TUTORIAL: Getting Started with Semantic Segmentation in PyTorch Using SMP

João Fernando Mari
joaofmari.github.io

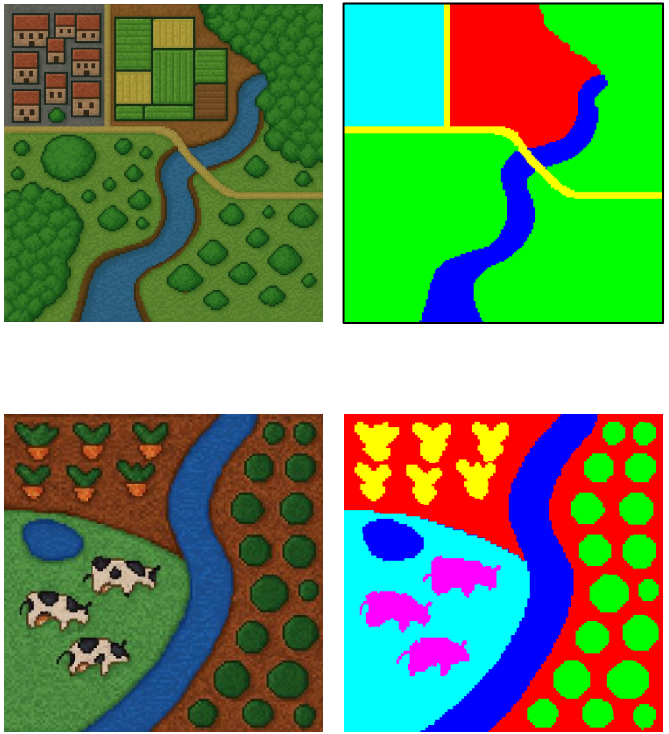
Leandro H. F. P. Silva
leandro.furtado@ufv.br

Mauricio Cunha Escarpinati
mauricio@ufu.br

André Ricardo Backes
arbackes@yahoo.com.br

Agenda

- Introduction
  - Semantic Segmentation
  - Tutorial Objectives
- Segmentation Models PyTorch – SMP
  - Installation
  - Models and Encoders
  - Reprodutibility
  - Datasets and DataLoaders
  - Loss Functions
  - Training/Validation Loop
  - Prediction Loop
  - Evaluation Metrics and Reduction Strategies
- Case Study I: Binary Segmentation
  - 38-Cloud Dataset
- Case Study II: Multiclass Segmentation
  - DeepGlobe Dataset
- Case Study III (Bonus): Multiclass Segmentation in a Grayscale Dataset
  - FUSAR-Map Dataset



INTRODUCTION

Team

UFU

Mauricio Escarpinati

UFV – Rio Paranaíba

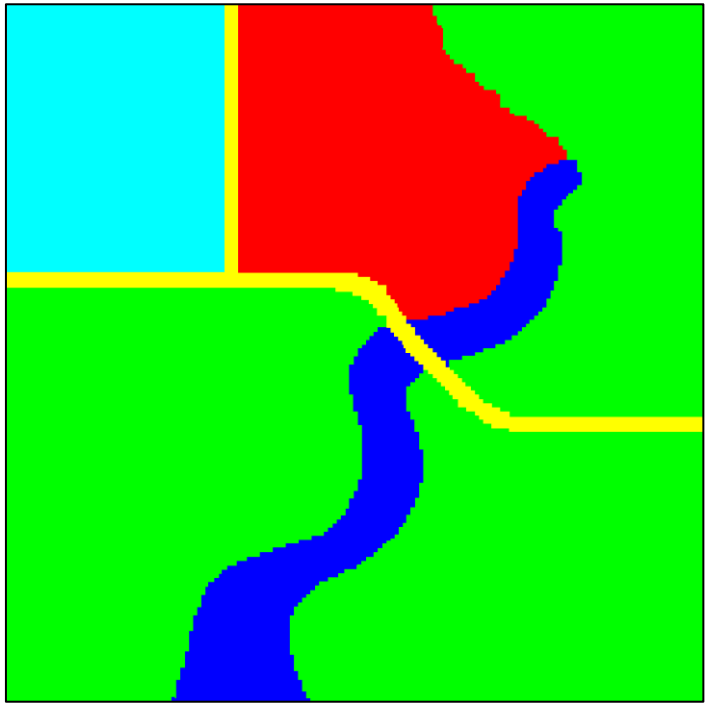
João F. Mari

Leandro Silva

UFSCar

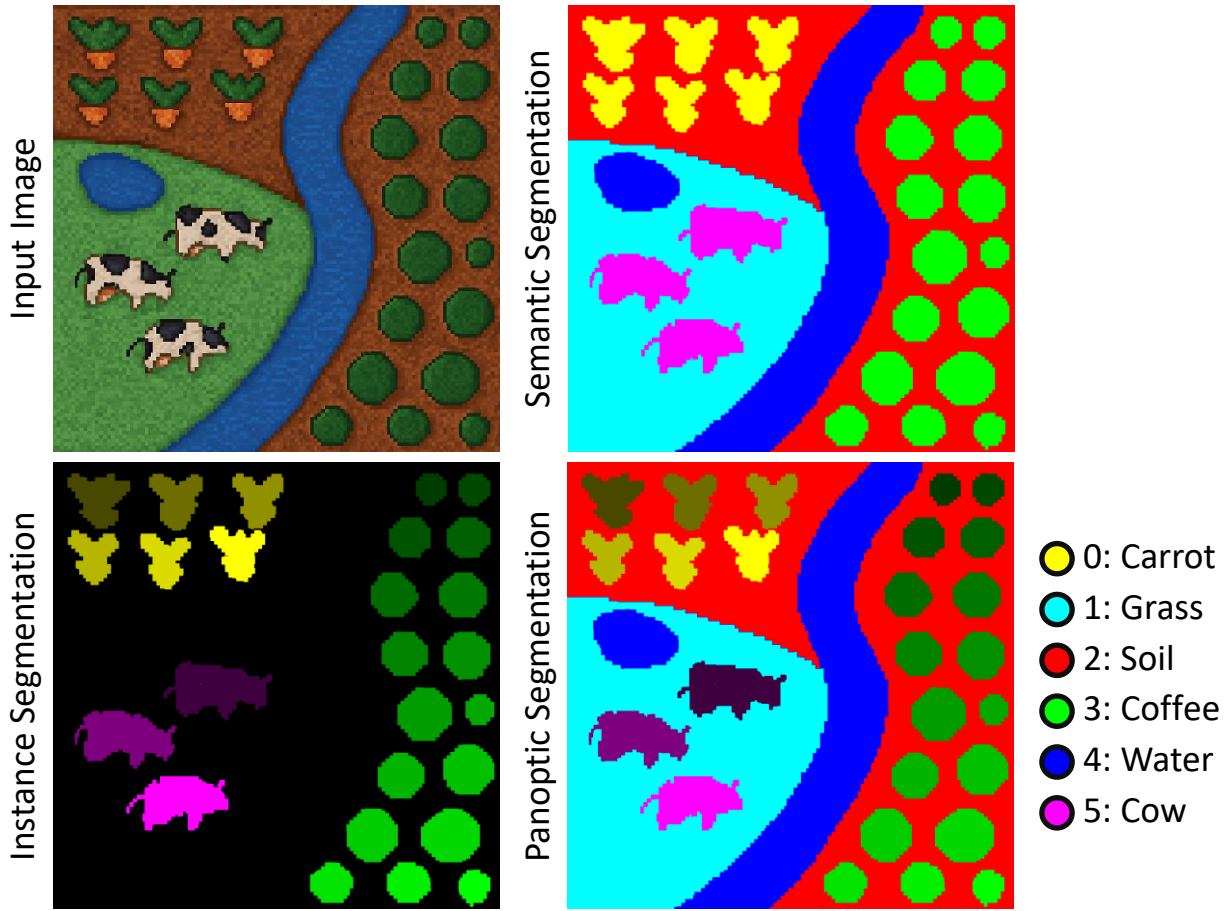
André Backes

Semantic Segmentation



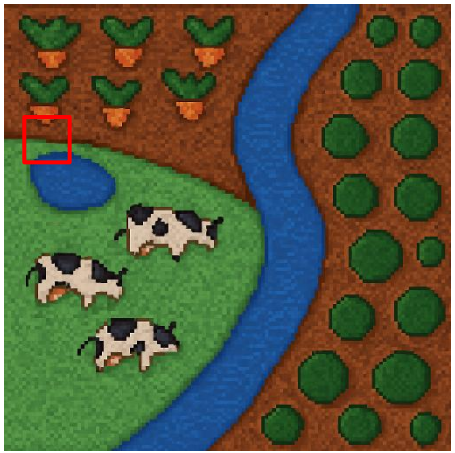
- 0: Urban Area
- 1: Agricultural Field
- 2: Native Vegetation
- 3: Water
- 4: Road

Semantic Vs. Instance Vs. Panoptic Segmentation

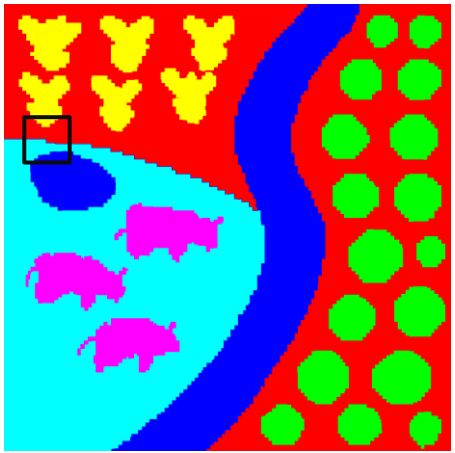


Classes Encoding

Input Image

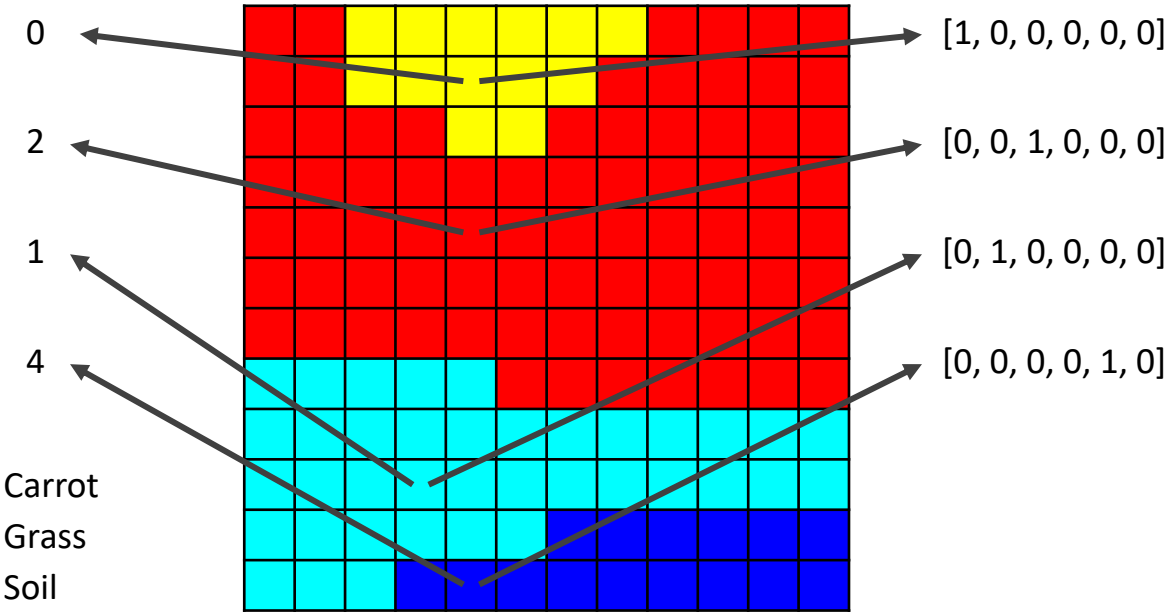


Panoptic Segmentation



Label Encoding:

One-Hot Encoding:



- 0: Carrot
- 1: Grass
- 2: Soil
- 3: Coffee
- 4: Water
- 5: Cow

Tutorial Objectives

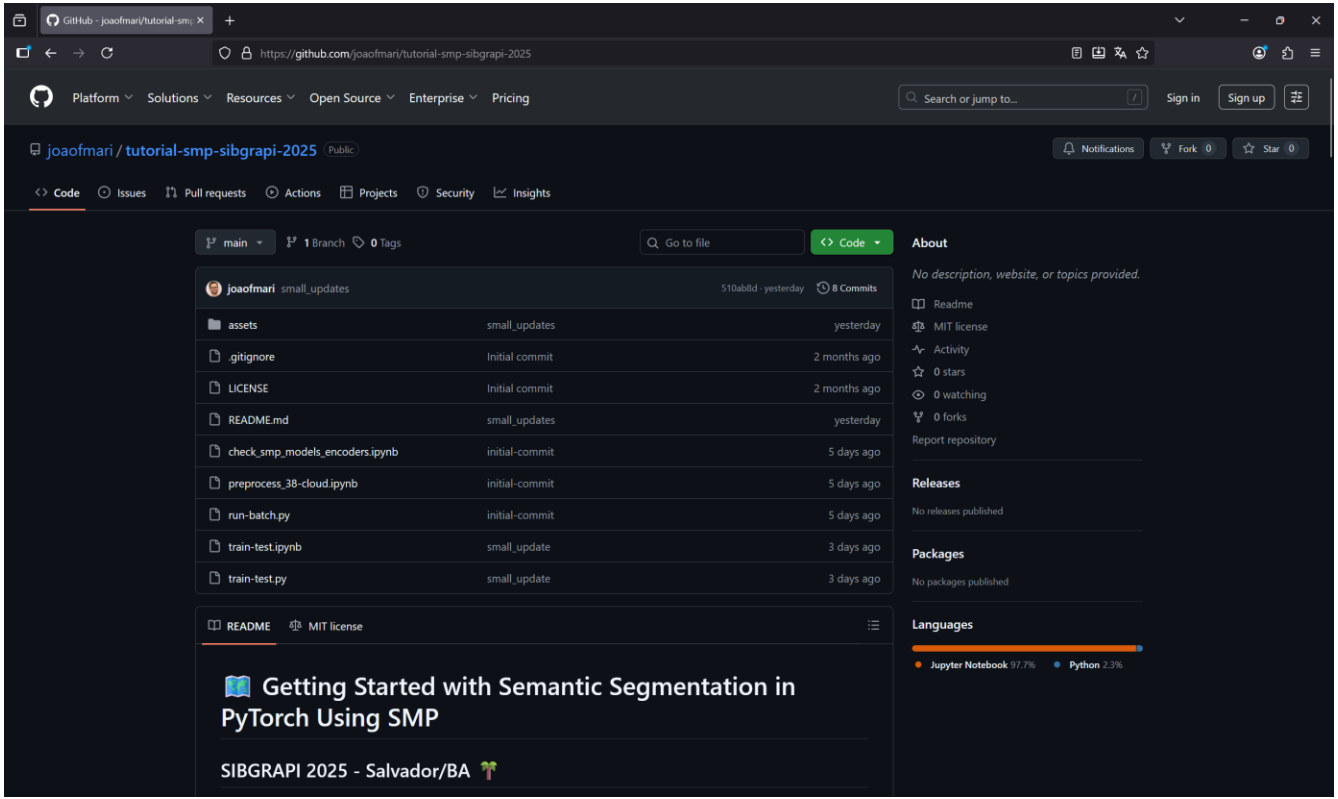
- 📚 Introduce semantic segmentation with the **Semantic Segmentation PyTorch (SMP)** library
- 🧩 Present main **architectures**, **pretrained encoders**, and the role of **transfer learning**
- ⚙️ Provide a **practical guide** for binary and multiclass experiments:
  - Minimal abstraction in loops (native PyTorch)
  - Built-in SMP losses and metrics for reproducibility
- 👥 Audience: beginners and practitioners seeking to learn SMP
- 🌍 Case studies:
  - 🛰️ Binary segmentation (multispectral images)
  - 🌈 Multiclass segmentation (RGB and grayscale datasets)

What This Tutorial Is (and Isn't)

- What **is** this tutorial for? ✅
  - A **hands-on guide** focused on practical experimentation.
  - Designing and running **segmentation projects** with SMP.
  - Setting up datasets, models, and training pipelines.
  - **Collecting**, **storing**, and **analyzing results** for reproducibility.
  - **Learning by doing**: minimal abstraction, maximum transparency.
- What **is not** this tutorial for ❌
  - A **theoretical introduction** to semantic segmentation.
  - Explaining in depth how **architectures** (e.g., U-Net, DeepLab) work internally.
  - Covering all mathematical foundations behind loss functions or metrics.
  - Replacing textbooks or survey papers on **segmentation concepts**.

GitHub Repository

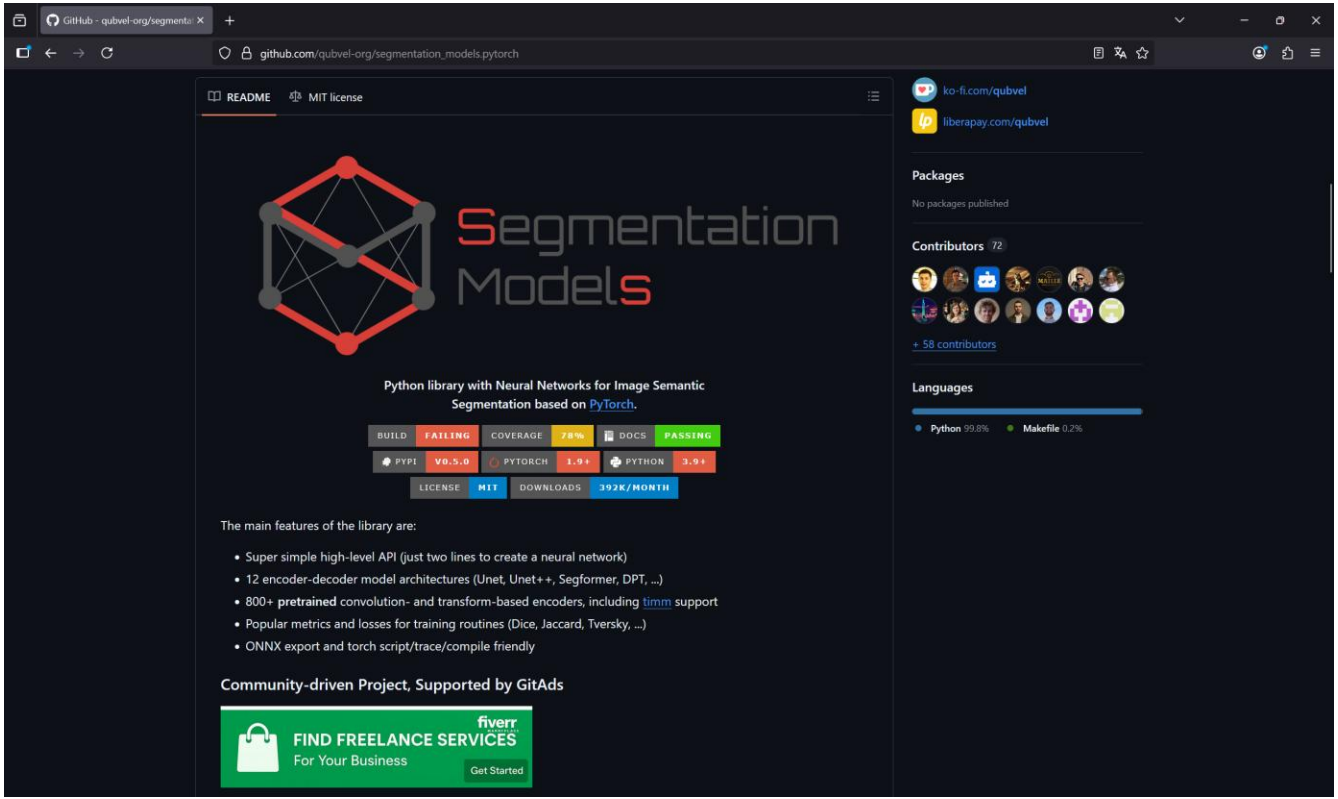
- Repository for this Tutorial: https://github.com/joaofmari/tutorial-smp-sibgrapi-2025



