# 改进yolo11-EfficientRepBiPAN等200+全套创新点大全：犬种实例分割系统源码＆数据集全套

## 3.背景

研究背景与意义  
  
随着计算机视觉技术的迅猛发展，实例分割作为其中一个重要的研究方向，已经在多个领域得到了广泛应用，包括自动驾驶、医疗影像分析和智能监控等。实例分割不仅需要识别图像中的物体，还要求精确地分割出每个物体的轮廓，这对于实现更高层次的智能理解至关重要。在众多实例分割算法中，YOLO（You Only Look Once）系列模型因其高效性和实时性而备受关注。YOLOv11作为该系列的最新版本，进一步提升了模型的精度和速度，为实例分割任务提供了更为强大的技术支持。  
  
本研究旨在基于改进的YOLOv11模型，构建一个专门针对犬种——柯基犬（Corgi）的实例分割系统。柯基犬因其独特的外形和广泛的受欢迎程度，成为了计算机视觉研究中的一个有趣案例。我们使用的Corgi v2数据集包含1600张经过精确标注的柯基犬图像，数据集的单一类别特性使得模型训练过程更加专注，有助于提升模型在特定犬种实例分割任务中的表现。  
  
通过对柯基犬的实例分割研究，不仅可以推动宠物识别技术的发展，还能为动物保护、宠物管理等相关领域提供数据支持和技术保障。此外，改进YOLOv11模型的应用也为实例分割技术的进一步优化提供了可能性。研究成果将为未来在其他犬种或动物实例分割任务中的应用奠定基础，推动计算机视觉技术在更广泛领域的应用与发展。

## 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  
  
本项目旨在改进YOLOv11的犬种实例分割系统，所使用的数据集以“Corgi v2”为主题，专注于对柯基犬这一特定犬种的实例分割任务。该数据集的设计初衷是为了提升模型在识别和分割柯基犬时的准确性和效率，进而推动计算机视觉领域在宠物识别方面的应用。数据集中包含了丰富的柯基犬图像，涵盖了不同的环境、姿态和光照条件，以确保模型能够在多样化的场景中进行有效的学习。  
  
在数据集的构建过程中，我们注重图像的多样性和质量，确保每一张图像都能为模型提供有价值的信息。数据集的类别数量为1，具体类别为“Corgi”，这意味着所有的图像均围绕柯基犬展开。通过集中训练于单一类别，我们可以更深入地挖掘柯基犬的特征，从而提高模型的专注度和识别能力。每张图像都经过精心标注，确保柯基犬的轮廓和特征得到准确的标识，为后续的实例分割提供了坚实的基础。  
  
此外，数据集还考虑到了柯基犬在不同背景下的表现，包括室内、室外、自然环境以及城市环境等。这种多样化的场景设置不仅增强了模型的泛化能力，也为实际应用提供了更广泛的适用性。通过使用该数据集进行训练，我们期望能显著提升YOLOv11在柯基犬实例分割任务中的表现，为未来的宠物识别技术发展奠定基础。总之，本项目的数据集不仅为模型训练提供了必要的素材，也为深入研究犬种识别和分割技术提供了宝贵的数据支持。

## 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

```以下是经过简化和注释的核心代码，保留了最重要的部分，并对每个部分进行了详细的中文注释：  
  
```python  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
from torch.autograd import Function  
import pywt  
  
# 创建小波滤波器  
def create\_wavelet\_filter(wave, in\_size, out\_size, type=torch.float):  
 # 通过pywt库获取小波的高通和低通滤波器  
 w = pywt.Wavelet(wave)  
 dec\_hi = torch.tensor(w.dec\_hi[::-1], dtype=type) # 低通滤波器  
 dec\_lo = torch.tensor(w.dec\_lo[::-1], dtype=type) # 高通滤波器  
   
 # 创建小波变换的滤波器  
 dec\_filters = torch.stack([  
 dec\_lo.unsqueeze(0) \* dec\_lo.unsqueeze(1),  
 dec\_lo.unsqueeze(0) \* dec\_hi.unsqueeze(1),  
 dec\_hi.unsqueeze(0) \* dec\_lo.unsqueeze(1),  
 dec\_hi.unsqueeze(0) \* dec\_hi.unsqueeze(1)  
 ], dim=0)  
  
