Module **unet**

▶ EXPAND SOURCE CODE

Functions

```
def sinusoidal_embedding(n, d)
```

Generates a sinusoidal embedding of size n x d.

Parameters

n (int): The number of rows in the embedding. d (int): The number of columns in the embedding.

Returns

`torch.Tensor`
The sinusoidal embedding of size n x d.

▶ EXPAND SOURCE CODE

Classes

```
class MyBlock  (shape, in_c, out_c, kernel_size=3, stride=1, padding=1, activation=None,
normalize=True)
```

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes::

```
 import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call :meth:`to`, etc.

Note

As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

:ivar training: Boolean represents whether this module is in training or evaluation mode. :vartype training: bool

Initializes a new instance of the MyBlock class.

Parameters

shape (tuple): The shape of the input tensor. in_c (int): The number of input channels. out_c (int): The number of output channels. kernel_size (int, optional): The size of the convolutional kernel. Defaults to 3. stride (int, optional): The stride of the convolutional kernel. Defaults to 1. padding (int, optional): The padding of the convolutional kernel. Defaults to 1. activation (nn.Module, optional): The activation function to use. Defaults to None. normalize (bool, optional): Whether to apply layer normalization. Defaults to True.

▶ EXPAND SOURCE CODE

Ancestors

torch.nn.modules.module.Module

Methods

```
def forward(self, x) -> Callable[..., Any]
```

Forward pass of the model.

Args

`x` : `torch.Tensor`
The input tensor.

Returns

`torch.Tensor`
The output tensor after passing through the model.

▶ EXPAND SOURCE CODE

```
class MyUNet  (channel, size, n_steps=500,
time_emb_dim=100)
```

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes::

```
 import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
```

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call :meth: `to` , etc.

Note

As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

:ivar training: Boolean represents whether this module is in training or evaluation mode. :vartype training: bool

Initializes the MyUNet model, a U-Net architecture with customizable configurations for image segmentation tasks.

This constructor initializes various layers and components based on the specified image size and configurations. It sets up a U-Net architecture with optional depth based on the 'size' parameter which can be 128 or 256 pixels, and constructs blocks and transitional layers correspondingly. The architecture uses sinusoidal embeddings for time encoding and multiple encoding and decoding blocks with convolutional and transposed convolutional layers.

Parameters: - channel (int): The number of channels in the input and output images. Typically, this corresponds to the number of color channels (e.g., 3 for RGB images). - size (int): The dimension of the input images (e.g., 128 or 256 pixels). This size determines the depth and complexity of the U-Net architecture. - n_steps (int, optional): The number of steps or positions for the time embedding, with a default of 500. This is used in the sinusoidal time embedding. - time_emb_dim (int, optional): The dimensionality of the time embedding vector, with a default of 100.

Raises: - ValueError: If the 'size' parameter is not recognized (i.e., not 128 or 256), a ValueError is raised.

The network comprises several stages of downsampling followed by upsampling to create a symmetric architecture, with additional bottleneck layers. Time-dependent features are integrated at multiple levels of the network.

Ancestors

torch.nn.modules.module.Module

Methods

```
def forward(self, x, t) -> Callable[..., Any]
```

Forward pass of the model.

Args

`x` : `torch.Tensor`
The input tensor.

`t` : `torch.Tensor`
The time tensor.

Returns

`torch.Tensor`
The output tensor after passing through the model.

Index

Functions

Classes