U-Net Lane Detection

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

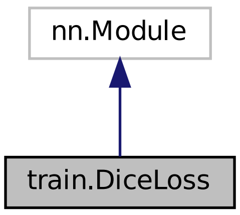
Chapter 3

Class Documentation

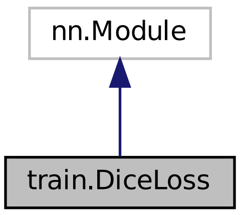
3.1 train.DiceLoss Class Reference

Dice Loss for segmentation tasks.

Inheritance diagram for train.DiceLoss:



Collaboration diagram for train.DiceLoss:



Public Member Functions

- def **__init__** (self)
- def forward (self, inputs, targets, smooth=1e-8)

    Computes the Dice Loss between predictions and targets.

3.1.1 Detailed Description

Dice Loss for segmentation tasks.

This class implements Dice Loss, which measures the overlap between predicted and ground truth masks.

Note

Adds a small smooth factor to avoid division by zero.

3.1.2 Member Function Documentation

3.1.2.1 forward()

```
def train.DiceLoss.forward (
              self,
              inputs,
              targets,
              smooth = 1e-8 )
```

Computes the Dice Loss between predictions and targets.

Parameters

inputs	(torch.Tensor): Raw model outputs (logits).
targets	(torch.Tensor): Ground truth binary masks.
smooth	(float, optional): Smoothing factor to avoid division by zero (default: 1e-8).

Returns

torch.Tensor: Computed Dice Loss.

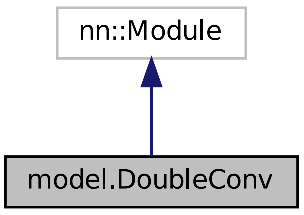
The documentation for this class was generated from the following file:

- train.py

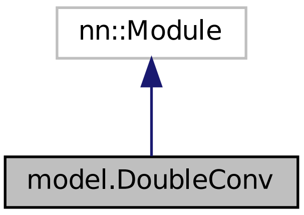
3.2 model.DoubleConv Class Reference

A double convolution block for the U-Net architecture.

Inheritance diagram for model.DoubleConv:



Collaboration diagram for model.DoubleConv:



Public Member Functions

- def **__init__** (self, in_channels, out_channels, dropout_rate=0.1)
- def forward (self, x)

  Forward pass of the double convolution block.

Public Attributes

- **conv**

3.2.1 Detailed Description

A double convolution block for the U-Net architecture.

This class implements a block consisting of two convolutional layers, each followed by batch normalization, ReLU activation, and a dropout layer for regularization.

Parameters

in_channels	(int): Number of input channels.
out_channels	(int): Number of output channels.
dropout_rate	(float, optional): Dropout probability for regularization (default: 0.1).

3.2.2 Member Function Documentation

3.2.2.1 forward()

```
def model.DoubleConv.forward (
              self,
              x )
```

Forward pass of the double convolution block.

Parameters

x	(torch.Tensor): Input tensor of shape [batch_size, in_channels, height, width].

Returns

torch.Tensor: Output tensor of shape [batch_size, out_channels, height, width].

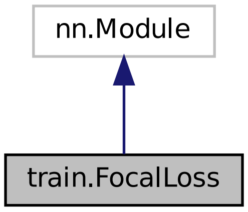
The documentation for this class was generated from the following file:

- model.py

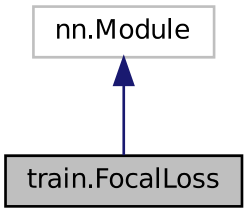
3.3 train.FocalLoss Class Reference

Focal Loss for handling class imbalance in segmentation tasks.

Inheritance diagram for train.FocalLoss:



Collaboration diagram for train.FocalLoss:



Public Member Functions

- def **__init__** (self, alpha=0.95, gamma=2.0, reduction='mean')
- def forward (self, inputs, targets)

    Computes the Focal Loss between predictions and targets.

Public Attributes

- **alpha**
- **gamma**
- **reduction**

3.3.1 Detailed Description

Focal Loss for handling class imbalance in segmentation tasks.

This class implements Focal Loss, which focuses training on hard examples by reducing the weight of easily classified samples.

Parameters

alpha	(float, optional): Weight for the positive class (default: 0.95).
gamma	(float, optional): Focusing parameter (default: 2.0).
reduction	(str, optional): Reduction method for the loss ('mean', 'sum', or 'none') (default: 'mean').

3.3.2 Member Function Documentation

3.3.2.1 forward()

```
def train.FocalLoss.forward (
              self,
              inputs,
              targets )
```

Computes the Focal Loss between predictions and targets.