 # 重复滤波器以适应输入通道数  
 dec\_filters = dec\_filters[:, None].repeat(in\_size, 1, 1, 1)  
  
 # 创建逆小波变换的滤波器  
 rec\_hi = torch.tensor(w.rec\_hi[::-1], dtype=type).flip(dims=[0])  
 rec\_lo = torch.tensor(w.rec\_lo[::-1], dtype=type).flip(dims=[0])  
 rec\_filters = torch.stack([  
 rec\_lo.unsqueeze(0) \* rec\_lo.unsqueeze(1),  
 rec\_lo.unsqueeze(0) \* rec\_hi.unsqueeze(1),  
 rec\_hi.unsqueeze(0) \* rec\_lo.unsqueeze(1),  
 rec\_hi.unsqueeze(0) \* rec\_hi.unsqueeze(1)  
 ], dim=0)  
  
 # 重复滤波器以适应输出通道数  
 rec\_filters = rec\_filters[:, None].repeat(out\_size, 1, 1, 1)  
  
 return dec\_filters, rec\_filters  
  
# 小波变换  
def wavelet\_transform(x, filters):  
 b, c, h, w = x.shape # 获取输入的形状  
 pad = (filters.shape[2] // 2 - 1, filters.shape[3] // 2 - 1) # 计算填充  
 x = F.conv2d(x, filters.to(x.dtype).to(x.device), stride=2, groups=c, padding=pad) # 进行卷积  
 x = x.reshape(b, c, 4, h // 2, w // 2) # 重塑输出形状  
 return x  
  
# 逆小波变换  
def inverse\_wavelet\_transform(x, filters):  
 b, c, \_, h\_half, w\_half = x.shape # 获取输入的形状  
 pad = (filters.shape[2] // 2 - 1, filters.shape[3] // 2 - 1) # 计算填充  
 x = x.reshape(b, c \* 4, h\_half, w\_half) # 重塑输入形状  
 x = F.conv\_transpose2d(x, filters.to(x.dtype).to(x.device), stride=2, groups=c, padding=pad) # 进行转置卷积  
 return x  
  
# 定义小波变换的函数  
class WaveletTransform(Function):  
 @staticmethod  
 def forward(ctx, input, filters):  
 ctx.filters = filters # 保存滤波器  
 with torch.no\_grad():  
 x = wavelet\_transform(input, filters) # 执行小波变换  
 return x  
  
 @staticmethod  
 def backward(ctx, grad\_output):  
 grad = inverse\_wavelet\_transform(grad\_output, ctx.filters) # 计算梯度  
 return grad, None  
  
# 定义逆小波变换的函数  
class InverseWaveletTransform(Function):  
 @staticmethod  
 def forward(ctx, input, filters):  
 ctx.filters = filters # 保存滤波器  
 with torch.no\_grad():  
 x = inverse\_wavelet\_transform(input, filters) # 执行逆小波变换  
 return x  
  
 @staticmethod  
 def backward(ctx, grad\_output):  
 grad = wavelet\_transform(grad\_output, ctx.filters) # 计算梯度  
 return grad, None  
  
# 定义小波卷积层  
class WTConv2d(nn.Module):  
 def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size=5, stride=1, bias=True, wt\_levels=1, wt\_type='db1'):  
 super(WTConv2d, self).\_\_init\_\_()  
  
 assert in\_channels == out\_channels # 输入通道数必须等于输出通道数  
  
 self.in\_channels = in\_channels  
 self.wt\_levels = wt\_levels  
 self.stride = stride  
  
 # 创建小波和逆小波滤波器  
 self.wt\_filter, self.iwt\_filter = create\_wavelet\_filter(wt\_type, in\_channels, in\_channels, torch.float)  
 self.wt\_filter = nn.Parameter(self.wt\_filter, requires\_grad=False)  
 self.iwt\_filter = nn.Parameter(self.iwt\_filter, requires\_grad=False)  
  
 # 基础卷积层  
 self.base\_conv = nn.Conv2d(in\_channels, in\_channels, kernel\_size, padding='same', stride=1, groups=in\_channels, bias=bias)  
  
 def forward(self, x):  
 # 小波变换和逆小波变换的前向传播  
 for i in range(self.wt\_levels):  
 x = wavelet\_transform(x, self.wt\_filter) # 小波变换  
  