SEGMENTATION MODELS PYTORCH – SMP

Segmentation Models PyTorch – SMP

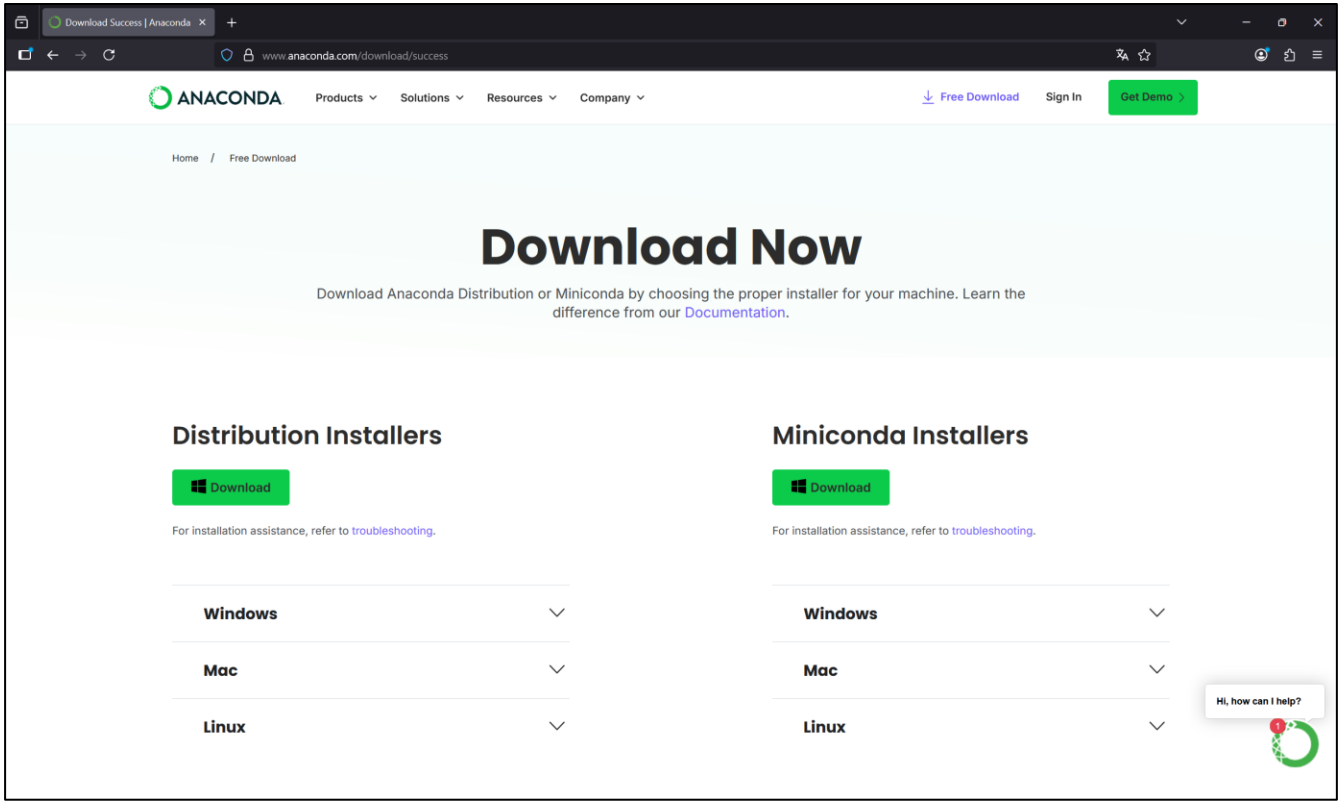
- https://github.com/qubvel-org/segmentation_models.pytorch



Version: 0.4.0

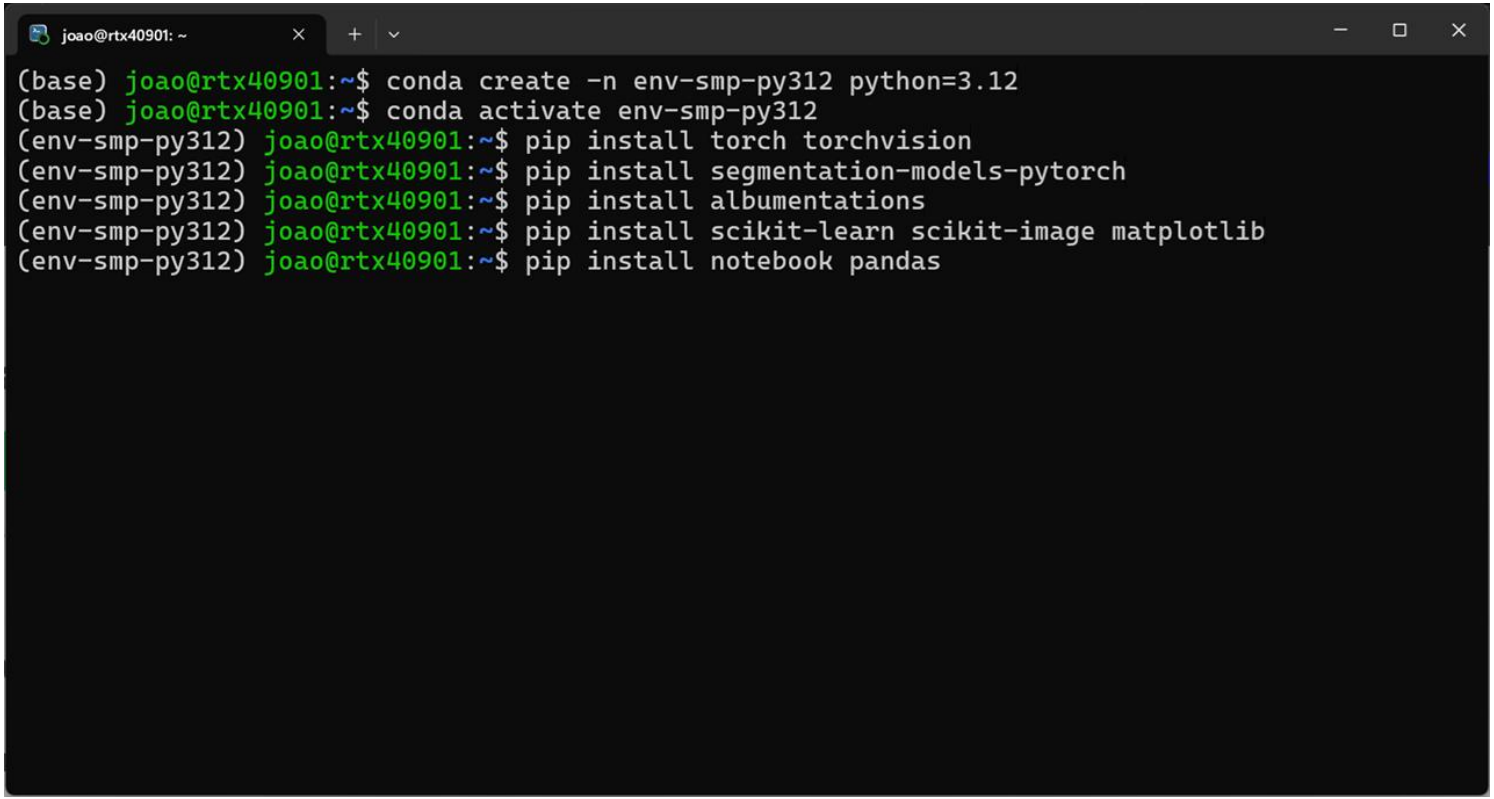
Installation

- https://www.anaconda.com/download



Installation

- Create a **Conda environment** and install the required libraries with **pip**.



Models and Encoders

#	Model	Description	Reference
1	Unet	U-Net: Encoder-decoder with skip connections	Ronneberger et al., 2015
2	Unet++	Nested U-Nets for improved feature fusion	Zhou et al., 2018
3	FPN	Feature Pyramid Network	Lin et al., 2017
4	PSPNet	Pyramid Scene Parsing Network	Zhao et al., 2017
5	DeepLabV3	DeepLabV3 with atrous spatial pyramid pooling	Chen et al., 2017
6	DeepLabV3+	DeepLabV3 with encoder-decoder refinement	Chen et al., 2018
7	Linknet	Lightweight encoder-decoder with residual blocks	Chaurasia et al., 2017
8	MAnet	Multi-scale Attention Network	Fan et al., 2020
9	PAN	Pyramid Attention Network	Li et al., 2018
10	UPerNet	Unified Perceptual Parsing Network	Xiao et al., 2018
11	Segformer	Segmentation Transformer	Xie et al., 2021
12	DPT	Dense Prediction Transformer	Ranftl et al., 2021

Models and Encoders

#	Encoder (Variant)	Pretrained Weights	Params (M)
1	ResNet-18	ImageNet / SSL / SWSL	11
2	ResNet-34	ImageNet	21
3	ResNet-50	ImageNet / SSL / SWSL	23
4	ResNet-101	ImageNet	42
5	ResNet-152	ImageNet	58
6	ResNeXt-50 32x4d	ImageNet / SSL / SWSL	22
7	ResNeXt-101 32x4d	SSL / SWSL	42
8	ResNeXt-101 32x8d	ImageNet / Instagram / SSL / SWSL	86
9	ResNeXt-101 32x16d	Instagram / SSL / SWSL	191
10	ResNeXt-101 32x32d	Instagram	466
11	ResNeXt-101 32x48d	Instagram	826
12	DPN-68	ImageNet	11
13	DPN-68b	ImageNet+5k	11
14	DPN-92	ImageNet+5k	34
15	DPN-98	ImageNet	58
16	DPN-107	ImageNet+5k	84
17	DPN-131	ImageNet	76
18	VGG-11 / 11_bn	ImageNet	9
19	VGG-13 / 13_bn	ImageNet	9
20	VGG-16 / 16_bn	ImageNet	14
21	VGG-19 / 19_bn	ImageNet	20
22	SENet-154	ImageNet	113
23	SE-ResNet50	ImageNet	26
24	SE-ResNet101	ImageNet	47
25	SE-ResNet152	ImageNet	64
26	SE-ResNeXt50_32x4d	ImageNet	25
27	SE-ResNeXt101_32x4d	ImageNet	46

#	Encoder (Variant)	Pretrained Weights	Params (M)
28	DenseNet-121	ImageNet	6
29	DenseNet-169	ImageNet	12
30	DenseNet-201	ImageNet	18
31	DenseNet-161	ImageNet	26
32	InceptionResNetV2	ImageNet / +Background	54
33	InceptionV4	ImageNet / +Background	41
34	EfficientNet-B0	ImageNet / AdvProp	4
35	EfficientNet-B1	ImageNet / AdvProp	6
36	EfficientNet-B2	ImageNet / AdvProp	7
37	EfficientNet-B3	ImageNet / AdvProp	10
38	EfficientNet-B4	ImageNet / AdvProp	17
39	EfficientNet-B5	ImageNet / AdvProp	28
40	EfficientNet-B6	ImageNet / AdvProp	40
41	EfficientNet-B7	ImageNet / AdvProp	63
42	MobileNet-V2	ImageNet	2
43	Mix Vision Transformer MIT-B0	ImageNet	3
44	Mix Vision Transformer MIT-B1	ImageNet	13
45	Mix Vision Transformer MIT-B2	ImageNet	24
46	Mix Vision Transformer MIT-B3	ImageNet	44
47	Mix Vision Transformer MIT-B4	ImageNet	60
48	Mix Vision Transformer MIT-B5	ImageNet	81
49	MobileOne-S0	ImageNet	4
50	MobileOne-S1	ImageNet	3
51	MobileOne-S2	ImageNet	5
52	MobileOne-S3	ImageNet	8
53	MobileOne-S4	ImageNet	12
54	Xception	ImageNet	20

Models and Encoders

#	Encoder (Variant)	Pretrained Weights	Params (M)
1	ResNet-18	ImageNet / SSL / SWSL	11
2	ResNet-34	ImageNet	21
3	ResNet-50	ImageNet / SSL / SWSL	23
4	ResNet-101	ImageNet	42
5	ResNet-152	ImageNet	58
6	ResNeXt-50 32x4d	ImageNet / SSL / SWSL	22
7	ResNeXt-101 32x4d	SSL / SWSL	42
8	ResNeXt-101 32x8d	ImageNet / Instagram / SSL / SWSL	86
9	ResNeXt-101 32x16d	Instagram / SSL / SWSL	191
10	ResNeXt-101 32x32d	Instagram	466
11	ResNeXt-101 32x48d	Instagram	826
12	DPN-68	ImageNet	11
13	DPN-68b	ImageNet+5k	11
14	DPN-92	ImageNet+5k	34
15	DPN-98	ImageNet	58
16	DPN-107	ImageNet+5k	84
17	DPN-131	ImageNet	76
18	VGG-11 / 11_bn	ImageNet	9
19	VGG-13 / 13_bn	ImageNet	9
20	VGG-16 / 16_bn	ImageNet	14
21	VGG-19 / 19_bn	ImageNet	20
22	SENet-154	ImageNet	113
23	SE-ResNet50	ImageNet	26
24	SE-ResNet101	ImageNet	47
25	SE-ResNet152	ImageNet	64
26	SE-ResNeXt50_32x4d	ImageNet	25
27	SE-ResNeXt101_32x4d	ImageNet	46

#	Encoder (Variant)	Pretrained Weights	Params (M)
28	DenseNet-121	ImageNet	6
29	DenseNet-169	ImageNet	12
30	DenseNet-201	ImageNet	18
31	DenseNet-161	ImageNet	26
32	InceptionResNetV2	ImageNet / +Background	54
33	InceptionV4	ImageNet / +Background	41
34	EfficientNet-B0	ImageNet / AdvProp	4
35	EfficientNet-B1	ImageNet / AdvProp	6
36	EfficientNet-B2	ImageNet / AdvProp	7
37	EfficientNet-B3	ImageNet / AdvProp	10
38	EfficientNet-B4	ImageNet / AdvProp	17
39	EfficientNet-B5	ImageNet / AdvProp	28
40	EfficientNet-B6	ImageNet / AdvProp	40
41	EfficientNet-B7	ImageNet / AdvProp	63
42	MobileNet-V2	ImageNet	2
43	Mix Vision Transformer MIT-B0	ImageNet	3
44	Mix Vision Transformer MIT-B1	ImageNet	13
45	Mix Vision Transformer MIT-B2	ImageNet	24
46	Mix Vision Transformer MIT-B3	ImageNet	44
47	Mix Vision Transformer MIT-B4	ImageNet	60
48	Mix Vision Transformer MIT-B5	ImageNet	81
49	MobileOne-S0	ImageNet	4
50	MobileOne-S1	ImageNet	3
51	MobileOne-S2	ImageNet	5
52	MobileOne-S3	ImageNet	8
53	MobileOne-S4	ImageNet	12
54	Xception	ImageNet	20

Models and Encoders Compatibility

#	Model	Encoder	Incompatibility
1	UnetPlusPlus	mit_b0, mit_b1, mit_b2, mit_b3, mit_b4, mit_b5	UnetPlusPlus is not support encoder_name=mit_b0
2	Linknet	mit_b0, mit_b1, mit_b2, mit_b3, mit_b4, mit_b5	Encoder `mit_b0` is not supported for Linknet
3	PAN	densenet121, densenet161, densenet169, densenet201	DenseNet encoders do not support dilated mode due to pooling operation for downsampling!
4	PAN	inceptionresnetv2	InceptionResnetV2 encoder does not support dilated mode due to pooling operation for downsampling!
5	PAN	inceptionv4	InceptionV4 encoder does not support dilated mode due to pooling operation for downsampling!
6	PAN	vgg11, vgg11_bn, vgg13, vgg13_bn vgg16, vgg16_bn, vgg19, vgg19_bn	'VGG' models do not support dilated mode due to Max Pooling operations for downsampling!
7	PAN	xception	Xception encoder does not support dilated mode due to pooling operation for downsampling!
8	DeepLabV3	densenet121, densenet161, densenet169, densenet201	DenseNet encoders do not support dilated mode due to pooling operation for downsampling!
9	DeepLabV3	inceptionresnetv2	InceptionResnetV2 encoder does not support dilated mode due to pooling operation for downsampling!
10	DeepLabV3	inceptionv4	InceptionV4 encoder does not support dilated mode due to pooling operation for downsampling!
11	DeepLabV3	vgg11, vgg11_bn, vgg13, vgg13_bn vgg16, vgg16_bn, vgg19, vgg19_bn	'VGG' models do not support dilated mode due to Max Pooling operations for downsampling!
12	DeepLabV3	xception	Xception encoder does not support dilated mode due to pooling operation for downsampling!
13	DeepLabV3Plus	densenet121, densenet161, densenet169, densenet201	DenseNet encoders do not support dilated mode due to pooling operation for downsampling!
14	DeepLabV3Plus	inceptionresnetv2	InceptionResnetV2 encoder does not support dilated mode due to pooling operation for downsampling!
15	DeepLabV3Plus	inceptionv4	InceptionV4 encoder does not support dilated mode due to pooling operation for downsampling!
16	DeepLabV3Plus	vgg11, vgg11_bn, vgg13, vgg13_bn vgg16, vgg16_bn, vgg19, vgg19_bn	'VGG' models do not support dilated mode due to Max Pooling operations for downsampling!
17	DeepLabV3Plus	xception	Xception encoder does not support dilated mode due to pooling operation for downsampling!

Pretrained Weights

#	Weights	Description	Mean	Stand. Dev
1	imagenet	Standard supervised weights trained on ImageNet classification (≈1K classes)	[0.485, 0.456, 0.406]	[0.229, 0.224, 0.225]
2	ssl	Self-supervised learning variants (trained without labels).	[0.485, 0.456, 0.406]	[0.229, 0.224, 0.225]
3	swsl	Semi-Weakly Supervised Learning weights, a mix of supervised and weak supervision.	[0.485, 0.456, 0.406]	[0.229, 0.224, 0.225]
4	instagram	Weights pretrained on large-scale Instagram images (unlabeled or hashtag metadata).	[0.485, 0.456, 0.406]	[0.229, 0.224, 0.225]
5	imagenet+5k	ImageNet trained with additional data, ImageNet 1K plus extra 5K categories.	[0.48627450980392156, 0.4588235294117647, 0.40784313725490196]	[0.23482446870963955, 0.23482446870963955, 0.23482446870963955]
6	imagenet+background	Possibly a variant of ImageNet+5K with background augmentation (not clearly documented)	[0.5, 0.5, 0.5],	[0.5, 0.5, 0.5]
7	advprop	Weights trained using Adversarial Propagation.	[0.5, 0.5, 0.5],	[0.5, 0.5, 0.5]

- For all pretrained weights `input_space='RGB'` and `input_range=[0, 1]`.

Models and Encoders


```
# Initialize model and encoder
model = smp.Unet(
    encoder_name="resnet50",
    encoder_weights="imagenet",
    in_channels=3,
    classes=7
)
```

Create model instance.

Define encoder.

Define pre-training weights.

Number of input channels.

Number of classes.

```
# Get preprocessing function for encoder
preprocessing_fn =
    smp.encoders.get_preprocessing_fn(
    "resnet50",
    "imagenet"
)
```

Define encoder.

Define pre-training weights.