Parameters

inputs	(torch.Tensor): Raw model outputs (logits).
targets	(torch.Tensor): Ground truth binary masks.

Returns

torch.Tensor: Computed Focal Loss.

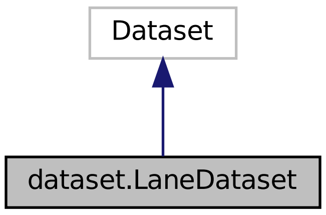
The documentation for this class was generated from the following file:

- train.py

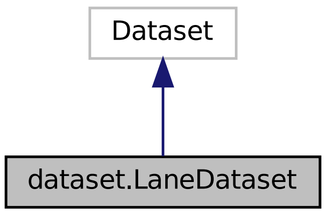
3.4 dataset.LaneDataset Class Reference

A custom dataset for lane detection.

Inheritance diagram for dataset.LaneDataset:



Collaboration diagram for dataset.LaneDataset:



Public Member Functions

- def **__init__** (self, image_dir, mask_dir, transform=None, num_augmentations=2)
- def __len__ (self)

  Returns the total number of samples in the dataset.
- def __getitem__ (self, index)

  Retrieves an image and its corresponding mask by index.

Public Attributes

- **image_dir**
- **mask_dir**
- **transform**
- **num_augmentations**
- **images**
- **total_samples**
- **base_transform**

3.4.1 Detailed Description

A custom dataset for lane detection.

This class loads images and their corresponding masks, applies optional data augmentations, and returns them as PyTorch tensors for training or validation.

Parameters

image_dir	(str): Directory containing the input images.
mask_dir	(str): Directory containing the ground truth masks.
transform	(albumentations.Compose, optional): Data augmentation pipeline (default: None).
num_augmentations	(int, optional): Number of augmented versions per image (default: 2).

3.4.2 Member Function Documentation

3.4.2.1 __getitem__()

```
def dataset.LaneDataset.__getitem__ (
              self,
              index )
```

Retrieves an image and its corresponding mask by index.

Loads an image and mask, applies augmentations if specified, and returns them as tensors.

Parameters

index	(int): Index of the sample to retrieve.

Returns

tuple: A tuple containing the image tensor ([C, H, W]) and mask tensor ([1, H, W]).

3.4.2.2 __len__()

```
def dataset.LaneDataset.__len__ (
              self )
```

Returns the total number of samples in the dataset.

Returns

int: Total number of samples, including augmentations.

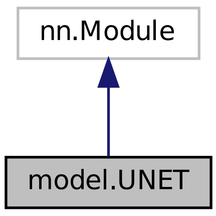
The documentation for this class was generated from the following file:

- dataset.py

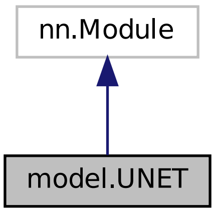
3.5 model.UNET Class Reference

U-Net architecture for image segmentation.

Inheritance diagram for model.UNET:



Collaboration diagram for model.UNET:



Public Member Functions

- def **__init__** (self, in_channels=3, out_channels=1, features=[32, 64, 128, 256])
- def forward (self, x)

  Forward pass of the U-Net model.
- def initialize_weights (self)

  Initializes the weights of convolutional and batch normalization layers.

Public Attributes

- **ups**
- **downs**
- **pool**
- **bottleneck**
- **final_conv**

3.5.1 Detailed Description

U-Net architecture for image segmentation.

This class implements the U-Net model with downsampling (encoder), bottleneck, and upsampling (decoder) paths, including skip connections for improved feature retention.

Parameters

in_channels	(int, optional): Number of input channels (default: 3).
out_channels	(int, optional): Number of output channels (default: 1).
features	(list, optional): List of feature sizes for each layer (default: [32, 64, 128, 256]).

3.5.2 Member Function Documentation

3.5.2.1 forward()

```
def model.UNET.forward (
              self,
              x )
```

Forward pass of the U-Net model.

Processes the input through the downsampling path, bottleneck, and upsampling path with skip connections.

Parameters

x	(torch.Tensor): Input tensor of shape [batch_size, in_channels, height, width].

Returns

torch.Tensor: Output tensor of shape [batch_size, out_channels, height, width].

3.5.2.2 initialize_weights()

```
def model.UNET.initialize_weights (
              self )
```

Initializes the weights of convolutional and batch normalization layers.

Uses Kaiming initialization for Conv2d and ConvTranspose2d layers, and constant initialization for BatchNorm2d layers.

The documentation for this class was generated from the following file:

- model.py

Index