 # 进行基础卷积  
 x = self.base\_conv(x)  
  
 # 逆小波变换  
 for i in range(self.wt\_levels):  
 x = inverse\_wavelet\_transform(x, self.iwt\_filter) # 逆小波变换  
  
 return x  
  
# 定义缩放模块  
class \_ScaleModule(nn.Module):  
 def \_\_init\_\_(self, dims, init\_scale=1.0):  
 super(\_ScaleModule, self).\_\_init\_\_()  
 self.weight = nn.Parameter(torch.ones(\*dims) \* init\_scale) # 初始化缩放权重  
  
 def forward(self, x):  
 return torch.mul(self.weight, x) # 进行缩放  
```  
  
### 代码说明：  
1. \*\*小波滤波器创建\*\*：`create\_wavelet\_filter` 函数用于生成小波变换和逆小波变换所需的滤波器。  
2. \*\*小波变换和逆小波变换\*\*：`wavelet\_transform` 和 `inverse\_wavelet\_transform` 函数分别实现小波变换和逆小波变换的具体操作。  
3. \*\*自定义函数\*\*：`WaveletTransform` 和 `InverseWaveletTransform` 类用于定义小波变换和逆小波变换的前向和反向传播。  
4. \*\*小波卷积层\*\*：`WTConv2d` 类实现了小波卷积层的结构，包含小波变换、基础卷积和逆小波变换的操作。  
5. \*\*缩放模块\*\*：`\_ScaleModule` 类用于实现对输入的缩放操作。  
  
这个简化的版本保留了核心功能，同时提供了详细的中文注释以帮助理解每个部分的作用。```

这个文件定义了一个名为 `WTConv2d` 的类，主要用于实现基于小波变换的二维卷积操作。该类继承自 PyTorch 的 `nn.Module`，并结合了小波变换和逆小波变换的功能。以下是对代码的逐步分析和说明。  
  
首先，文件导入了必要的库，包括 PyTorch 和 PyWavelets。`create\_wavelet\_filter` 函数用于生成小波变换和逆小波变换所需的滤波器。它接受小波类型、输入通道数和输出通道数作为参数，并使用 PyWavelets 库创建小波滤波器。生成的滤波器包括高通和低通滤波器，并通过张量操作进行组合，最后返回小波变换和逆小波变换的滤波器。  
  
接下来，`wavelet\_transform` 和 `inverse\_wavelet\_transform` 函数分别实现了小波变换和逆小波变换的具体操作。小波变换通过对输入张量进行卷积操作，将输入特征图分解为低频和高频部分。逆小波变换则是将这些部分重构回原始特征图。  
  
在 `WaveletTransform` 和 `InverseWaveletTransform` 类中，分别实现了小波变换和逆小波变换的前向和反向传播方法。这些类使用了 PyTorch 的自定义 `Function`，允许在反向传播时计算梯度。  
  
`wavelet\_transform\_init` 和 `inverse\_wavelet\_transform\_init` 函数用于初始化小波变换和逆小波变换的应用函数。这些函数返回一个应用于输入的函数，便于在 `WTConv2d` 类中使用。  
  
`WTConv2d` 类的构造函数中，首先检查输入通道数和输出通道数是否相等。然后，它创建小波变换和逆小波变换的滤波器，并将其设为不可训练的参数。接着，定义了基本的卷积层和小波卷积层，并使用 `\_ScaleModule` 类来实现特征图的缩放。  
  
在 `forward` 方法中，输入数据经过多层小波变换和逆小波变换的处理。首先，输入数据被分解为低频和高频部分，随后在每一层进行卷积和缩放操作。最后，经过逆小波变换重构出特征图，并与基本卷积的输出相加，形成最终的输出。  
  
`\_ScaleModule` 类是一个简单的缩放模块，用于对输入特征图进行缩放操作。它包含一个可训练的权重参数，能够根据需要调整输入的尺度。  
  
整体来看，这个文件实现了一个结合小波变换的卷积神经网络模块，能够在处理图像时提取多层次的特征信息，适用于图像处理和计算机视觉任务。

```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  
  
# 定义一个函数，用于生成相对位置的坐标  
def rel\_pos(kernel\_size):  
 # 生成从 -1 到 1 的线性空间，作为位置坐标  
 tensors = [torch.linspace(-1, 1, steps=kernel\_size) for \_ in range(2)]  
 # 创建网格坐标  
 kernel\_coord = torch.stack(torch.meshgrid(\*tensors), dim=-0)  
 kernel\_coord = kernel\_coord.unsqueeze(0) # 增加一个维度  
 return kernel\_coord  
  
# 定义一个自定义卷积层  
class SMPConv(nn.Module):  
 def \_\_init\_\_(self, planes, kernel\_size, n\_points, stride, padding, groups):  
 super().\_\_init\_\_()  
  
 self.planes = planes # 输出通道数  
 self.kernel\_size = kernel\_size # 卷积核大小  
 self.n\_points = n\_points # 关键点数量  
 self.init\_radius = 2 \* (2/kernel\_size) # 初始化半径  
  
 # 生成相对位置坐标  
 kernel\_coord = rel\_pos(kernel\_size)  
 self.register\_buffer('kernel\_coord', kernel\_coord) # 注册为缓冲区，不会被视为模型参数  
  