Models and Encoders

- Unet + ResNet50

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
1.  Unet                                 [1, 7, 512, 512]          --
2.  ├─ResNetEncoder: 1-1                 [1, 3, 512, 512]          --
3.  │  └─Conv2d: 2-1                     [1, 64, 256, 256]         9,408
4.  │  └─BatchNorm2d: 2-2                [1, 64, 256, 256]         128
5.  │  └─ReLU: 2-3                       [1, 64, 256, 256]         --
6.  │  └─MaxPool2d: 2-4                  [1, 64, 128, 128]         --
7.  │  └─Sequential: 2-5                 [1, 256, 128, 128]        --
8.  │  │  └─Bottleneck: 3-1             [1, 256, 128, 128]        75,008
9.  │  │  └─Bottleneck: 3-2             [1, 256, 128, 128]        70,400
10. │  │  └─Bottleneck: 3-3             [1, 256, 128, 128]        70,400
11. │  └─Sequential: 2-6                 [1, 512, 64, 64]          --
12. │  │  └─Bottleneck: 3-4             [1, 512, 64, 64]          379,392
13. │  │  └─Bottleneck: 3-5             [1, 512, 64, 64]          280,064
14. │  │  └─Bottleneck: 3-6             [1, 512, 64, 64]          280,064
15. │  │  └─Bottleneck: 3-7             [1, 512, 64, 64]          280,064
16. │  └─Sequential: 2-7                 [1, 1024, 32, 32]         --
17. │  │  └─Bottleneck: 3-8             [1, 1024, 32, 32]         1,512,448
18. │  │  └─Bottleneck: 3-9             [1, 1024, 32, 32]         1,117,184
19. │  │  └─Bottleneck: 3-10            [1, 1024, 32, 32]         1,117,184
20. │  │  └─Bottleneck: 3-11            [1, 1024, 32, 32]         1,117,184
```

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
21. │  │  └─Bottleneck: 3-12            [1, 1024, 32, 32]         1,117,184
22. │  │  └─Bottleneck: 3-13            [1, 1024, 32, 32]         1,117,184
23. │  └─Sequential: 2-8                 [1, 2048, 16, 16]         --
24. │  │  └─Bottleneck: 3-14            [1, 2048, 16, 16]         6,039,552
25. │  │  └─Bottleneck: 3-15            [1, 2048, 16, 16]         4,462,592
26. │  │  └─Bottleneck: 3-16            [1, 2048, 16, 16]         4,462,592
27. ├─UnetDecoder: 1-2                   [1, 16, 512, 512]         --
28. │  └─Identity: 2-9                   [1, 2048, 16, 16]         --
29. │  └─ModuleList: 2-10                --                        --
30. │  │  └─UnetDecoderBlock: 3-17       [1, 256, 32, 32]          7,668,736
31. │  │  └─UnetDecoderBlock: 3-18       [1, 128, 64, 64]          1,032,704
32. │  │  └─UnetDecoderBlock: 3-19       [1, 64, 128, 128]         258,304
33. │  │  └─UnetDecoderBlock: 3-20       [1, 32, 256, 256]         46,208
34. │  │  └─UnetDecoderBlock: 3-21       [1, 16, 512, 512]         6,976
35. ├─SegmentationHead: 1-3              [1, 7, 512, 512]          --
36. │  └─Conv2d: 2-11                    [1, 7, 512, 512]          1,015
37. │  └─Identity: 2-12                  [1, 7, 512, 512]          --
38. │  └─Activation: 2-13                [1, 7, 512, 512]          --
39. │  │  └─Identity: 3-22               [1, 7, 512, 512]          --
```

Models and Encoders

- Unet + EfficientNet-B2

	Layer (type:depth-idx)	Output Shape	Param #
1.	Unet	[1, 7, 512, 512]	--
2.	├─EfficientNetEncoder: 1-1	[1, 3, 512, 512]	498,432
3.	│ └─Conv2dStaticSamePadding: 2-1	[1, 32, 256, 256]	864
4.	│ │ └─ZeroPad2d: 3-1	[1, 3, 513, 513]	--
5.	│ └─BatchNorm2d: 2-2	[1, 32, 256, 256]	64
6.	│ └─SiLU: 2-3	[1, 32, 256, 256]	--
7.	│ └─ModuleList: 2-4	--	--
8.	│ │ └─MBConvBlock: 3-2	[1, 16, 256, 256]	1,448
9.	│ │ └─MBConvBlock: 3-3	[1, 16, 256, 256]	612
10.	│ │ └─MBConvBlock: 3-4	[1, 24, 128, 128]	6,004
11.	│ │ └─MBConvBlock: 3-5	[1, 24, 128, 128]	10,710
12.	│ │ └─MBConvBlock: 3-6	[1, 24, 128, 128]	10,710
13.	│ │ └─MBConvBlock: 3-7	[1, 48, 64, 64]	16,518
14.	│ │ └─MBConvBlock: 3-8	[1, 48, 64, 64]	43,308
15.	│ │ └─MBConvBlock: 3-9	[1, 48, 64, 64]	43,308
16.	│ │ └─MBConvBlock: 3-10	[1, 88, 32, 32]	50,300
17.	│ │ └─MBConvBlock: 3-11	[1, 88, 32, 32]	123,750
18.	│ │ └─MBConvBlock: 3-12	[1, 88, 32, 32]	123,750
19.	│ │ └─MBConvBlock: 3-13	[1, 88, 32, 32]	123,750
20.	│ │ └─MBConvBlock: 3-14	[1, 120, 32, 32]	149,158
21.	│ │ └─MBConvBlock: 3-15	[1, 120, 32, 32]	237,870
22.	│ │ └─MBConvBlock: 3-16	[1, 120, 32, 32]	237,870
23.	│ │ └─MBConvBlock: 3-17	[1, 120, 32, 32]	237,870
24.	│ │ └─MBConvBlock: 3-18	[1, 208, 16, 16]	301,406
25.	│ │ └─MBConvBlock: 3-19	[1, 208, 16, 16]	686,868
26.	│ │ └─MBConvBlock: 3-20	[1, 208, 16, 16]	686,868
27.	│ │ └─MBConvBlock: 3-21	[1, 208, 16, 16]	686,868
28.	│ │ └─MBConvBlock: 3-22	[1, 208, 16, 16]	686,868
29.	│ │ └─MBConvBlock: 3-23	[1, 352, 16, 16]	846,900
30.	│ │ └─MBConvBlock: 3-24	[1, 352, 16, 16]	1,888,920
31.	├─UnetDecoder: 1-2	[1, 16, 512, 512]	--
32.	│ └─Identity: 2-5	[1, 352, 16, 16]	--
33.	│ └─ModuleList: 2-6	--	--
34.	│ │ └─UnetDecoderBlock: 3-25	[1, 256, 32, 32]	1,678,336
35.	│ │ └─UnetDecoderBlock: 3-26	[1, 128, 64, 64]	498,176
36.	│ │ └─UnetDecoderBlock: 3-27	[1, 64, 128, 128]	124,672
37.	│ │ └─UnetDecoderBlock: 3-28	[1, 32, 256, 256]	36,992
38.	│ │ └─UnetDecoderBlock: 3-29	[1, 16, 512, 512]	6,976
39.	├─SegmentationHead: 1-3	[1, 7, 512, 512]	--
40.	│ └─Conv2d: 2-7	[1, 7, 512, 512]	1,015
41.	│ └─Identity: 2-8	[1, 7, 512, 512]	--
42.	│ └─Activation: 2-9	[1, 7, 512, 512]	--
43.	│ │ └─Identity: 3-30	[1, 7, 512, 512]	--

Models and Encoders

- FPN + ResNet50

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
1.  FPN                                  [1, 7, 512, 512]          --
2.  ├─ResNetEncoder: 1-1                 [1, 3, 512, 512]          --
3.  │  └─Conv2d: 2-1                     [1, 64, 256, 256]         9,408
4.  │  └─BatchNorm2d: 2-2                [1, 64, 256, 256]         128
5.  │  └─ReLU: 2-3                       [1, 64, 256, 256]         --
6.  │  └─MaxPool2d: 2-4                  [1, 64, 128, 128]         --
7.  │  └─Sequential: 2-5                 [1, 256, 128, 128]        --
8.  │  │  └─Bottleneck: 3-1             [1, 256, 128, 128]        75,008
9.  │  │  └─Bottleneck: 3-2             [1, 256, 128, 128]        70,400
10. │  │  └─Bottleneck: 3-3             [1, 256, 128, 128]        70,400
11. │  └─Sequential: 2-6                 [1, 512, 64, 64]          --
12. │  │  └─Bottleneck: 3-4             [1, 512, 64, 64]          379,392
13. │  │  └─Bottleneck: 3-5             [1, 512, 64, 64]          280,064
14. │  │  └─Bottleneck: 3-6             [1, 512, 64, 64]          280,064
15. │  │  └─Bottleneck: 3-7             [1, 512, 64, 64]          280,064
16. │  └─Sequential: 2-7                 [1, 1024, 32, 32]         --
17. │  │  └─Bottleneck: 3-8             [1, 1024, 32, 32]         1,512,448
18. │  │  └─Bottleneck: 3-9             [1, 1024, 32, 32]         1,117,184
19. │  │  └─Bottleneck: 3-10            [1, 1024, 32, 32]         1,117,184
20. │  │  └─Bottleneck: 3-11            [1, 1024, 32, 32]         1,117,184
21. │  │  └─Bottleneck: 3-12            [1, 1024, 32, 32]         1,117,184
22. │  │  └─Bottleneck: 3-13            [1, 1024, 32, 32]         1,117,184
23. │  └─Sequential: 2-8                 [1, 2048, 16, 16]         --
```

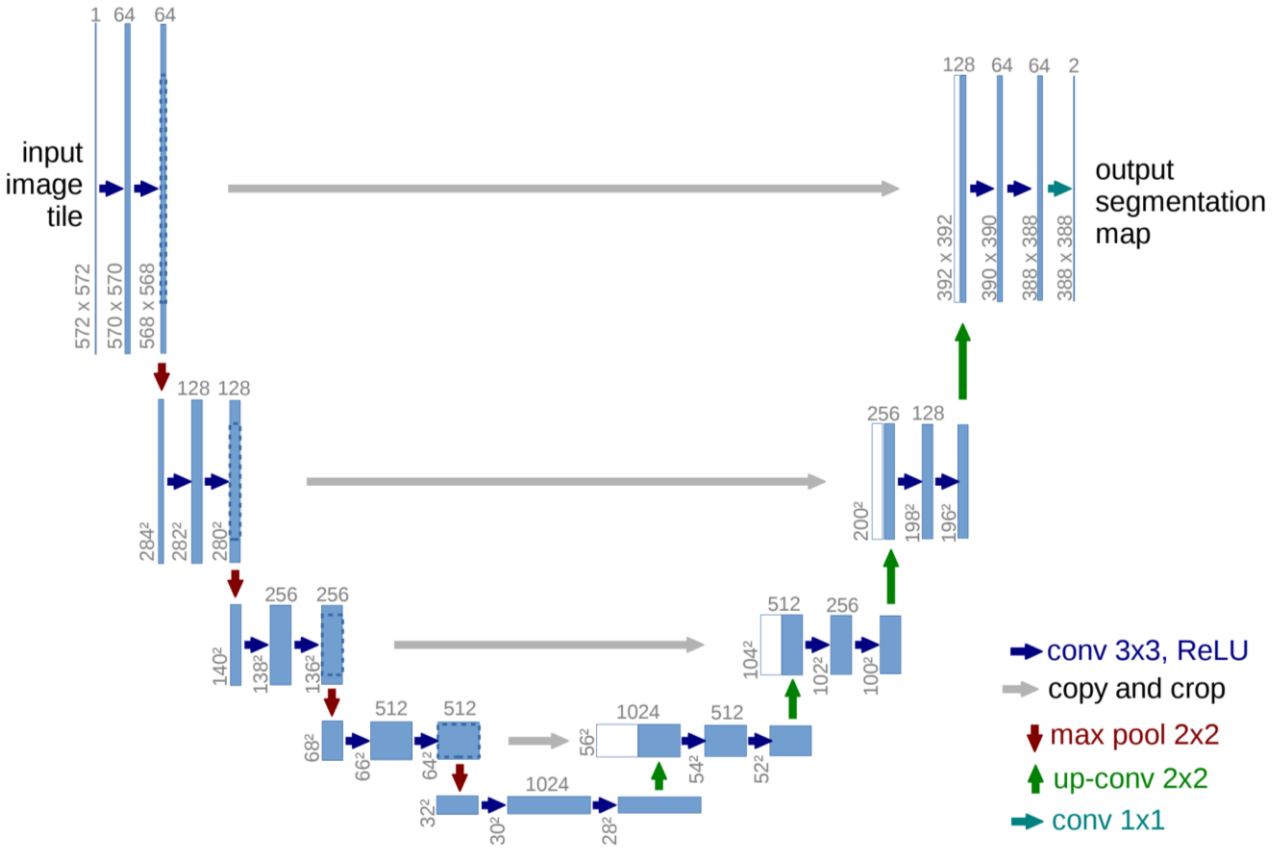
```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
24. │  │  └─Bottleneck: 3-14            [1, 2048, 16, 16]         6,039,552
25. │  │  └─Bottleneck: 3-15            [1, 2048, 16, 16]         4,462,592
26. │  │  └─Bottleneck: 3-16            [1, 2048, 16, 16]         4,462,592
27. ├─FPNDecoder: 1-2                    [1, 128, 128, 128]        --
28. │  └─Conv2d: 2-9                     [1, 256, 16, 16]          524,544
29. │  └─FPNBlock: 2-10                  [1, 256, 32, 32]          --
30. │  │  └─Conv2d: 3-17                [1, 256, 32, 32]          262,400
31. │  └─FPNBlock: 2-11                  [1, 256, 64, 64]          --
32. │  │  └─Conv2d: 3-18                [1, 256, 64, 64]          131,328
33. │  └─FPNBlock: 2-12                  [1, 256, 128, 128]        --
34. │  │  └─Conv2d: 3-19                [1, 256, 128, 128]        65,792
35. │  └─ModuleList: 2-13                --                        --
36. │  │  └─SegmentationBlock: 3-20     [1, 128, 128, 128]        590,592
37. │  │  └─SegmentationBlock: 3-21     [1, 128, 128, 128]        442,880
38. │  │  └─SegmentationBlock: 3-22     [1, 128, 128, 128]        295,168
39. │  │  └─SegmentationBlock: 3-23     [1, 128, 128, 128]        295,168
40. │  └─MergeBlock: 2-14                [1, 128, 128, 128]        --
41. │  └─Dropout2d: 2-15                 [1, 128, 128, 128]        --
42. ├─SegmentationHead: 1-3              [1, 7, 512, 512]          --
43. │  └─Conv2d: 2-16                    [1, 7, 128, 128]          903
44. │  └─UpsamplingBilinear2d: 2-17      [1, 7, 512, 512]          --
45. │  └─Activation: 2-18                [1, 7, 512, 512]          --
46. │  │  └─Identity: 3-24               [1, 7, 512, 512]          --
```

Models and Encoders

- FPN + EfficientNet-B2

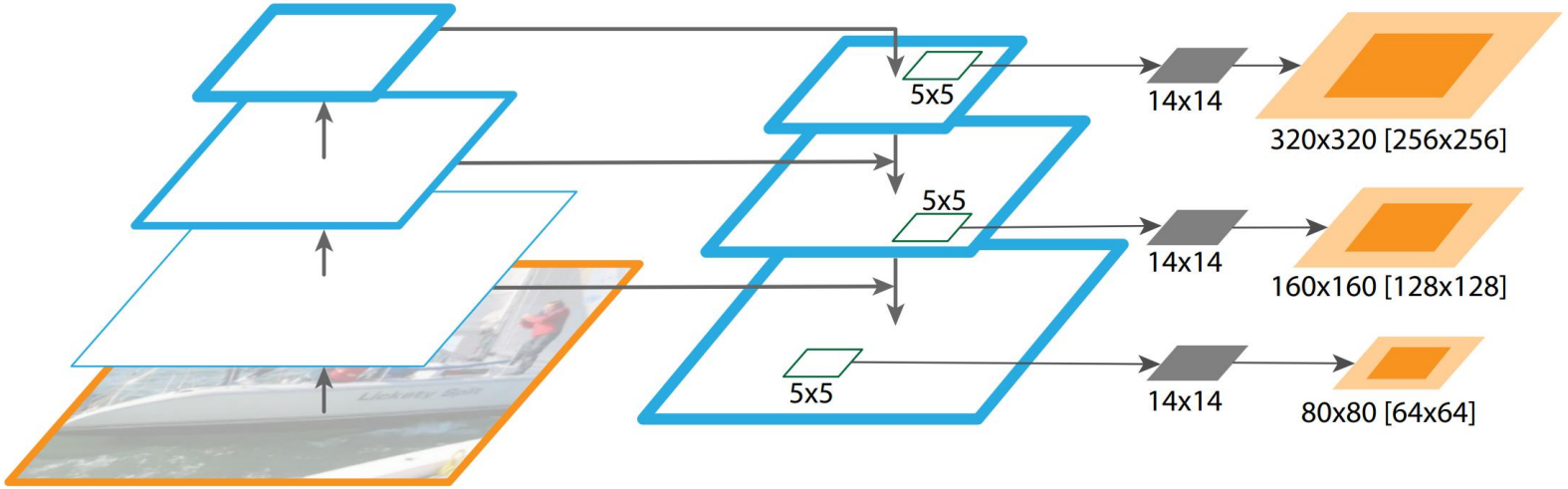
	Layer (type:depth-idx)	Output Shape	Param #
1.	FPN	[1, 7, 512, 512]	--
2.	├─EfficientNetEncoder: 1-1	[1, 3, 512, 512]	498,432
3.	│ └─Conv2dStaticSamePadding: 2-1	[1, 32, 256, 256]	864
4.	│ │ └─ZeroPad2d: 3-1	[1, 3, 513, 513]	--
5.	│ └─BatchNorm2d: 2-2	[1, 32, 256, 256]	64
6.	│ └─SiLU: 2-3	[1, 32, 256, 256]	--
7.	│ └─ModuleList: 2-4	--	--
8.	│ │ └─MBConvBlock: 3-2	[1, 16, 256, 256]	1,448
9.	│ │ └─MBConvBlock: 3-3	[1, 16, 256, 256]	612
10.	│ │ └─MBConvBlock: 3-4	[1, 24, 128, 128]	6,004
11.	│ │ └─MBConvBlock: 3-5	[1, 24, 128, 128]	10,710
12.	│ │ └─MBConvBlock: 3-6	[1, 24, 128, 128]	10,710
13.	│ │ └─MBConvBlock: 3-7	[1, 48, 64, 64]	16,518
14.	│ │ └─MBConvBlock: 3-8	[1, 48, 64, 64]	43,308
15.	│ │ └─MBConvBlock: 3-9	[1, 48, 64, 64]	43,308
16.	│ │ └─MBConvBlock: 3-10	[1, 88, 32, 32]	50,300
17.	│ │ └─MBConvBlock: 3-11	[1, 88, 32, 32]	123,750
18.	│ │ └─MBConvBlock: 3-12	[1, 88, 32, 32]	123,750
19.	│ │ └─MBConvBlock: 3-13	[1, 88, 32, 32]	123,750
20.	│ │ └─MBConvBlock: 3-14	[1, 120, 32, 32]	149,158
21.	│ │ └─MBConvBlock: 3-15	[1, 120, 32, 32]	237,870
22.	│ │ └─MBConvBlock: 3-16	[1, 120, 32, 32]	237,870
23.	│ │ └─MBConvBlock: 3-17	[1, 120, 32, 32]	237,870
24.	│ │ └─MBConvBlock: 3-18	[1, 208, 16, 16]	301,406
25.	│ │ └─MBConvBlock: 3-19	[1, 208, 16, 16]	686,868
26.	│ │ └─MBConvBlock: 3-20	[1, 208, 16, 16]	686,868
27.	│ │ └─MBConvBlock: 3-21	[1, 208, 16, 16]	686,868
28.	│ │ └─MBConvBlock: 3-22	[1, 208, 16, 16]	686,868
29.	│ │ └─MBConvBlock: 3-23	[1, 352, 16, 16]	846,900
30.	│ │ └─MBConvBlock: 3-24	[1, 352, 16, 16]	1,888,920
31.	├─FPNDecoder: 1-2	[1, 128, 128, 128]	--
32.	│ └─Conv2d: 2-5	[1, 256, 16, 16]	90,368
33.	│ └─FPNBlock: 2-6	[1, 256, 32, 32]	--
34.	│ │ └─Conv2d: 3-25	[1, 256, 32, 32]	30,976
35.	│ └─FPNBlock: 2-7	[1, 256, 64, 64]	--
36.	│ │ └─Conv2d: 3-26	[1, 256, 64, 64]	12,544
37.	│ └─FPNBlock: 2-8	[1, 256, 128, 128]	--
38.	│ │ └─Conv2d: 3-27	[1, 256, 128, 128]	6,400
39.	│ └─ModuleList: 2-9	--	--
40.	│ │ └─SegmentationBlock: 3-28	[1, 128, 128, 128]	590,592
41.	│ │ └─SegmentationBlock: 3-29	[1, 128, 128, 128]	442,880
42.	│ │ └─SegmentationBlock: 3-30	[1, 128, 128, 128]	295,168
43.	│ │ └─SegmentationBlock: 3-31	[1, 128, 128, 128]	295,168
44.	│ └─MergeBlock: 2-10	[1, 128, 128, 128]	--
45.	│ └─Dropout2d: 2-11	[1, 128, 128, 128]	--
46.	├─SegmentationHead: 1-3	[1, 7, 512, 512]	--
47.	│ └─Conv2d: 2-12	[1, 7, 128, 128]	903
48.	│ └─UpsamplingBilinear2d: 2-13	[1, 7, 512, 512]	--
49.	│ └─Activation: 2-14	[1, 7, 512, 512]	--
50.	│ │ └─Identity: 3-32	[1, 7, 512, 512]	--

U-Net



Ronneberger et al. "**U-net: Convolutional networks for biomedical image segmentation**" MICCAI, 2015

FPN



Lin, et al. **“Feature pyramid networks for object detection”** CVPR, 2017.

Reprodutibility

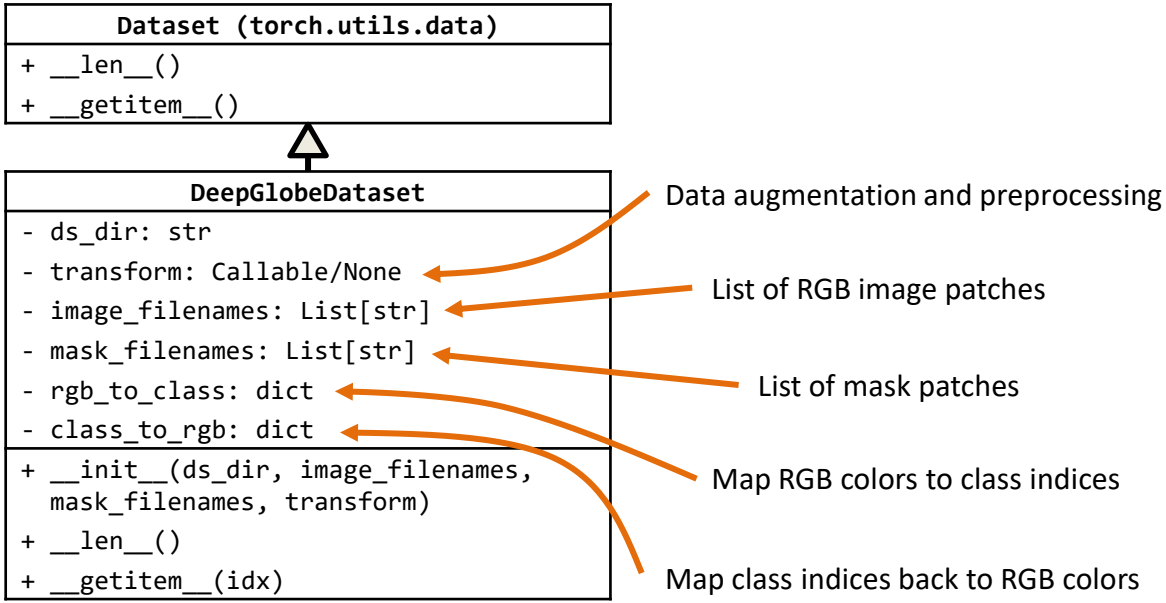
- **Set random seeds** across Python, NumPy, and PyTorch for reproducibility.
- **Benefit:** ensures results can be replicated.
- **Trade-off:** may slightly reduce training performance.
- ⚠️ **Important:** Some models (e.g., *PSPNet in SMP*) require non-deterministic operations.
  - They may **fail** if you enforce:

`torch.use_deterministic_algorithms(True)`

