_(self.bn.bias, 0)  
  
class Block(nn.Module):  
 """  
 定义StarNet中的基本模块，包括深度卷积和元素级乘法。  
 """  
 def \_\_init\_\_(self, dim, mlp\_ratio=3, drop\_path=0.):  
 super().\_\_init\_\_()  
 # 深度卷积层  
 self.dwconv = ConvBN(dim, dim, 7, 1, (7 - 1) // 2, groups=dim, with\_bn=True)  
 # 两个1x1卷积层  
 self.f1 = ConvBN(dim, mlp\_ratio \* dim, 1, with\_bn=False)  
 self.f2 = ConvBN(dim, mlp\_ratio \* dim, 1, with\_bn=False)  
 # 另一个1x1卷积层  
 self.g = ConvBN(mlp\_ratio \* dim, dim, 1, with\_bn=True)  
 # 第二个深度卷积层  
 self.dwconv2 = ConvBN(dim, dim, 7, 1, (7 - 1) // 2, groups=dim, with\_bn=False)  
 self.act = nn.ReLU6() # 激活函数  
 self.drop\_path = DropPath(drop\_path) if drop\_path > 0. else nn.Identity() # 随机深度  
  
 def forward(self, x):  
 input = x # 保存输入  
 x = self.dwconv(x) # 深度卷积  
 x1, x2 = self.f1(x), self.f2(x) # 通过两个1x1卷积  
 x = self.act(x1) \* x2 # 元素级乘法  
 x = self.dwconv2(self.g(x)) # 通过另一个卷积层  
 x = input + self.drop\_path(x) # 残差连接  
 return x  
  
class StarNet(nn.Module):  
 """  
 StarNet网络结构的实现。  
 """  
 def \_\_init\_\_(self, base\_dim=32, depths=[3, 3, 12, 5], mlp\_ratio=4, drop\_path\_rate=0.0, num\_classes=1000, \*\*kwargs):  
 super().\_\_init\_\_()  
 self.num\_classes = num\_classes  
 self.in\_channel = 32  
 # stem层，初始卷积层  
 self.stem = nn.Sequential(ConvBN(3, self.in\_channel, kernel\_size=3, stride=2, padding=1), nn.ReLU6())  
 dpr = [x.item() for x in torch.linspace(0, drop\_path\_rate, sum(depths))] # 随机深度  
 # 构建网络的各个阶段  
 self.stages = nn.ModuleList()  
 cur = 0  
 for i\_layer in range(len(depths)):  
 embed\_dim = base\_dim \* 2 \*\* i\_layer # 当前阶段的嵌入维度  
 down\_sampler = ConvBN(self.in\_channel, embed\_dim, 3, 2, 1) # 下采样层  
 self.in\_channel = embed\_dim  
 blocks = [Block(self.in\_channel, mlp\_ratio, dpr[cur + i]) for i in range(depths[i\_layer])] # 当前阶段的Block  
 cur += depths[i\_layer]  
 self.stages.append(nn.Sequential(down\_sampler, \*blocks)) # 将下采样层和Block组合成一个阶段  
   
 self.apply(self.\_init\_weights) # 初始化权重  
  
 def \_init\_weights(self, m):  
 """  
 权重初始化函数。  
 """  
 if isinstance(m, (nn.Linear, nn.Conv2d)):  
 trunc\_normal\_(m.weight, std=.02) # 使用截断正态分布初始化权重  
 if isinstance(m, nn.Linear) and m.bias is not None:  
 nn.init.constant\_(m.bias, 0) # 初始化偏置为0  
 elif isinstance(m, (nn.LayerNorm, nn.BatchNorm2d)):  
 nn.init.constant\_(m.bias, 0) # 初始化偏置为0  
 nn.init.constant\_(m.weight, 1.0) # 初始化权重为1.0  
  