 # 初始化权重坐标  
 weight\_coord = torch.empty(1, n\_points, 2)  
 nn.init.trunc\_normal\_(weight\_coord, std=0.2, a=-1., b=1.) # 使用截断正态分布初始化  
 self.weight\_coord = nn.Parameter(weight\_coord) # 注册为可训练参数  
  
 # 初始化半径参数  
 self.radius = nn.Parameter(torch.empty(1, n\_points).unsqueeze(-1).unsqueeze(-1))  
 self.radius.data.fill\_(value=self.init\_radius) # 填充初始值  
  
 # 初始化卷积权重  
 weights = torch.empty(1, planes, n\_points)  
 nn.init.trunc\_normal\_(weights, std=.02) # 使用截断正态分布初始化  
 self.weights = nn.Parameter(weights) # 注册为可训练参数  
  
 def forward(self, x):  
 # 生成卷积核  
 kernels = self.make\_kernels().unsqueeze(1)  
 x = x.contiguous() # 确保输入张量是连续的  
 kernels = kernels.contiguous() # 确保卷积核张量是连续的  
  
 # 根据输入数据类型选择不同的卷积实现  
 if x.dtype == torch.float32:  
 x = \_DepthWiseConv2dImplicitGEMMFP32.apply(x, kernels) # 使用 FP32 的深度卷积  
 elif x.dtype == torch.float16:  
 x = \_DepthWiseConv2dImplicitGEMMFP16.apply(x, kernels) # 使用 FP16 的深度卷积  
 else:  
 raise TypeError("Only support fp32 and fp16, get {}".format(x.dtype)) # 抛出不支持的类型错误  
 return x   
  
 def make\_kernels(self):  
 # 计算卷积核的差异  
 diff = self.weight\_coord.unsqueeze(-2) - self.kernel\_coord.reshape(1, 2, -1).transpose(1, 2) # 计算权重坐标与卷积核坐标的差异  
 diff = diff.transpose(2, 3).reshape(1, self.n\_points, 2, self.kernel\_size, self.kernel\_size) # 调整维度  
 diff = F.relu(1 - torch.sum(torch.abs(diff), dim=2) / self.radius) # 计算经过 ReLU 激活的差异  
  
 # 计算卷积核  
 kernels = torch.matmul(self.weights, diff.reshape(1, self.n\_points, -1)) # 计算加权卷积核  
 kernels = kernels.reshape(1, self.planes, \*self.kernel\_coord.shape[2:]) # 调整维度  
 kernels = kernels.squeeze(0) # 去掉多余的维度  
 kernels = torch.flip(kernels.permute(0, 2, 1), dims=(1,)) # 翻转卷积核  
 return kernels  
  
# 定义一个包含卷积和激活函数的模块  
def conv\_bn\_relu(in\_channels, out\_channels, kernel\_size, stride, padding, groups, dilation=1, n\_points=None):  
 if padding is None:  
 padding = kernel\_size // 2 # 默认填充为卷积核大小的一半  
 result = nn.Sequential() # 创建一个顺序容器  
 result.add\_module('conv', SMPConv(in\_channels, out\_channels, kernel\_size, stride, padding, groups)) # 添加自定义卷积层  
 result.add\_module('nonlinear', nn.ReLU()) # 添加 ReLU 激活函数  
 return result  
  
# 定义主网络结构  
class SMPCNN(nn.Module):  
 def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size, stride, groups, n\_points=None):  
 super().\_\_init\_\_()  
 self.smp = conv\_bn\_relu(in\_channels, out\_channels, kernel\_size, stride, padding=kernel\_size // 2, groups=groups, n\_points=n\_points) # 添加卷积层  
 self.small\_conv = nn.Conv2d(in\_channels, out\_channels, kernel\_size=5, stride=stride, padding=2, groups=groups) # 添加小卷积层  
  