```python
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    # If you use multi-GPU
    torch.cuda.manual_seed_all(seed)

    os.environ["PYTHONHASHSEED"] = str(seed)
    # For CUDA >= 10.2
    os.environ["CUBLAS_WORKSPACE_CONFIG"] = ":4096:8"

    torch.backends.cudnn.benchmark = False
    torch.backends.cudnn.deterministic = True

    torch.use_deterministic_algorithms(
        True,
        warn_only=True
    )
```

Datasets and DataLoaders

- Multiclass dataset with RGB images:
  - DeepGlobe Dataset




```
~/
└── Datasets/
    └── DeepGlobe/
        ├── test/
        ├── train/
        │   ├── 100694_mask.png
        │   ├── 100694_sat.jpg
        │   ├── ...
        │   ├── 100694_mask.png
        │   └── 100694_sat.jpg
        ├── valid/
        ├── class_dict.csv
        ├── metadata.csv
        └── archive.zip
```

6399_sat.jpg

6399_mask.png

Datasets and DataLoaders

DeepGlobe Dataset:

```python
class DeepGlobeDataset(Dataset):

    def __init__(self, ds_dir, image_filenames, mask_filenames, transform=None):
        self.ds_dir = ds_dir
        self.transform = transform
        self.image_filenames = image_filenames
        self.mask_filenames = mask_filenames

        # Define RGB-to-class index mapping
        self.rgb_to_class = {
            (0, 255, 255): 0,    # Urban land
            (255, 255, 0): 1,    # Agriculture land
            (255, 0, 255): 2,    # Rangeland
            (0, 255, 0): 3,      # Forest land
            (0, 0, 255): 4,      # Water
            (255, 255, 255): 5,  # Barren land
            (0, 0, 0): 6         # Unknown (Clouds & others)
        }

        # Inverse mapping: class index to RGB (for visualization)
        self.class_to_rgb = {v: k for k, v in self.rgb_to_class.items()}

    def __len__(self):
        """Returns the total number of samples in the dataset."""
        return len(self.image_filenames)

```

1/3

Datasets and DataLoaders

DeepGlobe Dataset:

```
18.         }
19.
20.         # Inverse mapping: class index to RGB (for visualization)
21.         self.class_to_rgb = {v: k for k, v in self.rgb_to_class.items()}
22.
23.     def __len__(self):
24.         """Returns the total number of samples in the dataset."""
25.         return len(self.image_filenames)
26.
27.     def __getitem__(self, idx):
28.         img_name = self.image_filenames[idx]
29.         mask_name = self.mask_filenames[idx]
30.
31.         # Check if the image and mask filenames share the same prefix
32.         if img_name.split('_')[0] != mask_name.split('_')[0]:
33.             print(f"Warning! The image and mask files do not match! {img_name} != {mask_name}")
34.
35.         img_path = os.path.join(self.ds_dir, img_name)
36.         mask_path = os.path.join(self.ds_dir, mask_name)
37.
38.         # Load image and mask (as RGB)
39.         image = np.array(Image.open(img_path).convert("RGB"))  # Ensure 3-channel RGB
40.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
41.
42.         # Convert RGB mask to class index mask
43.         mask = np.zeros((mask_rgb.shape[0], mask_rgb.shape[1]), dtype=np.uint8)  #
```

2/3

Datasets and DataLoaders

DeepGlobe Dataset:

```
18.         }
19.
20.         # Inverse mapping: class index to RGB (for visualization)
21.         self.class_to_rgb = {v: k for k, v in self.rgb_to_class.items()}
22.
23.     def __len__(self):
24.         """Returns the total number of samples in the dataset."""
25.         return len(self.image_filenames)
26.
27.     def __getitem__(self, idx):
28.         img_name = self.image_filenames[idx]
29.         mask_name = self.mask_filenames[idx]
30.
31.         # Check if the image and mask filenames share the same prefix
32.         if img_name.split('_')[0] != mask_name.split('_')[0]:
33.             print(f"Warning! The image and mask files do not match! {img_name} != {mask_name}")
34.
35.         img_path = os.path.join(self.ds_dir, img_name)
36.         mask_path = os.path.join(self.ds_dir, mask_name)
37.
38.         # Load image and mask (as RGB)
39.         image = np.array(Image.open(img_path).convert("RGB"))  # Ensure 3-channel RGB
40.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
41.
42.         # Convert RGB mask to class index mask
43.         mask = np.zeros((mask_rgb.shape[0], mask_rgb.shape[1]), dtype=np.uint8)  #
```

2/3

Datasets and DataLoaders

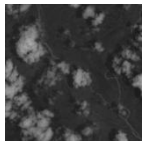
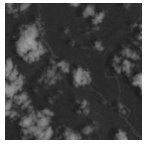
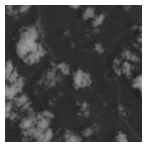
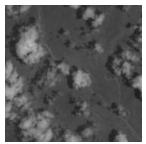
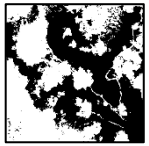
DeepGlobe Dataset:

```
34.
35.         img_path = os.path.join(self.ds_dir, img_name)
36.         mask_path = os.path.join(self.ds_dir, mask_name)
37.
38.         # Load image and mask (as RGB)
39.         image = np.array(Image.open(img_path).convert("RGB"))  # Ensure 3-channel RGB
40.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
41.
42.         # Convert RGB mask to class index mask
43.         mask = np.zeros((mask_rgb.shape[0], mask_rgb.shape[1]), dtype=np.uint8)
44.         for rgb, class_idx in self.rgb_to_class.items():
45.             # Assign class index to corresponding pixels
46.             mask[(mask_rgb == rgb).all(axis=-1)] = class_idx
47.
48.         # Apply augmentations if defined
49.         if self.transform:
50.             augmented = self.transform(image=image, mask=mask)
51.             image = augmented["image"]
52.             mask = augmented["mask"]
53.
54.         else:
55.             # Ensure float32 in [0, 1] and Tensor format (C, H, W)
56.             image = torch.from_numpy(img_as_float(image)).permute(2, 0, 1).float()
57.             mask = torch.from_numpy(mask).unsqueeze(0).float()
58.
59.         return image, mask, img_path
```

3/3

Datasets and DataLoaders

- Binary dataset with multispectral images
  - 38-Cloud Dataset

red_[...].TIF 
green_[...].TIF 
blue _[...].TIF 
nir _[...].TIF 
gt _[...].TIF 

```
~/
└── Datasets/
    └── 38-cloud/
        ├── 38-Cloud_training/
        │   ├── Entire_scene_gts/
        │   ├── Natural_False_Color/
        │   ├── train_blue/
        │   ├── train_green/
        │   ├── train_gt/
        │   ├── train_nir/
        │   └── train_red/
        │       ├── red_patch_[...].TIF
        │       └── ...
        └── 38-Cloud_test/
```

Dataset (torch.utils.data)

```
+ __len__()
+ __getitem__()
```

CloudDataset

```
- transform: Callable/None
- files: list of dict
+ __init__(ds_path, file_list,
  transform)
+ combine_files(file, red_dir,
  green_dir, blue_dir, nir_dir, gt_dir)
+ open_image(idx, include_nir=False)
+ open_mask(idx)
+ __len__()
+ __getitem__(idx)
```

Data augmentation and preprocessing

Dictionary with paths to the channels and ground-truth

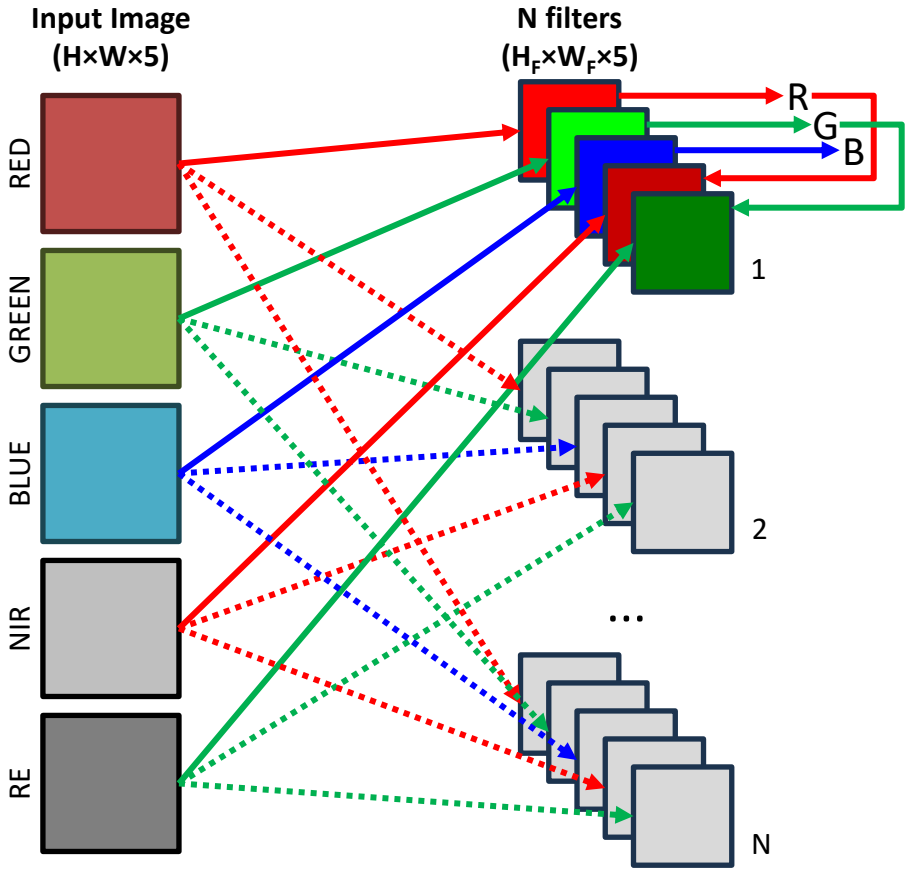
Combine channels to create a multispectral image

Open the multispectral image and convert it to the `uint8` data type.

Open and preprocess mask

Extending Pre-training to Multispectral Channels

- **RGB**: pretrained weights are used directly.
- **Multispectral**: extra channels (e.g., NIR, RE) have no pretrained filters.
- They are initialized by **copying RGB** weights cyclically, for a consistent start.
  - R → G → B → R → G → B …



Datasets and DataLoaders

38-Cloud Dataset:

```python
class CloudDataset(Dataset):

    def __init__(self, ds_path, file_list, transform=None):
        super().__init__()

        self.transform = transform

        # Build the paths for the images
        red_path = os.path.join(ds_path, "train_red")
        green_path = os.path.join(ds_path, "train_green")
        blue_path = os.path.join(ds_path, "train_blue")
        nir_path = os.path.join(ds_path, "train_nir")
        gt_path = os.path.join(ds_path, "train_gt")

        # Loop through the files in red folder and combine, into a dictionary, the other bands
        self.files = [self.combine_files(file_,
                                         red_path,
                                         green_path,
                                         blue_path,
                                         nir_path,
                                         gt_path) for file_ in file_list]

    def combine_files(self, file, red_dir, green_dir, blue_dir, nir_dir, gt_dir):
        bands = {
            'red': red_dir,
            'green': green_dir,
            'blue': blue_dir,
            'nir': nir_dir,
            'gt': gt_dir
```

1/3

Datasets and DataLoaders

38-Cloud Dataset:

```
23.     def combine_files(self, file, red_dir, green_dir, blue_dir, nir_dir, gt_dir):
24.         bands = {
25.             'red': red_dir,
26.             'green': green_dir,
27.             'blue': blue_dir,
28.             'nir': nir_dir,
29.             'gt': gt_dir
30.         }
31.         return {band: os.path.join(dir_, f'{band}_{file}.TIF') for band, dir_ in bands.items()}
32.
33.     def __len__(self):
34.         """Returns the total number of samples in the dataset."""
35.         return len(self.files)
36.
37.     def open_image(self, idx, include_nir=False):
38.         # Load RGB channels and stack them
39.         raw_rgb = np.stack([np.array(Image.open(self.files[idx]['red'])),
40.                             np.array(Image.open(self.files[idx]['green'])),
41.                             np.array(Image.open(self.files[idx]['blue'])), ], axis=2)
42.
43.         # Optionally append the NIR channel
44.         if include_nir:
45.             nir = np.expand_dims(np.array(Image.open(self.files[idx]['nir'])), 2)
46.             raw_rgb = np.concatenate([raw_rgb, nir], axis=2)
47.
48.         # Convert 16-bit original images to 8-bit for processing
49.         raw_rgb = img_as_ubyte(raw_rgb)
50.
51.         return raw_rgb
```

2/3

Datasets and DataLoaders

38-Cloud Dataset:

```python
    def combine_files(self, file, red_dir, green_dir, blue_dir, nir_dir, gt_dir):
        bands = {
            'red': red_dir,
            'green': green_dir,
            'blue': blue_dir,
            'nir': nir_dir,
            'gt': gt_dir
        }
        return {band: os.path.join(dir_, f'{band}_{file}.TIF') for band, dir_ in bands.items()}

    def __len__(self):
        """Returns the total number of samples in the dataset."""
        return len(self.files)

    def open_image(self, idx, include_nir=False):
        # Load RGB channels and stack them
        raw_rgb = np.stack([np.array(Image.open(self.files[idx]['red'])),
                            np.array(Image.open(self.files[idx]['green'])),
                            np.array(Image.open(self.files[idx]['blue'])), ], axis=2)

        # Optionally append the NIR channel
        if include_nir:
            nir = np.expand_dims(np.array(Image.open(self.files[idx]['nir'])), 2)
            raw_rgb = np.concatenate([raw_rgb, nir], axis=2)

        # Convert 16-bit original images to 8-bit for processing
        raw_rgb = img_as_ubyte(raw_rgb)

        return raw_rgb
```

2/3

Datasets and DataLoaders

38-Cloud Dataset:

```
23.     def combine_files(self, file, red_dir, green_dir, blue_dir, nir_dir, gt_dir):
24.         bands = {
25.             'red': red_dir,
26.             'green': green_dir,
27.             'blue': blue_dir,
28.             'nir': nir_dir,
29.             'gt': gt_dir
30.         }
31.         return {band: os.path.join(dir_, f'{band}_{file}.TIF') for band, dir_ in bands.items()}
32.
33.     def __len__(self):
34.         """Returns the total number of samples in the dataset."""
35.         return len(self.files)
36.
37.     def open_image(self, idx, include_nir=False):
38.         # Load RGB channels and stack them
39.         raw_rgb = np.stack([np.array(Image.open(self.files[idx]['red'])),
40.                             np.array(Image.open(self.files[idx]['green'])),
41.                             np.array(Image.open(self.files[idx]['blue'])), ], axis=2)
42.
43.         # Optionally append the NIR channel
44.         if include_nir:
45.             nir = np.expand_dims(np.array(Image.open(self.files[idx]['nir'])), 2)
46.             raw_rgb = np.concatenate([raw_rgb, nir], axis=2)
47.
48.         # Convert 16-bit original images to 8-bit for processing
49.         raw_rgb = img_as_ubyte(raw_rgb)
50.
51.         return raw_rgb
```

2/3

Datasets and DataLoaders

38-Cloud Dataset:

```
51.          return raw_rgb
52.
53.      def open_mask(self, idx):
54.          raw_mask = np.array(Image.open(self.files[idx]['gt']))
55.          raw_mask = np.where(raw_mask==255, 1, 0)
56.
57.          return raw_mask
58.
59.      def __getitem__(self, idx):
60.          # Load image with RGB + NIR bands
61.          image = self.open_image(idx, include_nir=True)
62.          # Load corresponding binary mask
63.          mask = self.open_mask(idx)
64.
65.          img_path = str(self.files[idx]['red'])
66.
67.          # Apply data augmentation if provided
68.          if self.transform is not None:
69.              augmented = self.transform(image=image, mask=mask)
70.              image = augmented['image']
71.              mask = augmented['mask']
72.
73.          else:
74.              # Ensure float32 in [0, 1] and Tensor format (C, H, W)
75.              image = torch.from_numpy(img_as_float(image)).permute(2, 0, 1).float()
76.              mask = torch.from_numpy(mask).unsqueeze(0).float()
77.
78.          # Return the image, mask and the file path.
79.          return image, mask, img_path
```

3/3

Datasets and DataLoaders

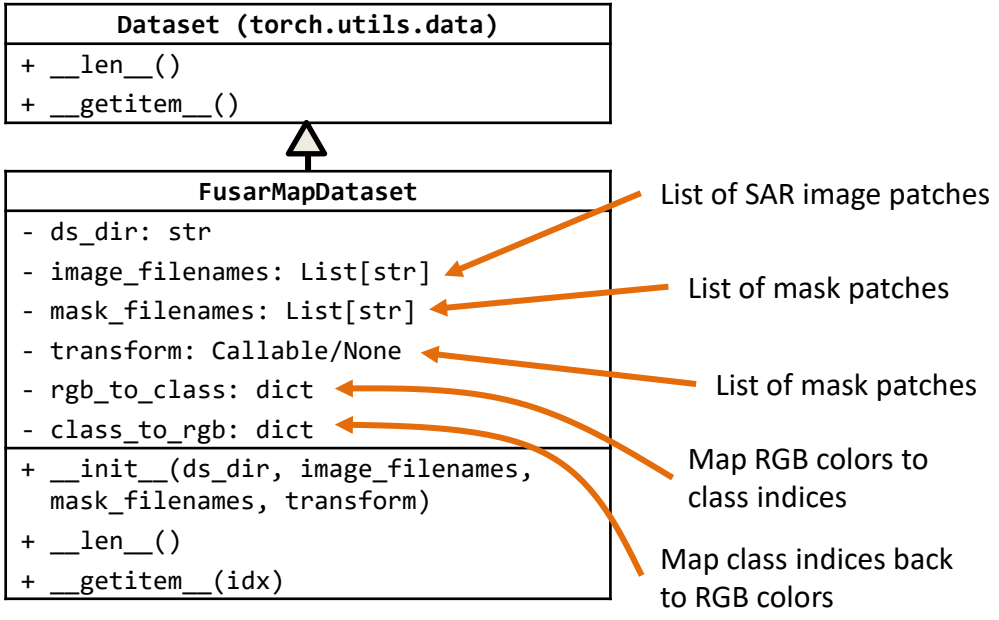
38-Cloud Dataset:

```
51.          return raw_rgb
52.
53.      def open_mask(self, idx):
54.          raw_mask = np.array(Image.open(self.files[idx]['gt']))
55.          raw_mask = np.where(raw_mask==255, 1, 0)
56.
57.          return raw_mask
58.
59.      def __getitem__(self, idx):
60.          # Load image with RGB + NIR bands
61.          image = self.open_image(idx, include_nir=True)
62.          # Load corresponding binary mask
63.          mask = self.open_mask(idx)
64.
65.          img_path = str(self.files[idx]['red'])
66.
67.          # Apply data augmentation if provided
68.          if self.transform is not None:
69.              augmented = self.transform(image=image, mask=mask)
70.              image = augmented['image']
71.              mask = augmented['mask']
72.
73.          else:
74.              # Ensure float32 in [0, 1] and Tensor format (C, H, W)
75.              image = torch.from_numpy(img_as_float(image)).permute(2, 0, 1).float()
76.              mask = torch.from_numpy(mask).unsqueeze(0).float()
77.
78.          # Return the image, mask and the file path.
79.          return image, mask, img_path
```

3/3

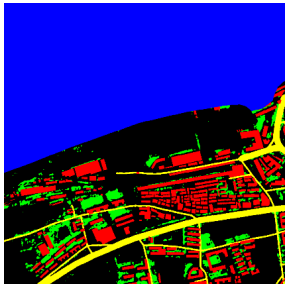
Datasets and DataLoaders

- Multiclass dataset with grayscale images
  - FUSAR-Map




```
~/
└── Datasets/
    └── FUSAR-Map/
        ├── FUSAR-Map1.0/
        │   ├── FUSAR-Map/
        │   │   ├── SAR_1024/
        │   │   │   ├── JiuJiang_01_01_01_01_SAR_001.tif
        │   │   │   ├── ...
        │   │   │   └── XiaMen_02_01_06_02_Label_124.tif
        │   │   ├── LAB_1024/
        │   │   │   ├── JiuJiang_01_01_01_01_SAR_001.tif
        │   │   │   ├── ...
        │   │   │   └── XiaMen_02_01_06_02_Label_124.tif
        │   │   └── datasets_info.mat
        │   └── ...
        └── FUSAR-Map1.0.zip
```

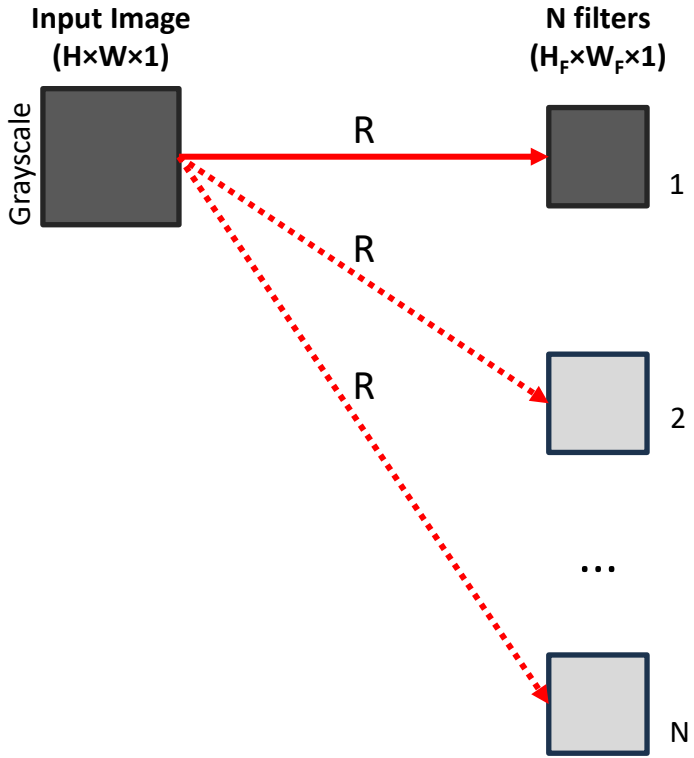
LAB_1024/JiuJiang_01_01_02_04_SAR_011.png



LAB_1024/JiuJiang_01_01_02_04_SAR_011.png

Pre-trained Weights for Grayscale Images

- **RGB**: pretrained weights are used directly.
- **Grayscale**: Initialize using only the Red channel weights.



Datasets and DataLoaders

FUSAR-Map Dataset:

```python
class FusarMapDataset(Dataset):

    def __init__(self, ds_dir, image_filenames, mask_filenames, transform=None):
        self.ds_dir = ds_dir
        self.image_filenames = image_filenames
        self.mask_filenames = mask_filenames
        self.transform = transform

        self.rgb_to_class = {
            (0, 0, 0):     0,   # black   - unknown/background
            (255, 0, 0):   1,   # red     - building
            (0, 255, 0):   2,   # green   - vegetation
            (0, 0, 255):   3,   # blue    - water
            (255, 255, 0): 4    # yellow  - road
        }

        # Inverse mapping: class index to RGB (for visualization)
        self.class_to_rgb = {v: k for k, v in self.rgb_to_class.items()}

    def __len__(self):
        return len(self.image_filenames)
```

1/3

Datasets and DataLoaders

FUSAR-Map Dataset:

```
17.         # Inverse mapping: class index to RGB (for visualization)
18.         self.class_to_rgb = {v: k for k, v in self.rgb_to_class.items()}
19.
20.     def __len__(self):
21.         return len(self.image_filenames)
22.
23.     def __getitem__(self, idx):
24.         img_name = self.image_filenames[idx]
25.         mask_name = self.mask_filenames[idx]
26.
27.         # Check filename match (optional)
28.         # Image: XiaMen_02_01_03_02_SAR_097.tif | Mask: XiaMen_02_01_03_02_Label_097.tif
29.         if re.sub(r'(SAR|Label)', '', img_name) != re.sub(r'(SAR|Label)', '', mask_name):
30.             print(f"Warning: Image and mask filename mismatch: {img_name} vs. {mask_name}")
31.
32.         img_path = os.path.join(self.ds_dir, 'SAR_1024', img_name)
33.         mask_path = os.path.join(self.ds_dir, 'LAB_1024', mask_name)
34.
35.         # Load grayscale image and mask
36.         image = np.array(Image.open(img_path).convert("L"))  # (H, W), grayscale
37.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
```

2/3

Datasets and DataLoaders

FUSAR-Map Dataset:

```
19.
20.     def __len__(self):
21.         return len(self.image_filenames)
22.
23.     def __getitem__(self, idx):
24.         img_name = self.image_filenames[idx]
25.         mask_name = self.mask_filenames[idx]
26.
27.         # Check filename match (optional)
28.         # Image: XiaMen_02_01_03_02_SAR_097.tif | Mask: XiaMen_02_01_03_02_Label_097.tif
29.         if re.sub(r'(SAR|Label)', '', img_name) != re.sub(r'(SAR|Label)', '', mask_name):
30.             print(f"Warning: Image and mask filename mismatch: {img_name} vs. {mask_name}")
31.
32.         img_path = os.path.join(self.ds_dir, 'SAR_1024', img_name)
33.         mask_path = os.path.join(self.ds_dir, 'LAB_1024', mask_name)
34.
35.         # Load grayscale image and mask
36.         image = np.array(Image.open(img_path).convert("L"))  # (H, W), grayscale
37.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
38.
```

2/3

Datasets and DataLoaders

FUSAR-Map Dataset:

```
31.
32.         img_path = os.path.join(self.ds_dir, 'SAR_1024', img_name)
33.         mask_path = os.path.join(self.ds_dir, 'LAB_1024', mask_name)
34.
35.         # Load grayscale image and mask
36.         image = np.array(Image.open(img_path).convert("L"))  # (H, W), grayscale
37.         mask_rgb = np.array(Image.open(mask_path).convert("RGB"))  # Read mask as RGB
38.
39.         # Convert RGB mask to class index mask
40.         mask = np.zeros((mask_rgb.shape[0], mask_rgb.shape[1]), dtype=np.uint8)
41.         for rgb, class_idx in self.rgb_to_class.items():
42.             # Assign class index to corresponding pixels
43.             mask[(mask_rgb == rgb).all(axis=-1)] = class_idx
44.
45.         if self.transform:
46.             augmented = self.transform(image=image, mask=mask)
47.             image = augmented["image"]
48.             mask = augmented["mask"]
49.         else:
50.             # Normalize to [0, 1] and convert to tensor with shape (1, H, W)
51.             image = torch.from_numpy(img_as_float(image)).unsqueeze(0).float()
52.             mask = torch.from_numpy(mask).unsqueeze(0).float()
53.
54.         return image, mask, img_path
```

3/3

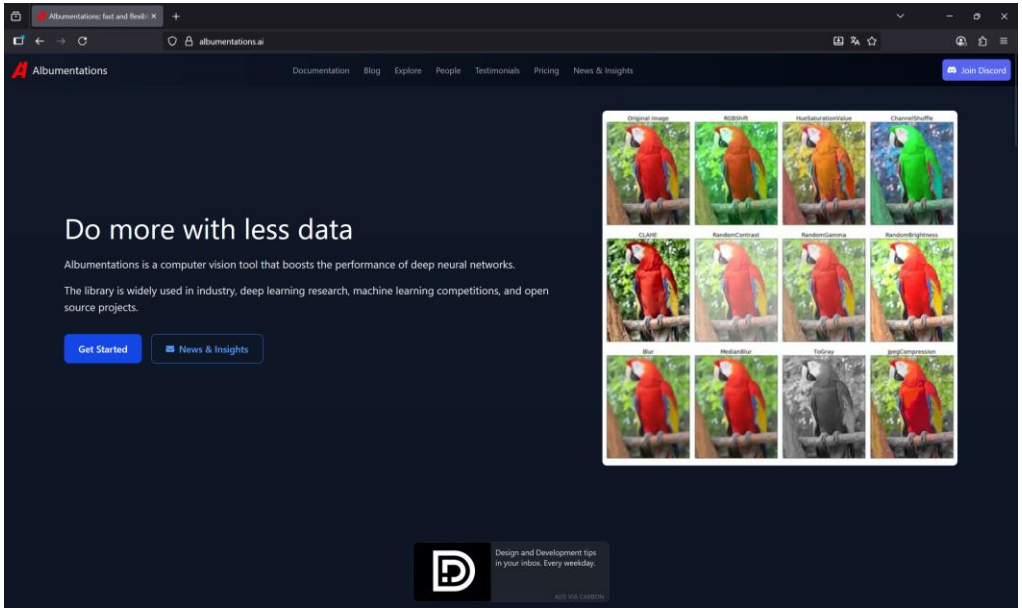
Datasets and DataLoaders

- Responsible for **loading** and **iterating** over datasets.
- Fetches one batch at a time.
- Can **shuffle the data** (recommended for training) or keep order (recommended for validation/test).
- `num_workers` defines how many CPU cores are used for data loading.
- `num_workers=0` ensures reproducibility but reduces performance.

```
g = torch.Generator().manual_seed(42)

# DataLoaders
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    shuffle=True, # Recommended for training
    num_workers=0, # Ensure reproducibility
    generator=g
)
val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=args.batch_size,
    shuffle=False,
    num_workers=0
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=args.batch_size,
    shuffle=False,
    num_workers=0
)
```

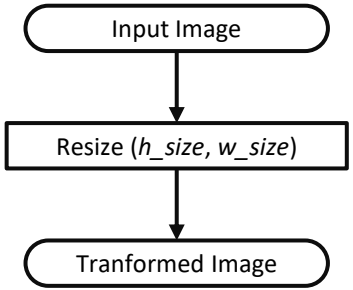
Data Augmentation

- Albumentations (https://albumentations.ai/)
  - A library for flexible and reliable transformations.
  - Ensures synchronized augmentations on input images and masks:
    - Geometric transforms (flips, rotations, scaling) applied consistently to both.
  - Supports image-only augmentations without affecting masks.
    - E.g.: color jittering



- DA Strategies:
  - No DA
  - Mild DA
  - Moderate DA
  - Strong DA

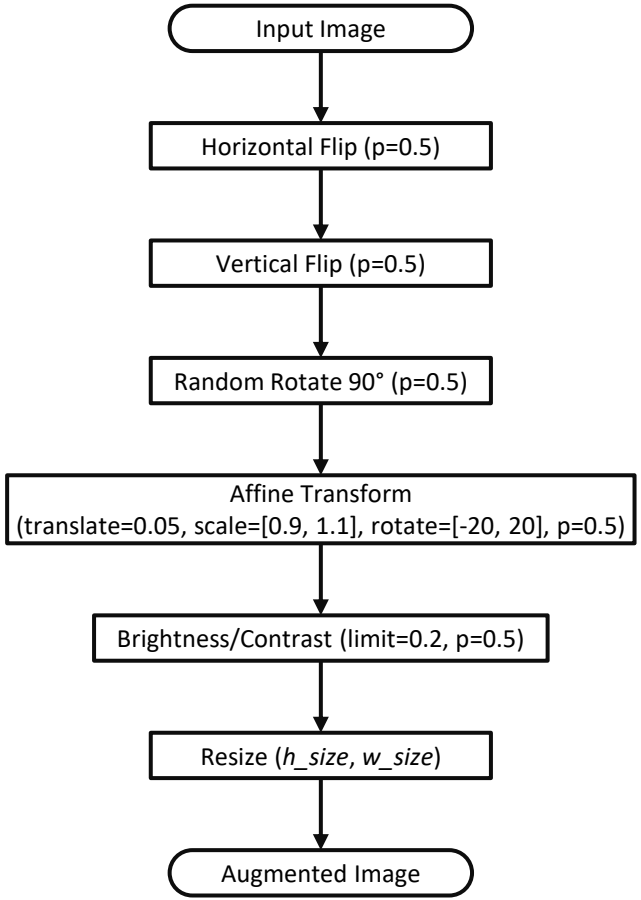
Data Augmentation




```
# Transformation for training
# without data augmentation or
# validation/test.
# No data augmentation.
t = [
    A.Resize(args.h_size,
     args.w_size)
]
```

No Data Augmentation

- Used for validation/test and training without DA (baseline)

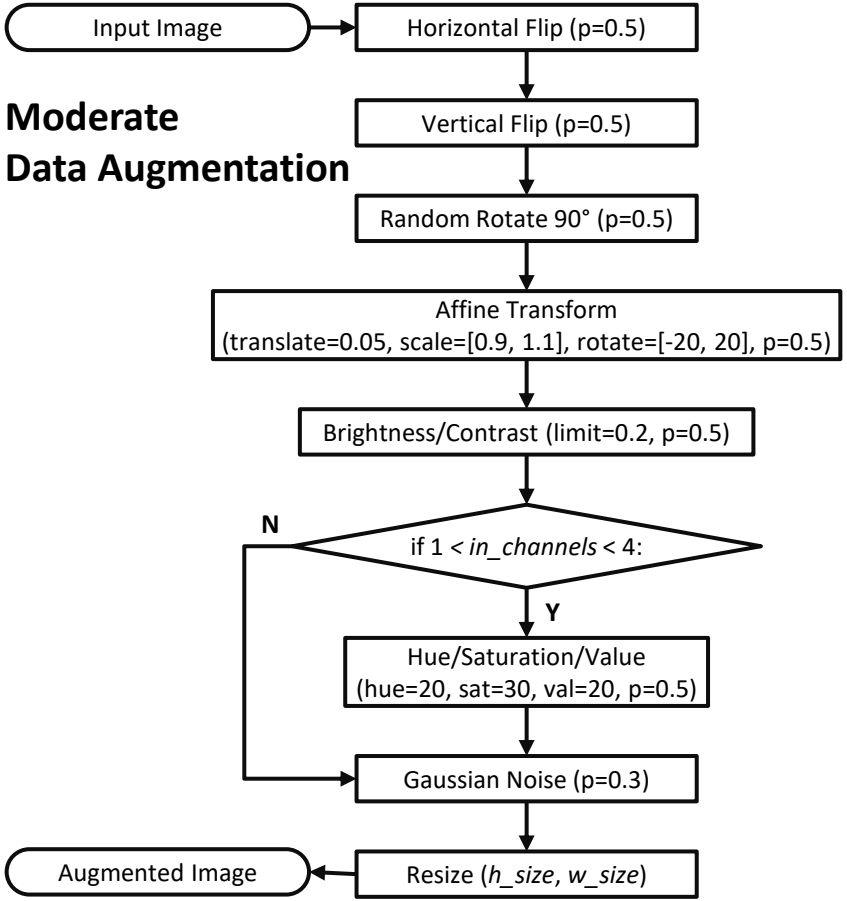


Mild Data Augmentation

```
# Transformations for training.
# Mild data augmentation.
t = [
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.Affine(
        translate_percent=0.05,
        scale=(0.9, 1.1),
        rotate=(-20, 20), p=0.5
    ),
   A.RandomBrightnessContrast(
        brightness_limit=0.2,
        contrast_limit=0.2, p=0.5
    ),
    A.Resize(args.h_size, args.w_size)
]
```

Data Augmentation

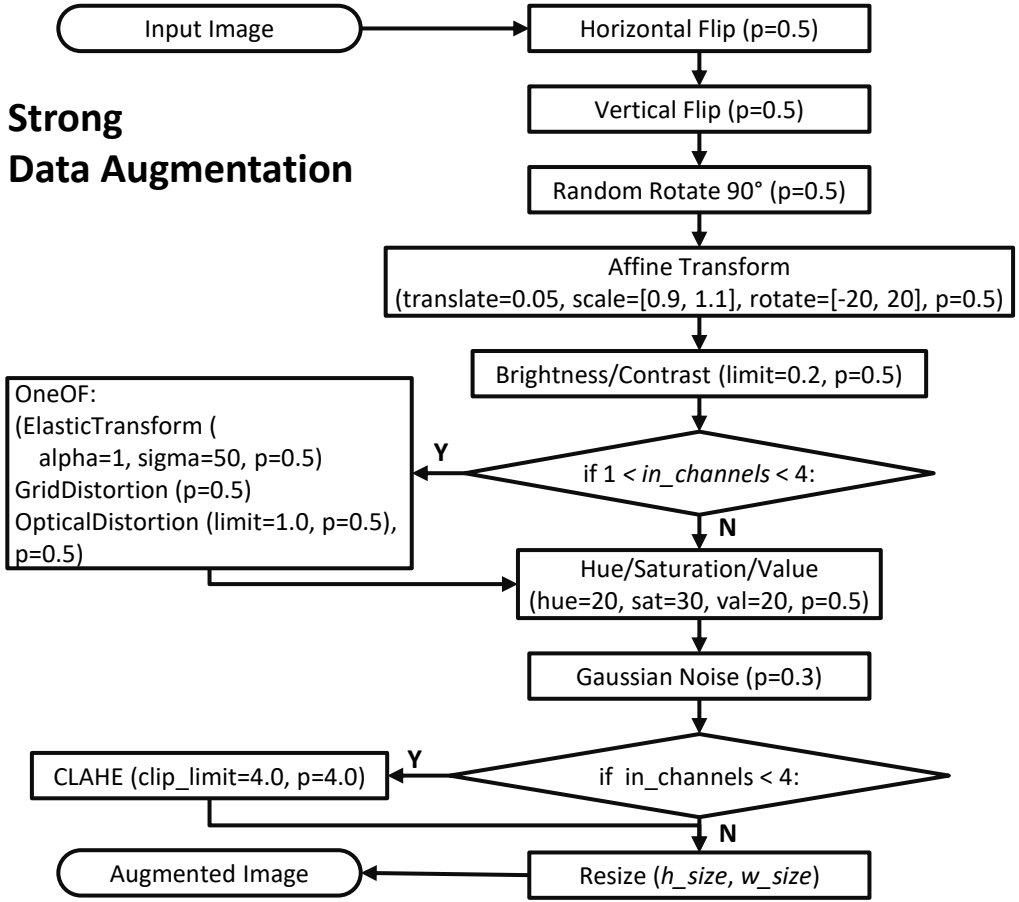
Moderate Data Augmentation




```python
# Transformations for training.
# Moderate data augmentation.
t = [
    A.HorizontalFlip(p=0.5), A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.Affine(translate_percent=0.05, scale=(0.9, 1.1),
        rotate=(-20, 20), p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2,
        contrast_limit=0.2, p=0.5)
]
# HueSaturationValue: apply only over RGB images
if in_channels > 1 and in_channels < 4:
    t += [
        A.HueSaturationValue(hue_shift_limit=20,
            sat_shift_limit=30, val_shift_limit=20, p=0.5)
    ]
t += [
    A.GaussNoise(p=0.3), A.Resize(args.h_size, args.w_size)
]
```

Data Augmentation

Strong Data Augmentation




```python
Strong data augmentation.
t = [
    A.HorizontalFlip(p=0.5), A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.Affine(translate_percent=0.05, scale=(0.9, 1.1),
        rotate=(-20, 20), p=0.5),
    A.RandomBrightnessContrast(brightness_limit=0.2,
        contrast_limit=0.2, p=0.5)
]
# HueSaturationValue: apply only over RGB images
if in_channels > 1 and in_channels < 4:
    t += [
        A.HueSaturationValue(hue_shift_limit=20,
            sat_shift_limit=30, val_shift_limit=20, p=0.5)
      ]
t += [
    A.OneOf([
        A.ElasticTransform(alpha=1, sigma=50, p=0.5)
        A.GridDistortion(p=0.5),
        A.OpticalDistortion(distort_limit=1.0, p=0.5)
    ], p=0.5),
       A.GaussNoise(p=0.3)
]
if in_channels < 4: # Avoid CLAHE for multispectral images
    t += [A.CLAHE(clip_limit=4.0, p=0.4)]
t += [A.Resize(args.h_size, args.w_size)]
```

Image Preprocessing

- You can use **`preprocessing_fn`** from SMP to correctly normalize images.
- Alternatively, normalization can be applied using Albumentations' **`Normalize`** method.
  - ⚠️ Be sure to match the statistics of the pretraining dataset (see *Pretrained Weights* slide).
- For **RGB** and **grayscale** images, usage is straightforward.
  - For **multispectral** images, compute **mean/std per channel** from the finetuning dataset.