 def forward(self, x):  
 """  
 前向传播函数。  
 """  
 features = []  
 x = self.stem(x) # 通过stem层  
 features.append(x) # 保存特征  
 for stage in self.stages:  
 x = stage(x) # 通过每个阶段  
 features.append(x) # 保存特征  
 return features # 返回所有特征  
```  
  
### 代码说明：  
1. \*\*ConvBN\*\*: 这是一个自定义的卷积层，后接批归一化层，方便构建卷积神经网络。  
2. \*\*Block\*\*: 这是StarNet的基本构建块，包含深度卷积、1x1卷积和元素级乘法操作，使用残差连接。  
3. \*\*StarNet\*\*: 这是整个网络的实现，包含多个阶段，每个阶段由下采样层和多个Block组成。前向传播过程中会保存每个阶段的特征。  
  
这些核心部分构成了StarNet的基本结构，展示了其关键的元素级乘法操作。```

这个程序文件实现了一个名为StarNet的神经网络模型，主要用于图像处理任务。文件开头的文档字符串说明了该模型的设计理念，强调了元素级乘法的关键贡献，并指出该模型在设计上尽量简化，例如没有使用层级缩放和训练过程中的指数移动平均（EMA），这些设计选择可能会进一步提升性能。  
  
程序首先导入了必要的库，包括PyTorch和一些用于构建神经网络的模块。接着，定义了一个包含多个StarNet模型的列表，方便后续调用。模型的预训练权重链接也被定义在一个字典中，以便于下载和加载。  
  
在代码中，`ConvBN`类是一个简单的卷积层与批归一化层的组合，构造函数接受多个参数以配置卷积操作。它会在初始化时添加卷积层，并根据需要添加批归一化层，并对其权重和偏置进行初始化。  
  
`Block`类实现了StarNet的基本构建块，包含深度可分离卷积、两个全连接层和元素级乘法操作。该类的前向传播方法接收输入数据，经过一系列的卷积和激活函数处理后，返回经过跳跃连接的输出。  
  
`StarNet`类是整个网络的核心，包含多个阶段（stages），每个阶段由下采样层和多个Block组成。构造函数中定义了网络的基础维度、每个阶段的深度、MLP比率、随机丢弃率等参数，并使用这些参数构建网络结构。`\_init\_weights`方法用于初始化网络中各层的权重。  
  
在前向传播方法中，输入数据经过stem层处理后，依次通过每个阶段，提取特征并存储在一个列表中，最终返回这些特征。  
  
文件还定义了多个函数（如`starnet\_s1`、`starnet\_s2`等），用于创建不同配置的StarNet模型，并支持加载预训练权重。这些函数通过传递不同的参数来实例化StarNet类，从而生成具有不同深度和宽度的网络结构。  
  
最后，文件还提供了一些非常小的网络版本（如`starnet\_s050`、`starnet\_s100`、`starnet\_s150`），以适应不同的应用场景和计算资源限制。整体来看，这个程序文件展示了StarNet模型的设计与实现，强调了其简洁性和高效性。

### 程序整体功能和构架概括  
  
该项目包含多个深度学习模型的实现，主要用于图像处理和相关任务。每个文件定义了不同的网络架构，具有特定的功能和设计理念。以下是各个文件的主要功能概述：  
  
1. \*\*TransNext.py\*\*：实现了一个灵活的模型导入机制，能够根据运行环境选择合适的模块（CUDA 或非 CUDA 实现），为后续的模型训练和推理提供支持。  
  
2. \*\*UniRepLKNet.py\*\*：实现了UniRepLKNet模型，结合了多种现有网络架构的优点，适用于音频、视频、点云、时间序列和图像识别等任务。该模型设计了多个模块和块，以提取特征并进行高效的前向传播。  
  
3. \*\*block.py\*\*：定义了构建神经网络所需的基本模块，如卷积层、归一化层和激活函数。这些模块是构建更复杂网络结构的基础，提供了灵活的组合方式。  
  
4. \*\*starnet.py\*\*：实现了StarNet模型，强调元素级乘法和深度可分离卷积的使用。该模型结构简单且高效，适用于图像处理任务，并提供了多种配置以适应不同的计算需求。  
  
### 文件功能整理表  
  
| 文件名 | 功能描述 |  
|---------------------|--------------------------------------------------------------------------------------------|  
| TransNext.py | 实现了TransNeXt模型的导入机制，根据CUDA支持选择合适的模块，确保模型在不同环境下的兼容性。 |  
| UniRepLKNet.py | 实现了UniRepLKNet模型，结合多种网络架构，适用于多种视觉和听觉任务，具有高效的特征提取能力。 |  
| block.py | 定义了基本的神经网络模块，如卷积层和归一化层，为构建复杂网络提供基础组件。 |  
| starnet.py | 实现了StarNet模型，强调元素级乘法和深度可分离卷积，提供多种配置以适应不同的计算需求。 |  
  
这个表格总结了每个文件的主要功能，帮助理解整个项目的结构和目的。