 def forward(self, inputs):  
 out = self.smp(inputs) # 通过自定义卷积层  
 out += self.small\_conv(inputs) # 加上小卷积层的输出  
 return out  
```  
  
### 代码说明：  
1. \*\*rel\_pos函数\*\*：生成相对位置坐标，用于卷积核的定位。  
2. \*\*SMPConv类\*\*：自定义卷积层，使用动态生成的卷积核，支持FP32和FP16数据类型。  
3. \*\*make\_kernels方法\*\*：计算卷积核的生成过程，涉及权重坐标和卷积核坐标的差异计算。  
4. \*\*conv\_bn\_relu函数\*\*：创建一个包含卷积和ReLU激活的模块。  
5. \*\*SMPCNN类\*\*：主网络结构，包含自定义卷积层和小卷积层的组合。  
  
这些核心部分构成了一个灵活且高效的卷积神经网络模块，适用于深度学习任务。```

这个程序文件 `SMPConv.py` 定义了一些用于深度学习模型的卷积模块，主要包括 `SMPConv`、`SMPCNN`、`SMPCNN\_ConvFFN` 和 `SMPBlock` 等类。以下是对代码的详细讲解。  
  
首先，文件导入了一些必要的库，包括 PyTorch 的核心库和一些自定义模块。特别是，它尝试导入深度可分离卷积的实现，如果导入失败则继续执行。  
  
`rel\_pos` 函数用于生成相对位置的坐标，这在卷积操作中可能用于计算相邻像素之间的关系。该函数创建一个二维网格，表示卷积核中每个点的相对位置。  
  
`SMPConv` 类是一个自定义的卷积层，继承自 `nn.Module`。在初始化方法中，它接受多个参数，包括输出通道数、卷积核大小、采样点数、步幅、填充和分组数。它使用 `rel\_pos` 函数生成卷积核的坐标，并初始化权重和半径。`forward` 方法实现了前向传播，使用不同的数据类型调用不同的深度可分离卷积实现。`make\_kernels` 方法则生成卷积核，通过计算权重坐标和卷积核坐标之间的差异，并应用 ReLU 激活函数来构建最终的卷积核。  
  
`radius\_clip` 方法用于限制半径的范围，确保其在给定的最小值和最大值之间。  
  
`get\_conv2d` 函数根据输入参数决定使用 `SMPConv` 还是标准的 `nn.Conv2d`，以便在满足特定条件时使用自定义卷积。  
  
`enable\_sync\_bn` 和 `get\_bn` 函数用于控制是否使用同步批归一化。`conv\_bn` 和 `conv\_bn\_relu` 函数则用于构建包含卷积层和批归一化层的顺序模块，并可选择性地添加 ReLU 激活函数。  
  
`fuse\_bn` 函数用于将卷积层和批归一化层融合，以提高推理速度。  
  
`SMPCNN` 类实现了一个包含自定义卷积层和小卷积层的网络结构。在前向传播中，它将两个卷积的输出相加，形成最终的输出。  
  
`SMPCNN\_ConvFFN` 类实现了一个前馈网络，包含两个逐点卷积层和一个非线性激活函数（GELU）。它还包含一个可选的 DropPath 机制，用于在训练过程中随机丢弃路径，以提高模型的泛化能力。  
  
最后，`SMPBlock` 类实现了一个包含多个卷积层和非线性激活函数的块结构。它通过前向传播将输入与经过卷积和激活后的输出相加，并应用 DropPath。  
  
整体而言，这个文件实现了一些复杂的卷积结构，旨在提高卷积神经网络的性能和灵活性，适用于各种深度学习任务。

```以下是经过简化并添加详细中文注释的核心代码部分：  
  
```python  
import sys  
import subprocess  
  
def run\_script(script\_path):  
 """  
 使用当前 Python 环境运行指定的脚本。  
  
 参数:  
 script\_path (str): 要运行的脚本路径  
  
 返回:  
 None  
 """  
 # 获取当前 Python 解释器的路径  
 python\_path = sys.executable  
  
 # 构建运行命令，使用 streamlit 运行指定的脚本  
 command = f'"{python\_path}" -m streamlit run "{script\_path}"'  
  
 # 执行命令，并等待其完成  
 result = subprocess.run(command, shell=True)  
   
 # 检查命令执行的返回码，0 表示成功，非0表示出错  
 if result.returncode != 0:  
 print("脚本运行出错。")  
  
# 如果该脚本是主程序，则执行以下代码  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 指定要运行的脚本路径  
 script\_path = "web.py" # 这里可以直接指定脚本名，假设在当前目录下  
  
 # 调用函数运行指定的脚本  
 run\_script(script\_path)  
```  
  
### 代码注释说明：  
1. \*\*导入模块\*\*：  
 - `sys`：用于获取当前 Python 解释器的路径。  
 - `subprocess`：用于执行外部命令。  
  
2. \*\*`run\_script` 函数\*\*：  
 - 接收一个脚本路径作为参数，并在当前 Python 环境中运行该脚本。  
 - 使用 `sys.executable` 获取当前 Python 解释器的路径。  
 - 构建命令字符串，使用 `streamlit` 运行指定的脚本。  
 - 使用 `subprocess.run` 执行命令，并等待其完成。  
 - 检查返回码，若非0则打印错误信息。  
  