```python
if preprocessing_fn is not None:
    print('Appliyng SMP preprocessing_fn...')
    t.append(A.Lambda(image=preprocessing_fn))
else:
    print('Applying A.Normalize...')
    t.append(A.Normalize(mean=mean, std=std))

# Convert to PyTorch Tensor
t.append(ToTensorV2())

# Fix Albumentations random seed to ensure reproducible augmentations
t = A.Compose(t, seed=seed)
```

Encoder-specific normalization using SMP's `preprocessing_fn`.

Manual normalization with `A.Normalize(mean, std)` (alternative to `preprocessing_fn`).

Convert to PyTorch tensors with `ToTensorV2()` (HWC → CHW; image scaled to [0,1]).

Fix Albumentations `seed` for reproducible augmentations.

Image Preprocessing

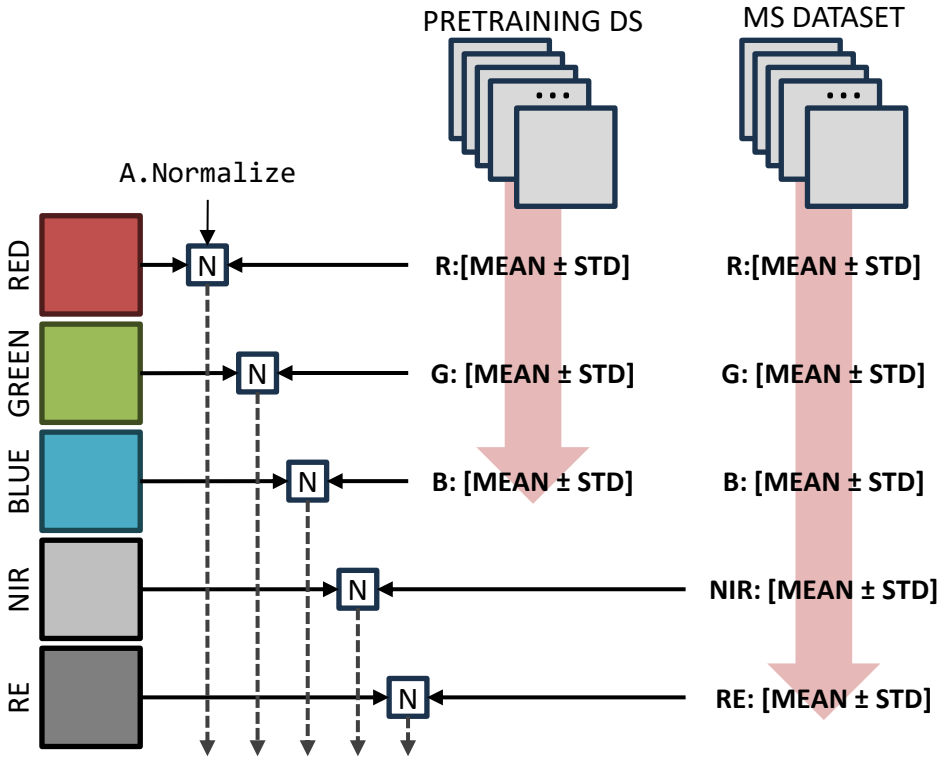
- This function computes the **mean and standard deviation** across all channels of a dataset.
  - It is especially useful for **multispectral datasets**.
  - ⚠️ Use only the **training set** to avoid data leakage.

```python
def compute_mean_std(dataset):
    sum_channels = sum_squared_channels = 0.0
    num_pixels = 0
    loader = DataLoader(dataset, batch_size=32, shuffle=False, num_workers=0)

    for images, _, *_ in loader:
        b, c, h, w = images.shape
        images = images.float()
        sum_channels += images.sum(dim=[0, 2, 3])
        sum_squared_channels += (images ** 2).sum(dim=[0, 2, 3])
        num_pixels += b * h * w

    mean = sum_channels / num_pixels
    std = torch.sqrt((sum_squared_channels / num_pixels) - mean ** 2)
    mean = [round(m.item(), 3) for m in mean]
    std = [round(s.item(), 3) for s in std]

    return mean, std
```

⚠️ This step may be slow. Recommended to run only once, during the first execution!

Image Preprocessing

- Multispectral images: Merge pretraining statistics with computed statistics
- Grayscale images: Use red channel statistics only




```python
if args.in_channels > 3:
    # For multispectral images (38-Cloud),
    # compute mean and std for the extra channels
    train_dataset = CloudDataset(DS_PATH, train_paths)
    # Compute mean and std for the extra channels.
    # Only for multispectral images.
    mean_ds, std_ds = compute_mean_std(train_dataset)
    # Merge pretraining stats with dataset-specific stats
    MEAN = MEAN + mean_ds[3:]
    STD = STD + std_ds[3:]

elif args.in_channels < 3:
    # For grayscale images,
    # Consider only the red channel statistics
    MEAN = MEAN[:args.in_channels]
    STD = STD[:args.in_channels]

print(f'Mean: {MEAN}')
print(f'Std Dev: {STD}')
```

Loss Functions

- Jaccard Loss or IoU Loss:
  - $\mathcal{L}_{Jaccard} = 1 - IoU$
  - If binary:
    - $IoU = \frac{TP}{TP+FP+FN} = \frac{|P \cap T|}{|P \cup T|}$
  - If multiclass:
    - $IoU = \frac{1}{C} = \sum_c IoU_c$

- Dice Loss:
  - $\mathcal{L}_{Dice} = 1 - Dice$
  - If binary:
    - $Dice = \frac{2\times TP}{2\times TP+FP+FN} = \frac{2|P \cap T|}{|P|+|T|}$
  - If multiclass:
    - $Dice = \frac{1}{C} = \sum_c Dice_c$

- Binary Cross-Entropy with Logits Loss:
  - $\mathcal{L}_{BCE} = -\left[y \log\left(\sigma(x)\right) + (1-y) \log\left(1-\sigma(x)\right)\right]$
- Cross-Entropy Loss
  - $\mathcal{L}_{CE} = -\sum_{c=1}^{C} y_c \log(p_c)$

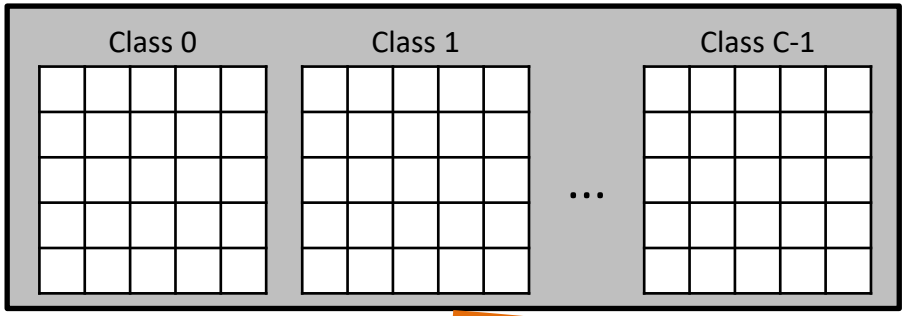
- Other Losses:
  - Tversky loss
  - Focal Loss
  - Lovász loss
  - Matthews Correlation Coefficient (MCC)
    For binary segmentation only

Loss Functions


```python
# Mapping loss function names to their corresponding SMP constructors
losses_map = {
    "jaccard": smp.losses.JaccardLoss,
    "dice": smp.losses.DiceLoss,
    "tversky": smp.losses.TverskyLoss,
    "focal": smp.losses.FocalLoss,
    "lavosz": smp.losses.LovaszLoss,
    # Cross-entropy handled separately due to its implementation differences in SMP
    # smp.losses.SoftBCEWithLogitsLoss and smp.losses.SoftCrossEntropyLoss,
}

if args.loss == 'crossentropy':
    if args.n_classes == 1: # Binary segmentation task — Using SoftBCEWithLogitsLoss
        criterion = smp.losses.SoftBCEWithLogitsLoss()
    else: # Multiclass segmentation task — Using SoftCrossEntropyLoss
        criterion = smp.losses.SoftCrossEntropyLoss(smooth_factor=0.0)
else:
    # Other losses: Jaccard, Dice, Tversky, Focal, Lovasz
    if args.n_classes == 1:
        criterion = losses_map[args.loss](
            mode='binary',
            from_logits=True
        )
    else:
        criterion = losses_map[args.loss](
            mode='multiclass',
            from_logits=True,
        )
```

Loss Functions

Logits (one channel per class)



(B, 1, H, W) for binary
(B, C, H, W) for multiclass

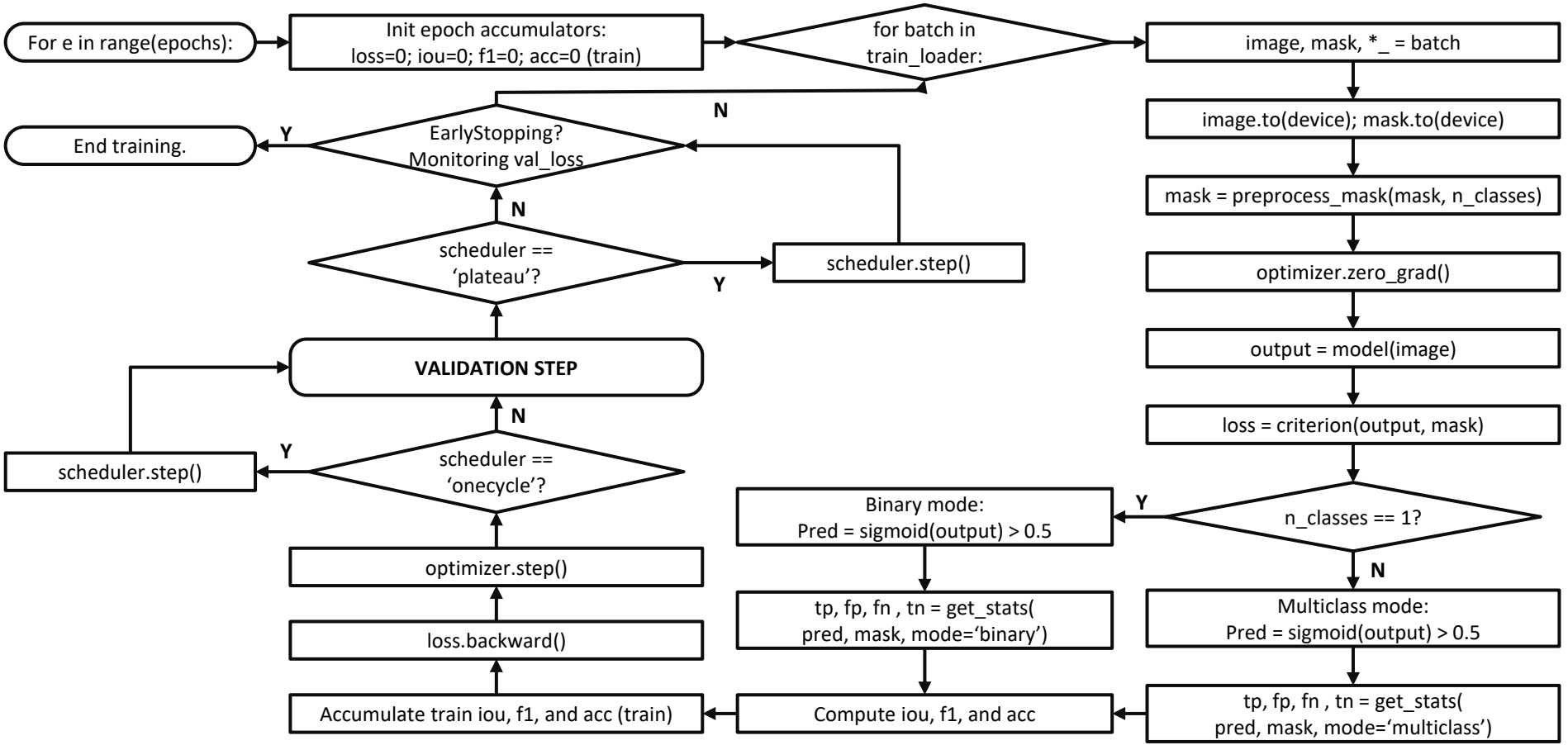
Label Coded Classes

0	1	2	1	2
0	1	0	1	2
1	0	1	2	0
0	1	2	0	1
1	2	0	1	2

(B, 1, H, W) for binary
(B, H, W) for multiclass

```
loss = criterion(output, mask)
```

Training/Validation Loop



Training/Validation Loop


```python
for e in range(epochs):
    since = time.time()
    model.train()
    train_loss_epoch = train_iou_smp_epoch = train_f1_smp_epoch = train_acc_smp_epoch = 0
    # TRAINING LOOP -----
    for i, data in enumerate(train_loader):
        image, mask, *_ = data
        # Send data to device
        image = image.float().to(device) # (B, I, H, W)
        mask = mask.to(device) # (B, H, W)
        # Preprocess mask:
        mask = preprocess_mask(mask, args.n_classes)
        # Reset gradient
        optimizer.zero_grad()
        # Forward pass
        output = model(image) # (B, 1, H, W) for binary loss. (B, C, H, W) for multiclass loss.
        # Loss computation
        loss = criterion(output, mask)
        # Prediction post-processing
        if args.n_classes == 1:
            # Binary mode. Output should be (B, 1, H, W) and dtype float
            pred = (torch.sigmoid(output) > 0.5).float() # (B, 1, H, W)
        else:
            # Multiclass mmode. Output should be (B, H, W) and dtype int
            pred = torch.argmax(output, dim=1) # (B, H, W)
        train_loss_epoch += loss.item() # Update epoch loss
```

1/4

Training/Validation Loop


```
27.         # Compute metrics using SMP
28.         if args.n_classes == 1:
29.             # PRED_MASK: (B, 1, H, W). MASK: (B, 1, H, W) and dtype long
30.             tp, fp, fn, tn = smp.metrics.get_stats(
31.                 pred.cpu(), mask.long().cpu(), mode='binary', threshold=0.5)
32.         else:
33.             # PRED_MASK: (B, H, W). MASK: (B, H, W)) and dtype long
34.             tp, fp, fn, tn = smp.metrics.get_stats(
35.                 pred.cpu(), mask.cpu(), mode='multiclass', num_classes=args.n_classes)
36.         # Updatate epoch metrics
37.         train_iou_smp_epoch += smp.metrics.iou_score(tp, fp, fn, tn, reduction='macro').item()
38.         train_f1_smp_epoch += smp.metrics.f1_score(tp, fp, fn, tn, reduction='macro').item()
39.         train_acc_smp_epoch += smp.metrics.accuracy(tp, fp, fn, tn, reduction='macro').item()
40.         # Backpropagation and optimizer step
41.         loss.backward()
42.         optimizer.step()
43.         # Scheduler step (OneCycle called every batch)
44.         if args.scheduler == 'onecycle':
45.            scheduler.step()
46.
47.     # Compute average metrics per epoch
48.     train_loss_epoch /= (i + 1); train_iou_smp_epoch /= (i + 1);
49.     train_f1_smp_epoch /= (i + 1); train_acc_smp_epoch /= (i + 1)
50.     # Store values
51.     train_loss_list.append(train_loss_epoch); train_iou_smp_list.append(train_iou_smp_epoch)
52.     train_f1_smp_list.append(train_f1_smp_epoch); train_acc_smp_list.append(train_acc_smp_epoch)
```

2/4

Training/Validation Loop


```
53.
54.     # VALIDATION LOOP -----
55.     for i, data in enumerate(train_loader):
56.         # Validation loop logic...
57.     # Compute average metrics per validation epoch
58.     val_loss_epoch /= (j + 1); val_iou_smp_epoch /= (j + 1)
59.     val_f1_smp_epoch /= (j + 1); val_acc_smp_epoch /= (j + 1)
60.
61.     # Scheduler step after epoch (Plateau or Cosine)
62.     if args.scheduler == 'plateau':
63.         scheduler.step(val_loss_epoch)
64.     elif args.scheduler == 'cosine' and e >= 10:
65.         # Warmup for the first 10 epochs.
66.         scheduler.step()
67.     # Store LR used this epoch
68.     lr_epoch = optimizer.param_groups[0]['lr']
69.     lr_list.append(lr_epoch)
70.     # Store values
71.     val_loss_list.append(val_loss_epoch); val_iou_smp_list.append(val_iou_smp_epoch)
72.     val_f1_smp_list.append(val_f1_smp_epoch); val_acc_smp_list.append(val_acc_smp_epoch)
73.
74.     # EARLY STOPPING LOGIC -----
75.     if val_loss_epoch < min_loss:
76.         print(f'Val loss improved: {min_loss:.4f} → {val_loss_epoch:.4f}. Saving model...')
77.         min_loss = val_loss_epoch; best_epoch = e; not_improve = 0
78.         try: # Save best model state_dict
```

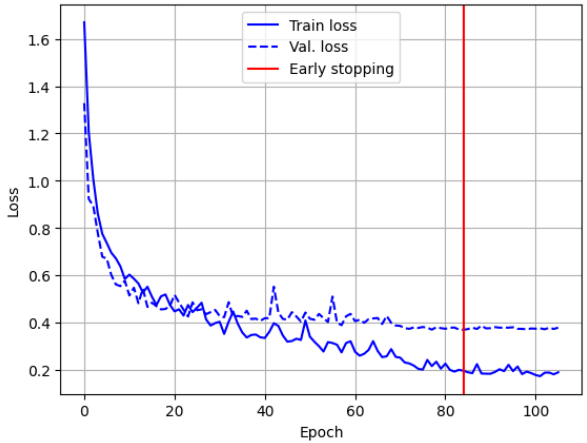
Training/Validation Loop

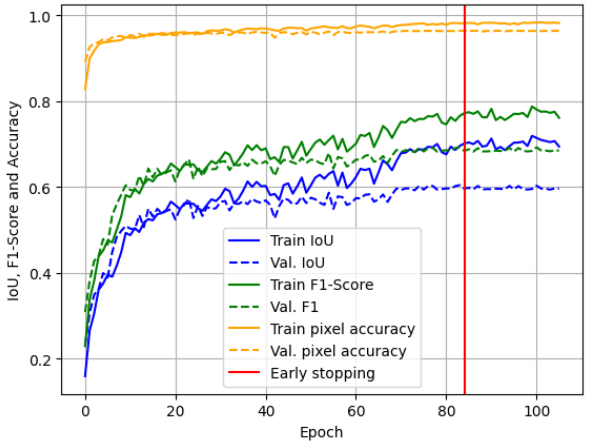

```
74.     # EARLY STOPPING LOGIC -----
75.     if val_loss_epoch < min_loss:
76.         print(f'Val loss improved: {min_loss:.4f} → {val_loss_epoch:.4f}. Saving model...')
77.         min_loss = val_loss_epoch; best_epoch = e; not_improve = 0
78.         try: # Save best model state_dict
79.             torch.save({
80.                 'epoch': e, 'model_state_dict': model.state_dict(),
81.                 'optimizer_state_dict': optimizer.state_dict(), 'val_loss': val_loss_epoch,
82.             }, os.path.join(EXP_PATH, 'best_model.pt'))
83.         except Exception as ex:
84.             print(f'Failed to save model: {ex}')
85.     else:
86.         not_improve += 1
87.         print(f'No improvement for {not_improve} epoch(s).')
88.         if not_improve >= args.patience:
89.             print(f'Validation loss did not improve for {args.patience} epochs. Stopping training...')
90.             break
91.
92. history = {
93.     'train_loss': train_loss_list, 'val_loss': val_loss_list, 'train_iou_smp': train_iou_smp_list,
94.     'val_iou_smp': val_iou_smp_list, 'train_f1_smp': train_f1_smp_list, 'val_f1_smp': val_f1_smp_list,
95.     'train_acc_smp': train_acc_smp_list, 'val_acc_smp': val_acc_smp_list, 'lrs': lr_list,
96.     'best_epoch': best_epoch, 'total_time': (time.time() - fit_time)/60
97. }
98.
99. print('Total training time: {:.2f}m'.format(history['total_time']))
```

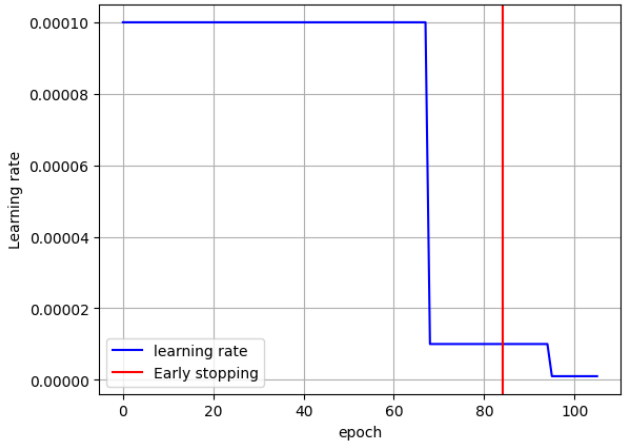
4/4

Training/Validation History

- Evaluate training quality
- Detect overfitting
- Identify training issues
- Assess the impact of learning rate scheduling







Prediction Loop