3. \*\*主程序部分\*\*：  
 - 检查是否为主程序执行，如果是，则指定要运行的脚本路径并调用 `run\_script` 函数。```

这个程序文件名为 `ui.py`，其主要功能是使用当前的 Python 环境来运行一个指定的脚本，具体是通过 Streamlit 框架来启动一个 Web 应用。  
  
首先，程序导入了必要的模块，包括 `sys`、`os` 和 `subprocess`。其中，`sys` 模块用于访问与 Python 解释器相关的变量和函数，`os` 模块提供了与操作系统交互的功能，而 `subprocess` 模块则用于在 Python 中执行外部命令。  
  
接着，程序从 `QtFusion.path` 模块中导入了 `abs\_path` 函数，这个函数的作用是获取文件的绝对路径。  
  
在 `run\_script` 函数中，程序接受一个参数 `script\_path`，这个参数是要运行的脚本的路径。函数首先获取当前 Python 解释器的路径，存储在 `python\_path` 变量中。然后，构建一个命令字符串 `command`，这个命令使用当前的 Python 解释器来运行指定的脚本，并通过 Streamlit 的 `run` 命令来启动 Web 应用。  
  
使用 `subprocess.run` 方法执行这个命令，`shell=True` 参数表示在 shell 中执行命令。执行后，程序检查返回码 `result.returncode`，如果返回码不为 0，表示脚本运行出错，程序会打印出相应的错误信息。  
  
在文件的最后部分，程序通过 `if \_\_name\_\_ == "\_\_main\_\_":` 语句来判断是否直接运行该脚本。如果是，则指定要运行的脚本路径为 `web.py`，并调用 `run\_script` 函数来执行这个脚本。  
  
总的来说，这个程序的核心功能是启动一个 Streamlit Web 应用，方便用户在本地环境中进行开发和测试。

```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  
import numpy as np  
from timm.models.layers import DropPath, to\_2tuple, trunc\_normal\_  
  
class Mlp(nn.Module):  
 """ 多层感知机 (MLP) 模块。"""  
  
 def \_\_init\_\_(self, in\_features, hidden\_features=None, out\_features=None, act\_layer=nn.GELU, drop=0.):  
 super().\_\_init\_\_()  
 out\_features = out\_features or in\_features # 输出特征数  
 hidden\_features = hidden\_features or in\_features # 隐藏层特征数  
 self.fc1 = nn.Linear(in\_features, hidden\_features) # 第一层线性变换  
 self.act = act\_layer() # 激活函数  
 self.fc2 = nn.Linear(hidden\_features, out\_features) # 第二层线性变换  
 self.drop = nn.Dropout(drop) # Dropout层  
  
 def forward(self, x):  
 """ 前向传播函数。"""  
 x = self.fc1(x) # 线性变换  
 x = self.act(x) # 激活  
 x = self.drop(x) # Dropout  
 x = self.fc2(x) # 线性变换  
 x = self.drop(x) # Dropout  
 return x  
  
class WindowAttention(nn.Module):  
 """ 基于窗口的多头自注意力 (W-MSA) 模块。"""  
  
 def \_\_init\_\_(self, dim, window\_size, num\_heads, qkv\_bias=True, qk\_scale=None, attn\_drop=0., proj\_drop=0.):  
 super().\_\_init\_\_()  
 self.dim = dim # 输入通道数  
 self.window\_size = window\_size # 窗口大小  
 self.num\_heads = num\_heads # 注意力头数  
 head\_dim = dim // num\_heads # 每个头的维度  
 self.scale = qk\_scale or head\_dim \*\* -0.5 # 缩放因子  
  
 # 定义相对位置偏置参数表  
 self.relative\_position\_bias\_table = nn.Parameter(  
 torch.zeros((2 \* window\_size[0] - 1) \* (2 \* window\_size[1] - 1), num\_heads)) # 位置偏置表  
  
 # 计算每个token在窗口内的相对位置索引  
 coords\_h = torch.arange(self.window\_size[0])  
 coords\_w = torch.arange(self.window\_size[1])  
 coords = torch.stack(torch.meshgrid([coords\_h, coords\_w])) # 生成网格坐标  
 coords\_flatten = torch.flatten(coords, 1) # 展平坐标  
 relative\_coords = coords\_flatten[:, :, None] - coords\_flatten[:, None, :] # 计算相对坐标  
 relative\_coords = relative\_coords.permute(1, 2, 0).contiguous() # 变换维度  
 relative\_coords[:, :, 0] += self.window\_size[0] - 1 # 偏移  
 relative\_coords[:, :, 1] += self.window\_size[1] - 1  
 relative\_coords[:, :, 0] \*= 2 \* self.window\_size[1] - 1  
 relative\_position\_index = relative\_coords.sum(-1) # 计算相对位置索引  
 self.register\_buffer("relative\_position\_index", relative\_position\_index) # 注册为缓冲区  
  
 self.qkv = nn.Linear(dim, dim \* 3, bias=qkv\_bias) # 线性变换生成Q、K、V  
 self.attn\_drop = nn.Dropout(attn\_drop) # 注意力权重的Dropout  
 self.proj = nn.Linear(dim, dim) # 输出线性变换  
 self.proj\_drop = nn.Dropout(proj\_drop) # 输出的Dropout  
 trunc\_normal\_(self.relative\_position\_bias\_table, std=.02) # 初始化相对位置偏置  
 self.softmax = nn.Softmax(dim=-1) # Softmax层  
  
 def forward(self, x, mask=None):  
 """ 前向传播函数。"""  
 B\_, N, C = x.shape # B: 批量大小, N: token数量, C: 通道数  
 qkv = self.qkv(x).reshape(B\_, N, 3, self.num\_heads, C // self.num\_heads).permute(2, 0, 3, 1, 4) # 计算Q、K、V  
 q, k, v = qkv[0], qkv[1], qkv[2] # 分离Q、K、V  
  
 q = q \* self.scale # 缩放Q  
 attn = (q @ k.transpose(-2, -1)) # 计算注意力权重  
  
 # 添加相对位置偏置  
 relative\_position\_bias = self.relative\_position\_bias\_table[self.relative\_position\_index.view(-1)].view(  
 self.window\_size[0] \* self.window\_size[1], self.window\_size[0] \* self.window\_size[1], -1) # 计算相对位置偏置  
 relative\_position\_bias = relative\_position\_bias.permute(2, 0, 1).contiguous() # 变换维度  
 attn = attn + relative\_position\_bias.unsqueeze(0) # 加入偏置  
  
 if mask is not None: # 如果有mask  
 nW = mask.shape[0]  
 attn = attn.view(B\_ // nW, nW, self.num\_heads, N, N) + mask.unsqueeze(1).unsqueeze(0) # 应用mask  
 attn = attn.view(-1, self.num\_heads, N, N)  
 attn = self.softmax(attn) # 归一化  
 else:  
 attn = self.softmax(attn) # 归一化  
  
 attn = self.attn\_drop(attn) # Dropout  
  
 x = (attn @ v).transpose(1, 2).reshape(B\_, N, C) # 计算输出  
 x = self.proj(x) # 线性变换  
 x = self.proj\_drop(x) # Dropout  
 return x  
  
class SwinTransformer(nn.Module):  
 """ Swin Transformer 主体。"""  
  
 def \_\_init\_\_(self, pretrain\_img\_size=224, patch\_size=4, in\_chans=3, embed\_dim=96, depths=[2, 2, 6, 2],  
 num\_heads=[3, 6, 12, 24], window\_size=7, mlp\_ratio=4., qkv\_bias=True, drop\_rate=0.,  
 attn\_drop\_rate=0., drop\_path\_rate=0.2, norm\_layer=nn.LayerNorm, ape=False, patch\_norm=True,  
 out\_indices=(0, 1, 2, 3), frozen\_stages=-1, use\_checkpoint=False):  
 super().\_\_init\_\_()  
  
 self.patch\_embed = PatchEmbed(patch\_size=patch\_size, in\_chans=in\_chans, embed\_dim=embed\_dim) # 图像分块嵌入  
 self.pos\_drop = nn.Dropout(p=drop\_rate) # Dropout层  
  
 # 构建层  
 self.layers = nn.ModuleList()  
 for i\_layer in range(len(depths)):  
 layer = BasicLayer(  
 dim=int(embed\_dim \* 2 \*\* i\_layer),  
 depth=depths[i\_layer],  
 num\_heads=num\_heads[i\_layer],  
 window\_size=window\_size,  
 mlp\_ratio=mlp\_ratio,  
 qkv\_bias=qkv\_bias,  
 drop=drop\_rate,  
 attn\_drop=attn\_drop\_rate,  
 drop\_path=drop\_path\_rate,  
 norm\_layer=norm\_layer,  
 downsample=PatchMerging if (i\_layer < len(depths) - 1) else None,  
 use\_checkpoint=use\_checkpoint)  
 self.layers.append(layer)  
  
 def forward(self, x):  
 """ 前向传播函数。"""  
 x = self.patch\_embed(x) # 图像分块嵌入  
 x = self.pos\_drop(x) # Dropout  
  
 outs = []  
 for layer in self.layers:  
 x\_out, \_, \_, x, \_, \_ = layer(x, x.size(1), x.size(2)) # 逐层前向传播  
 outs.append(x\_out) # 收集输出  
  
 return outs # 返回所有层的输出  
```  
  