```
1.  # Path list
2.  path_list = [] # List of image patches
3.  # SMP metrics
4.  all_tp, all_fp, all_fn, all_tn = [], [], [], []
5.
6.  model.eval()
7.  model.to(device)
8.
9.  with torch.no_grad():
10.     for i, data in enumerate(dataloader):
11.         image, mask, path = data
12.         # Send data to device
13.         image = image.float().to(device)
14.         mask = mask.to(device)
15.         # Forward pass
16.         output = model(image)
17.         # Preprocess predictions and masks for metrics
18.         if args.n_classes == 1:
19.             # Binary mode. Output should be (B, 1, H, W) and dtype float
20.             pred = (torch.sigmoid(output) > 0.5).float()
21.             if pred.ndim == 3:  # (B, H, W) → (B, 1, H, W)
22.                 pred_mask = pred.unsqueeze(1)
23.             else:
24.                 pred_mask = pred
```

1/4

Prediction Loop


```
17.         # Preprocess predictions and masks for metrics
18.         if args.n_classes == 1:
19.             # Binary mode. Output should be (B, 1, H, W) and dtype float
20.             pred = (torch.sigmoid(output) > 0.5).float()
21.             if pred.ndim == 3:  # (B, H, W) → (B, 1, H, W)
22.                 pred_mask = pred.unsqueeze(1)
23.             else:
24.                 pred_mask = pred
25.             if mask.ndim == 3: # (B, H, W) → (B, 1, H, W)
26.                 mask = mask.unsqueeze(1)
27.         else:
28.             # Multiclass mode. Output should be (B, H, W) and dtype long
29.             pred_mask = torch.argmax(output, dim=1)  # (B, H, W)
30.             if mask.ndim == 4 and mask.shape[1] == 1:
31.                 mask = mask.squeeze(1)  # (B, H, W)
32.         # Get SMP stats (TP, FP, FN, TN)
33.         if args.n_classes == 1:  # binary
34.             # PRED_MASK: (B, 1, H, W). MASK: (B, 1, H, W) and dtype long
35.             tp, fp, fn, tn = smp.metrics.get_stats(
36.                 pred_mask.cpu(), mask.long().cpu(), mode='binary', threshold=0.5
37.             )
38.         else:
39.             # PRED_MASK: (B, H, W). MASK: (B, H, W) and dtype long
40.             tp, fp, fn, tn = smp.metrics.get_stats(
```

2/4

Prediction Loop


```
        # Get SMP stats (TP, FP, FN, TN)
        if args.n_classes == 1:  # binary
            # PRED_MASK: (B, 1, H, W). MASK: (B, 1, H, W) and dtype long
            tp, fp, fn, tn = smp.metrics.get_stats(
                pred_mask.cpu(), mask.long().cpu(), mode='binary', threshold=0.5
            )
        else:
            # PRED_MASK: (B, H, W). MASK: (B, H, W) and dtype long
            tp, fp, fn, tn = smp.metrics.get_stats(
                pred_mask.cpu(), mask.long().cpu(),
                mode='multiclass', num_classes=args.n_classes
            )
        # Aggregate the TP, FP, FN, TN across all batches
        all_tp.append(tp); all_fp.append(fp); all_fn.append(fn); all_tn.append(tn)
        path_list.extend(path)

# Concatenate all TP, FP, FN, TN across batches
tp_sum = torch.cat(all_tp); fp_sum = torch.cat(all_fp)
fn_sum = torch.cat(all_fn); tn_sum = torch.cat(all_tn)

 # Compute aggregated metrics using SMP
 smp_metrics = {
     f'iou': smp.metrics.functional.iou_score(
         tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
```

3/4

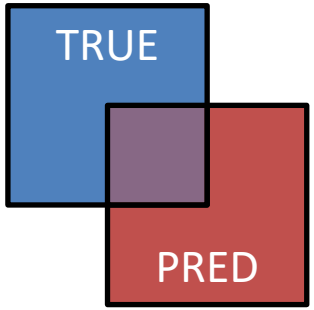
Prediction Loop

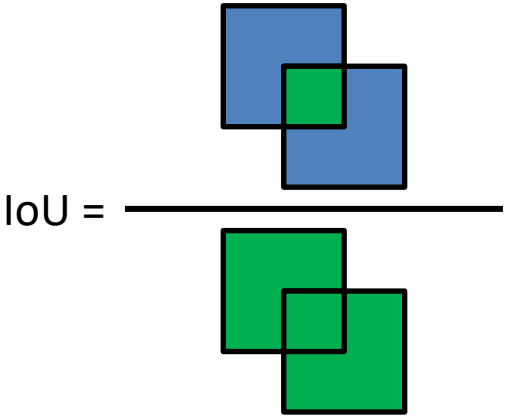