### 主要功能说明：  
1. \*\*Mlp类\*\*：实现了一个简单的多层感知机，包括两层线性变换和激活函数。  
2. \*\*WindowAttention类\*\*：实现了窗口基础的多头自注意力机制，支持相对位置偏置。  
3. \*\*SwinTransformer类\*\*：实现了Swin Transformer的主体结构，包含了图像分块嵌入、多个Transformer层的堆叠，以及最终的前向传播逻辑。  
  
这些类和方法构成了Swin Transformer的核心功能，适用于图像处理和计算机视觉任务。```

这个程序文件实现了Swin Transformer模型的主要结构，Swin Transformer是一种基于视觉的Transformer架构，采用了分层的设计和窗口注意力机制。代码中包含多个类和函数，每个部分负责模型的不同功能。  
  
首先，导入了必要的库，包括PyTorch和一些自定义的层。`Mlp`类实现了一个多层感知机（MLP），包含两个线性层和一个激活函数（默认为GELU），并在每个线性层后添加了Dropout以防止过拟合。  
  
接下来，`window\_partition`和`window\_reverse`函数用于将输入特征图划分为窗口和将窗口合并回特征图。这是Swin Transformer的核心思想之一，通过局部窗口进行自注意力计算来提高计算效率。  
  
`WindowAttention`类实现了窗口基础的多头自注意力机制（W-MSA），支持相对位置偏置。该类的构造函数定义了输入通道数、窗口大小、注意力头数等参数，并初始化了相对位置偏置表。前向传播方法计算查询、键、值，并应用相对位置偏置。  
  
`SwinTransformerBlock`类定义了Swin Transformer的基本块，包含归一化层、窗口注意力层和MLP。它还实现了窗口的循环移位，以便在不同的块之间共享信息。  
  
`PatchMerging`类用于将特征图的补丁合并，降低特征图的分辨率。它通过线性层将4个输入通道合并为2个输出通道，并应用归一化。  
  
`BasicLayer`类表示Swin Transformer的一个基本层，包含多个Swin Transformer块，并在最后可能应用下采样。  
  
`PatchEmbed`类负责将输入图像划分为补丁并进行嵌入，使用卷积层将图像转换为嵌入向量。  
  
`SwinTransformer`类是整个模型的主类，构造函数中定义了各个层的参数，并构建了模型的各个层。它还支持绝对位置嵌入和随机深度。  
  
最后，`update\_weight`函数用于更新模型的权重，`SwinTransformer\_Tiny`函数则用于创建一个小型的Swin Transformer模型实例，并加载预训练权重（如果提供）。  
  
整体而言，这个文件实现了Swin Transformer的核心组件，适用于图像分类等视觉任务。通过窗口注意力机制和分层设计，Swin Transformer在处理高分辨率图像时表现出色。

### 程序整体功能和构架概括  
  
该程序包含多个模块，主要用于构建和运行深度学习模型，特别是针对计算机视觉任务的卷积神经网络和Transformer架构。各个文件实现了不同的功能，形成了一个完整的深度学习框架。具体功能包括：  
  
1. \*\*wtconv2d.py\*\*：实现了基于小波变换的二维卷积操作，结合了传统卷积和小波变换的优点，适用于图像处理任务。  
2. \*\*SMPConv.py\*\*：定义了一些自定义卷积层和网络结构，采用深度可分离卷积和其他创新的卷积设计，旨在提高模型的性能和灵活性。  
3. \*\*ui.py\*\*：提供了一个简单的用户界面，使用Streamlit框架来启动Web应用，方便用户进行模型的测试和开发。  
4. \*\*SwinTransformer.py\*\*：实现了Swin Transformer模型，采用窗口注意力机制和分层设计，适用于高分辨率图像的处理。  
  
### 文件功能整理表  
  
| 文件名 | 功能描述 |  
|---------------------|------------------------------------------------------------------------------------------|  
| wtconv2d.py | 实现基于小波变换的二维卷积操作，结合小波变换和逆小波变换的功能，适用于图像处理任务。 |  
| SMPConv.py | 定义自定义卷积层和网络结构，采用深度可分离卷积和其他创新卷积设计，提高模型性能和灵活性。 |  
| ui.py | 提供一个简单的用户界面，使用Streamlit框架启动Web应用，方便用户进行模型的测试和开发。 |  
| SwinTransformer.py | 实现Swin Transformer模型，采用窗口注意力机制和分层设计，适用于高分辨率图像的处理。 |  
  
这个表格清晰地展示了每个文件的功能，帮助理解整个程序的架构和各个模块之间的关系。