```
41.                 pred_mask.cpu(), mask.long().cpu(),
42.                 mode='multiclass', num_classes=args.n_classes
43.             )
44.         # Aggregate the TP, FP, FN, TN across all batches
45.         all_tp.append(tp); all_fp.append(fp); all_fn.append(fn); all_tn.append(tn)
46.         path_list.extend(path)
47.
48. # Concatenate all TP, FP, FN, TN across batches
49. tp_sum = torch.cat(all_tp); fp_sum = torch.cat(all_fp)
50. fn_sum = torch.cat(all_fn); tn_sum = torch.cat(all_tn)
51.
52.  # Compute aggregated metrics using SMP
53.  smp_metrics = {
54.      f'iou': smp.metrics.functional.iou_score(
55.          tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
56.      f'f1': smp.metrics.functional.f1_score(
57.         tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
58.      f'accuracy': smp.metrics.functional.accuracy(
59.         tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
60.      f'precision': smp.metrics.functional.precision(
61.         tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
62.      f'recall': smp.metrics.functional.recall(
63.         tp_sum, fp_sum, fn_sum, tn_sum, reduction=reduction).item(),
64. }
```

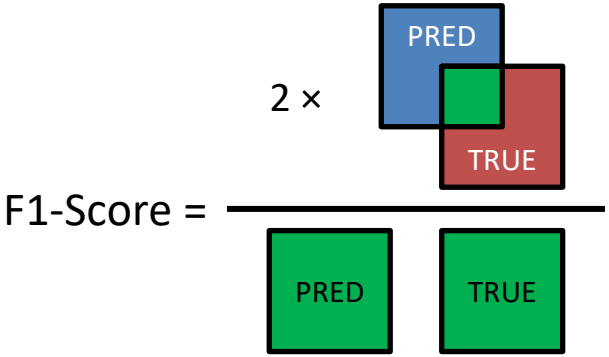
4/4

Evaluation Metrics and Reduction Strategies

- $IoU = \frac{TP}{TP+FP+FN} = \frac{|P \cap T|}{|P \cup T|}$
- $F1\text{-}Score = \frac{2 \times TP}{2 \times TP+FP+FN} = \frac{2|P \cap T|}{|P|+|T|}$
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$







Evaluation Metrics and Reduction Strategies

- Considering a metric function, $f(\cdot)$:

$$\text{Micro} - \text{Metric} = f\left(\sum_{i,c} TP_{i,c}, \sum_{i,c} FP_{i,c}, \sum_{i,c} FN_{i,c}, \sum_{i,c} TP_{i,c}\right)$$

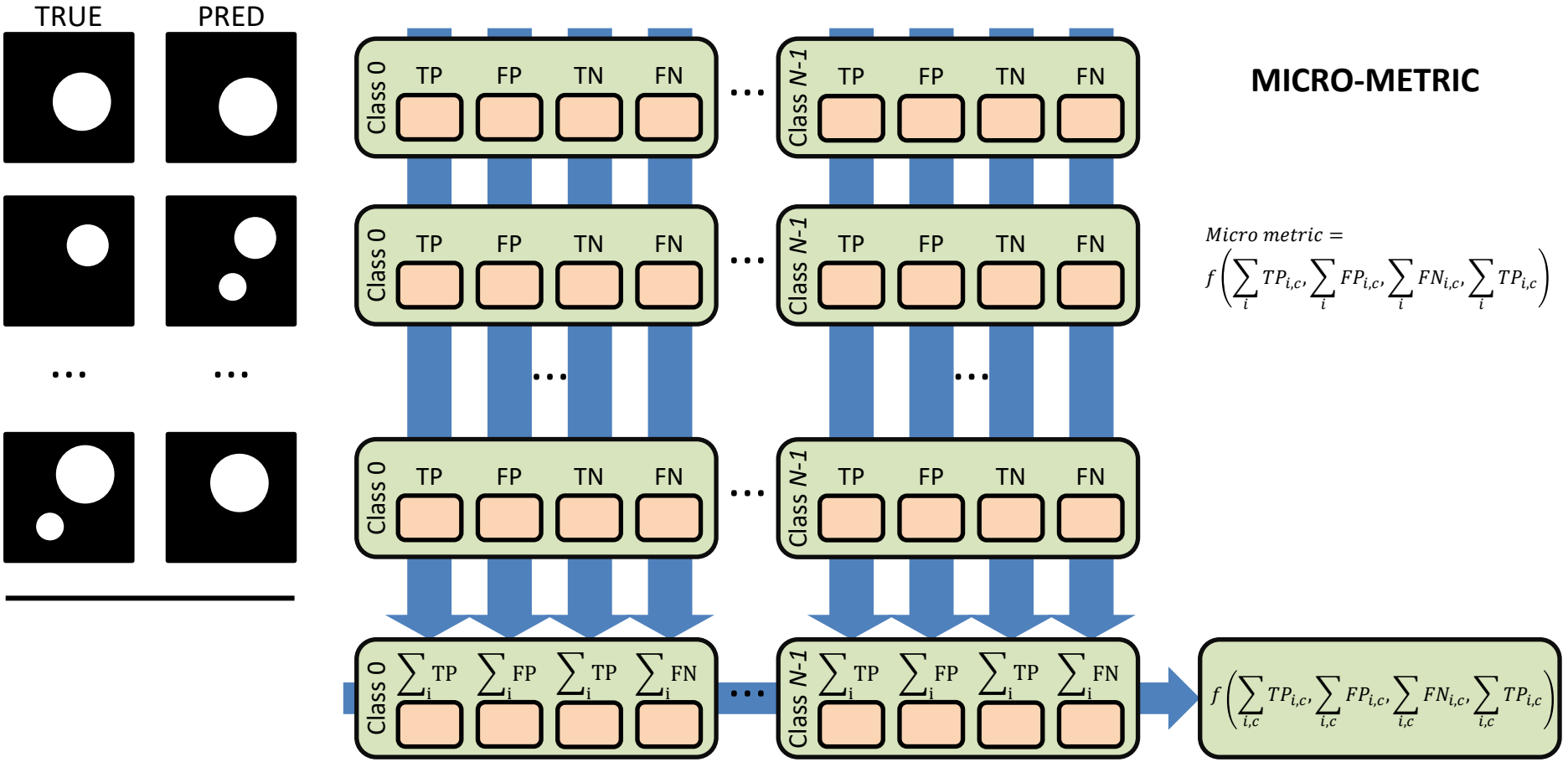
$$Macro - Metric = \frac{1}{C}\sum_{c=1}^{C} f\left(\sum_{i} TP_{i}, \sum_{i} FP_{i}, \sum_{i} FN_{i}, \sum_{i} TP_{i}\right)$$

$$Micro - Imagewise\ Metric = \frac{1}{N}\sum_{i=1}^{N} f\left(\sum_{c} TP_{i,c}, \sum_{c} FP_{i,c}, \sum_{c} FN_{i,c}, \sum_{c} TP_{i,c}\right)$$

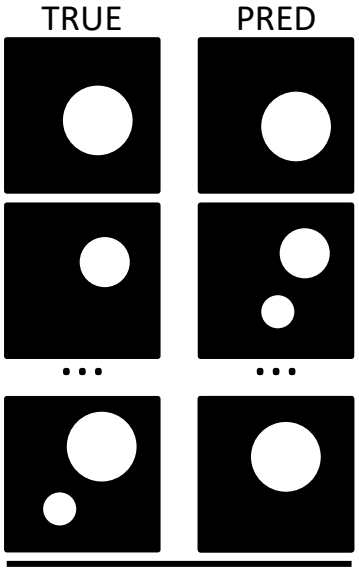
$$Macro - Imagewise\ Metric = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{C}\sum f\left(TP_{i,c}, FP_{i,c}, FN_{i,c}, TP_{i,c}\right)\right)$$

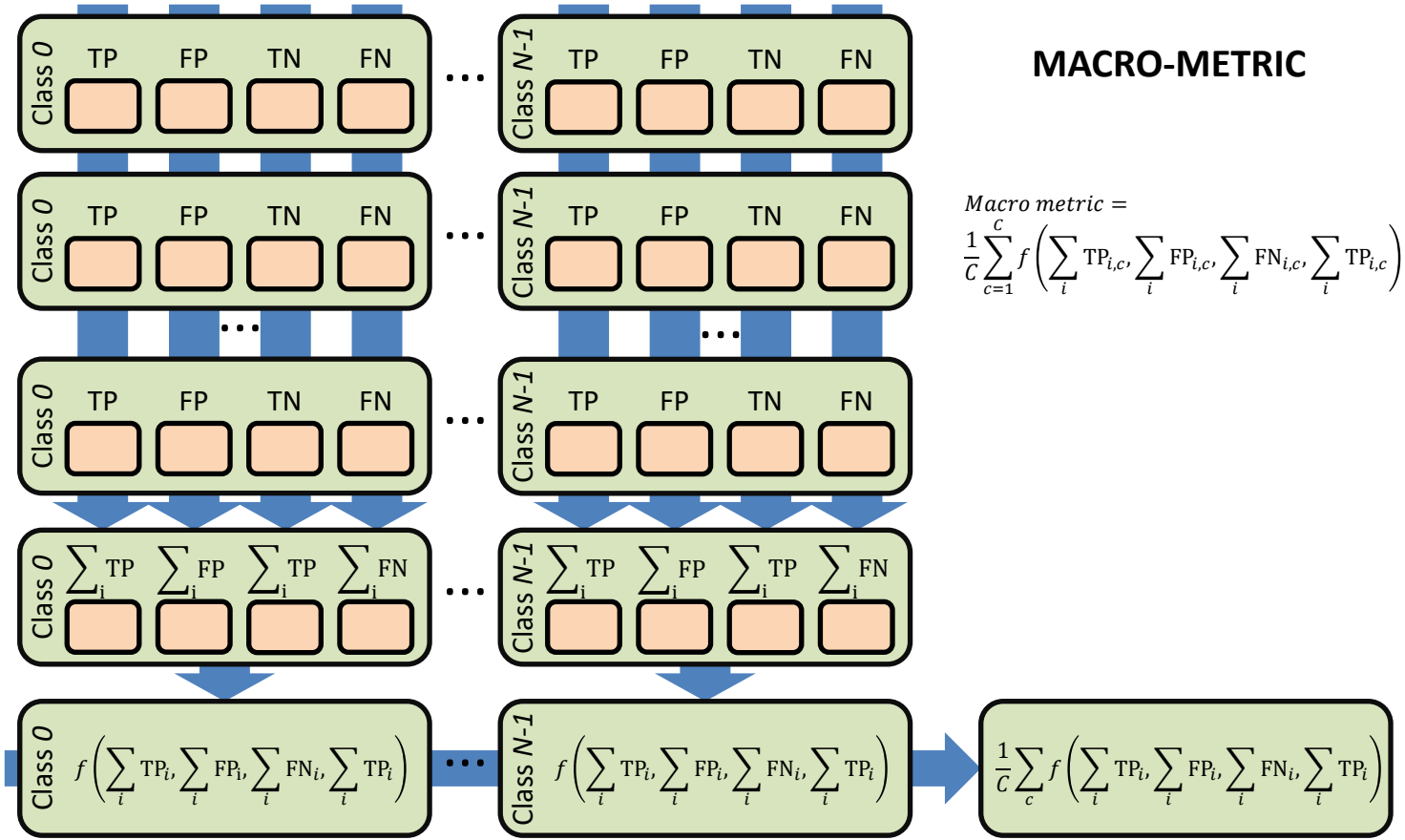
- C is the number of classes and c iterates along the classes.
- N is the number of images and i iterates along the images.

Evaluation Metrics and Reduction Strategies



Evaluation Metrics and Reduction Strategies





MACRO-METRIC

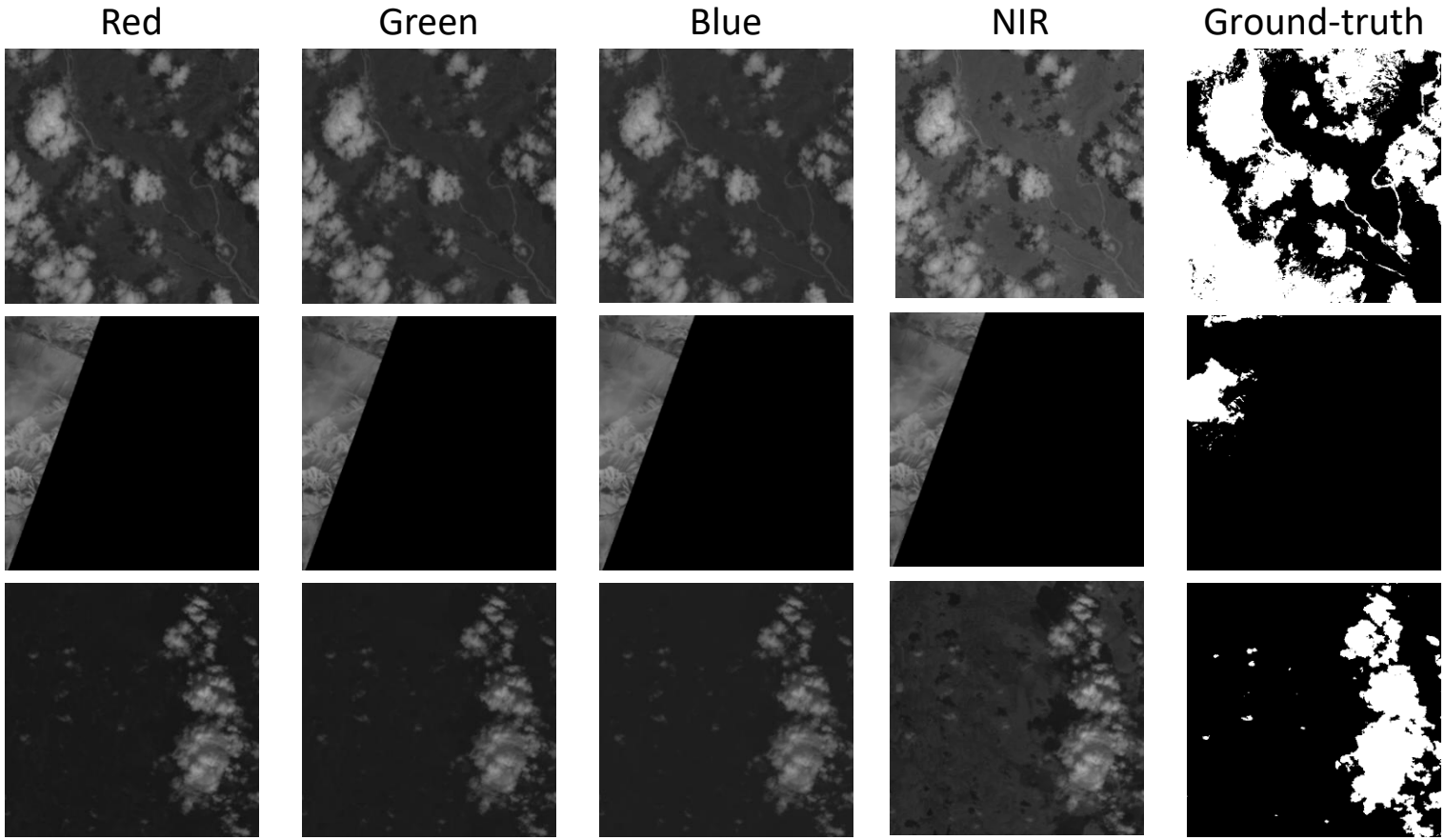
$$Macro\ metric = \frac{1}{C}\sum_{c=1}^{C} f\left(\sum_i \text{TP}_{i,c}, \sum_i \text{FP}_{i,c}, \sum_i \text{FN}_{i,c}, \sum_i \text{TP}_{i,c}\right)$$

CASE STUDY I: BINARY SEGMENTATION

38-Cloud Dataset

- 38-Cloud: A Cloud Segmentation Dataset
  - https://github.com/SorourMo/38-Cloud-A-Cloud-Segmentation-Dataset
- Scenes and images:
  - Training: 18 scenes → 8,400 patches
  - Testing: 20 scenes → 9,201 patches
- Spectral bands: Red (B4), Green (B3), Blue (B2), Near-Infrared (B5).
- Masks:
  - Training set: Pixel-level ground truth per patch (384×384).
  - Test set: Ground truth available only for the entire scenes (not individual patches).
- Notes:
  - Thin clouds (haze) are also labeled as clouds.
  - Evaluation is performed by reconstructing patch predictions into full-scene masks.

38-Cloud Dataset



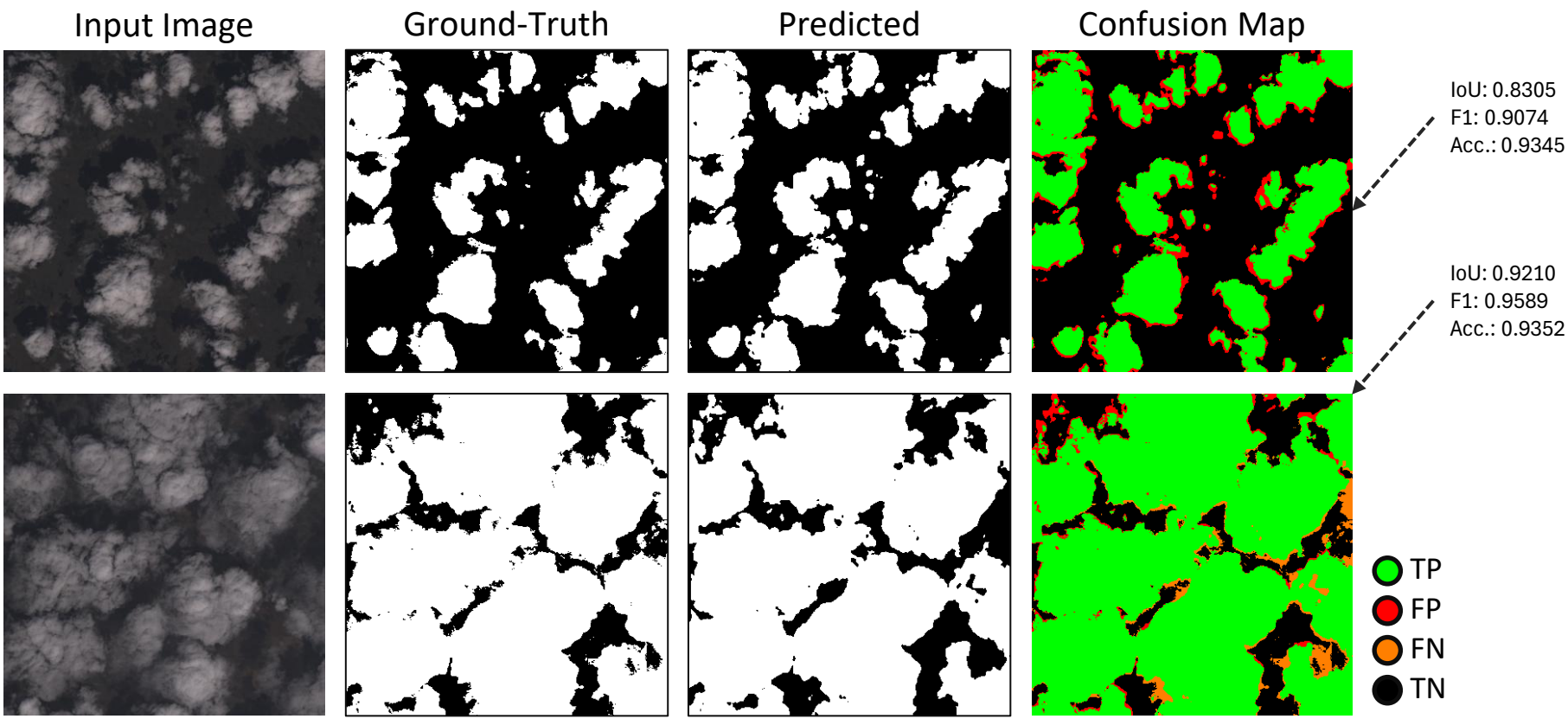
38-Cloud Dataset

- Batch size: 8
- Learning rate: 0.0001
- LR Scheduler: ReduceLROnPlateau

Loss function →		Cross Entropy						Dice					
Data Augmentation →		No DA			Moderate DA			No DA			Moderate DA		
Model	Encoder	IoU	F1	Acc.	IoU	F1	Acc.	IoU	F1	Acc.	IoU	F1	Acc.
U-Net	ResNet-50	0.9374	0.9677	0.9714	0.9297	0.9636	0.9677	0.9356	0.9667	0.9704	0.9185	0.9575	0.9621
	EfficientNet-b2	**0.9384**	**0.9682**	**0.9718**	0.9187	0.9576	0.9626	0.9344	0.9661	0.9699	0.9022	0.9486	0.9539
FPN	ResNet-50	0.9324	0.9650	0.9689	0.9217	0.9592	0.9638	**0.9342**	**0.9660**	**0.9697**	0.9079	0.9517	0.9570
	EfficientNet-b2	0.9331	0.9654	0.9692	0.9177	0.9571	0.9619	0.9261	0.9616	0.9659	0.8974	0.9459	0.9520

Macro reduction.

38-Cloud Dataset

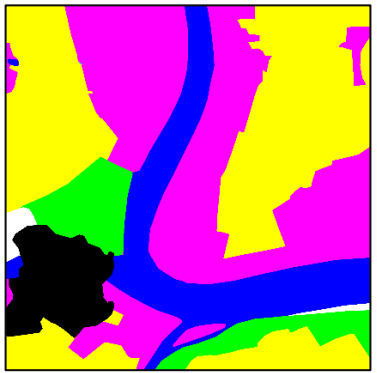


*Natural Color Composite

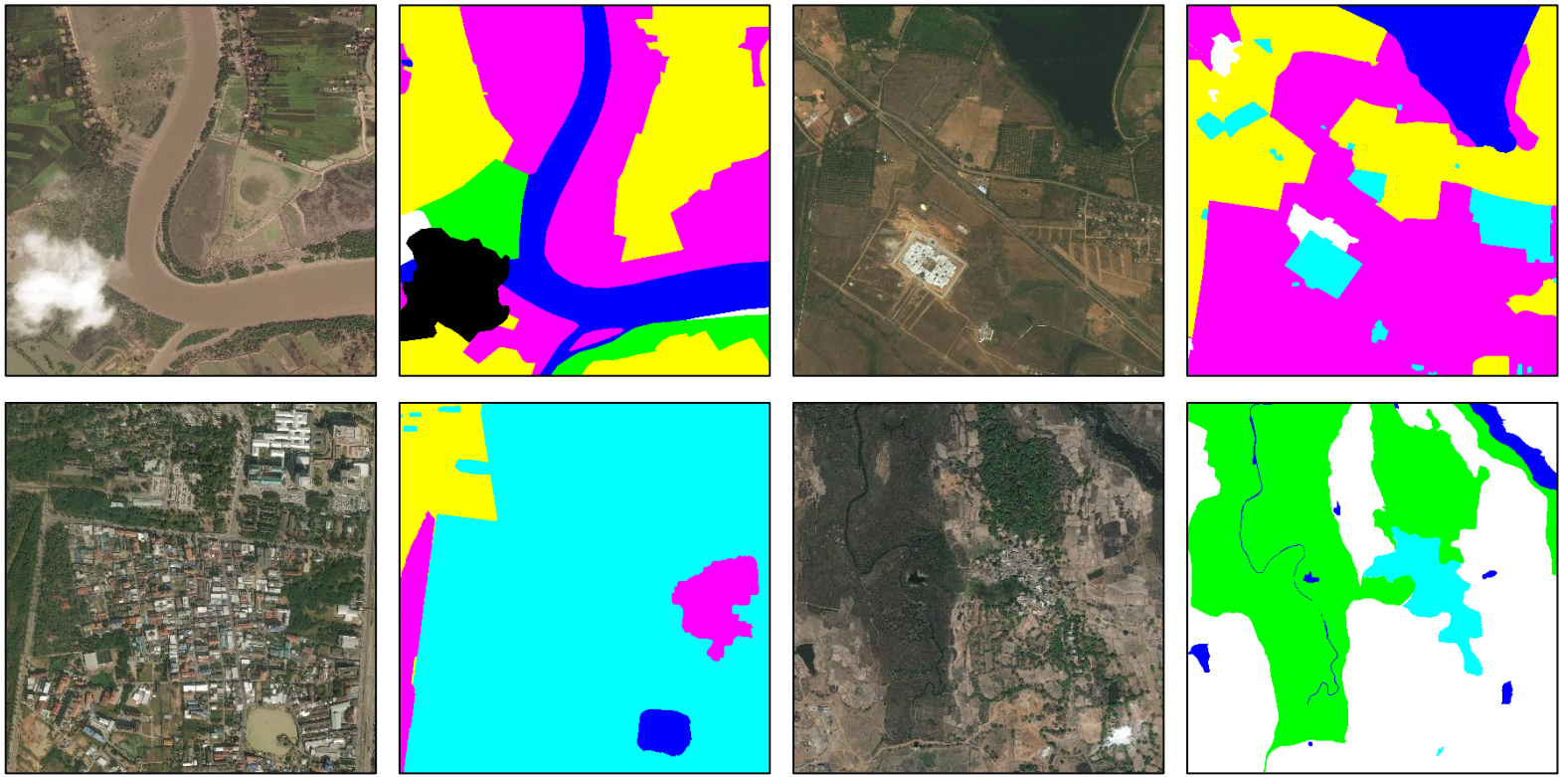
CASE STUDY II: MULTICLASS SEGMENTATION

DeepGlobe Dataset

- DeepGlobe Land Cover Classification Dataset
  - https://www.kaggle.com/datasets/balraj98/deepglobe-land-cover-classification-dataset
- Scenes:
  - Training: 803 RGB images (2,448×2,448, 50 cm resolution)
  - Validation: 171 images (no masks provided)
  - Testing: 172 images (no masks provided)
- Masks:
  - Each training/validation image has a paired RGB mask.
- 7 annotated classes (color-coded):
  - Urban: (0, 255, 255) - cyan
  - Agriculture: (255, 255, 0) - yellow
  - Rangeland: (255, 0, 255) - magenta
  - Forest: (0, 255, 0) - green
  - Water: (0, 0, 255) - blue
  - Barren: (255, 255, 255) - white
  - Unknown (clouds/others): (0, 0, 0) - black
- Notes:
  - Collected by DigitalGlobe satellites.
  - Challenge introduced in DeepGlobe 2018 (CVPR Workshops).



DeepGlobe Dataset



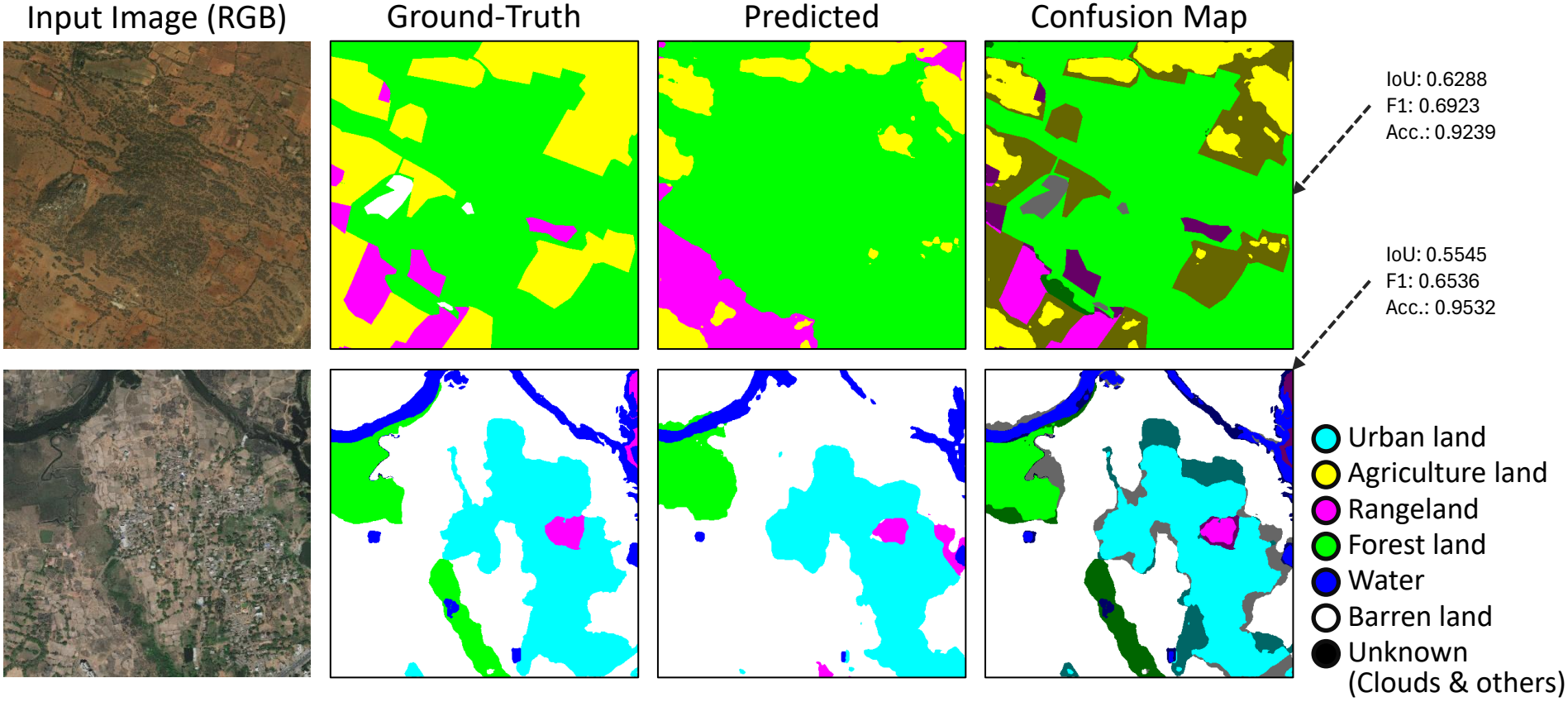
DeepGlobe Dataset

- Batch size: 8
- Learning rate: 0.0001
- LR Scheduler: ReduceLROnPlateau

Loss function →		**Cross Entropy**						**Dice**					
Data Augmentation →		**No DA**			**Moderate DA**			**No DA**			**Moderate DA**		
Model	**Encoder**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**
U-Net	ResNet-50	0.5814	0.6826	0.9582	0.5969	0.6941	0.9617	**0.6698**	**0.7912**	0.9599	0.5931	0.6916	0.9582
	EfficientNet-b2	0.5814	0.6838	0.9589	0.5835	0.6832	0.9608	0.6009	0.6965	0.9614	0.6079	0.7026	**0.9622**
FPN	ResNet-50	0.6045	0.7256	0.9598	0.6187	0.7472	0.9578	0.6778	0.7963	0.9616	0.6682	0.7905	0.9592
	EfficientNet-b2	0.6197	0.7453	0.9598	0.6195	0.7493	0.9569	0.6592	0.7832	0.9585	**0.7003**	**0.8154**	**0.9626**

Macro reduction.

DeepGlobe Dataset

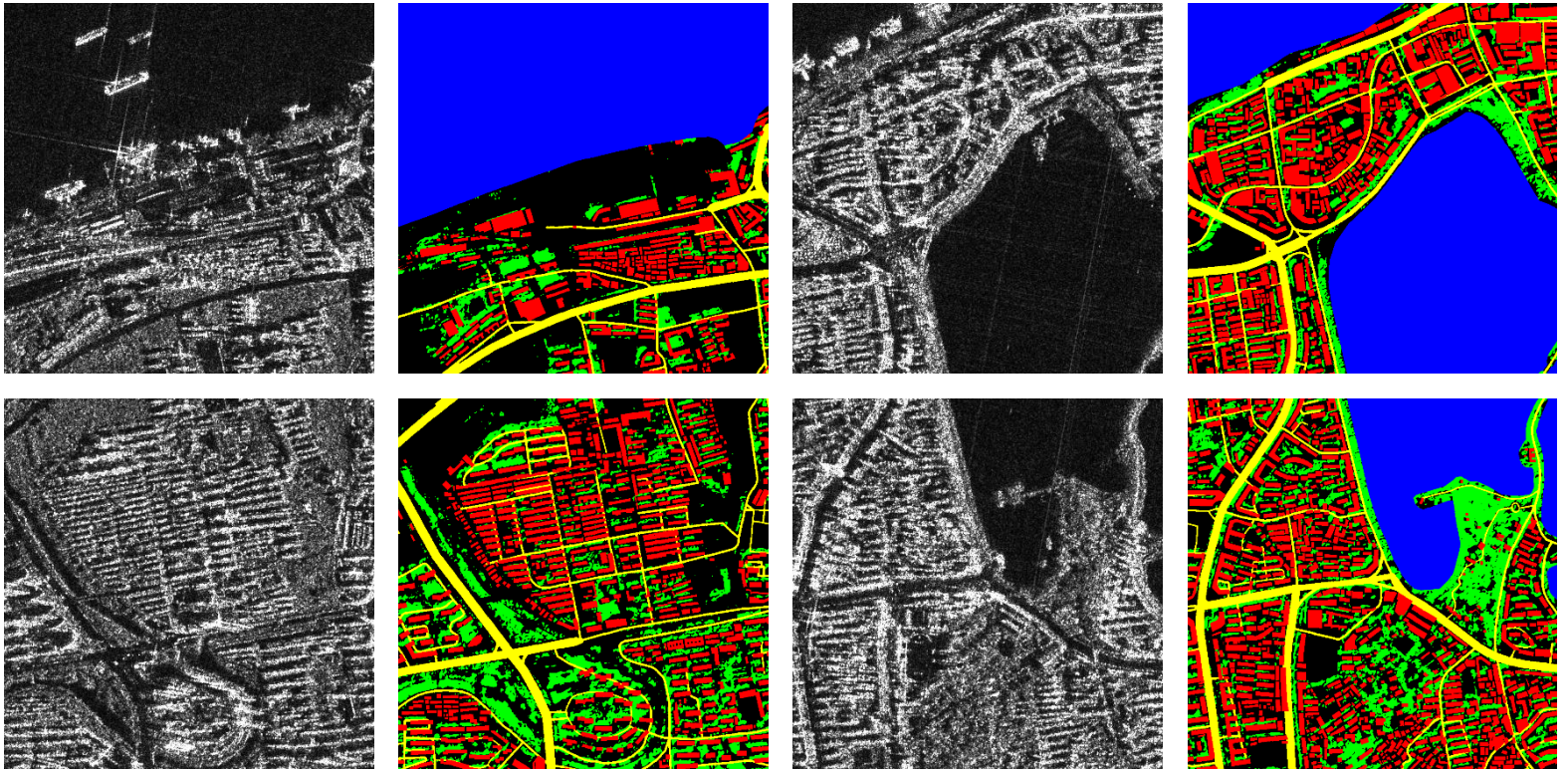


CASE STUDY III: GRAYSCALE IMAGE SEGMENTATION (BONUS)

FUSAR-Map Dataset

- FUSAR-Map: a benchmark dataset for SAR semantic segmentation
  - https://shixianzheng.github.io/FUSAR-Map/
- Scenes:
  - 610 SAR images: uint8, 1024×1024
  - Labels: uint8, RGB-coded masks (1024×1024×3)
  - Collected from 8 areas in 6 provinces (China), covering ~4,500 km²
- Masks:
  - Water: (0, 0, 255) - blue
  - Road: (255, 255, 0) - yellow
  - Building: (255, 0, 0) - red
  - Vegetation: (0, 255, 0) - green
  - Unknown pixels: (0, 0, 0) - black
- Features:
  - Each image has accurate latitude/longitude metadata.
  - Detailed naming convention encodes location, source image, row/column.
  - Includes SAR parameters (incidence angles, resolution, bounding boxes).

FUSAR-Map Dataset



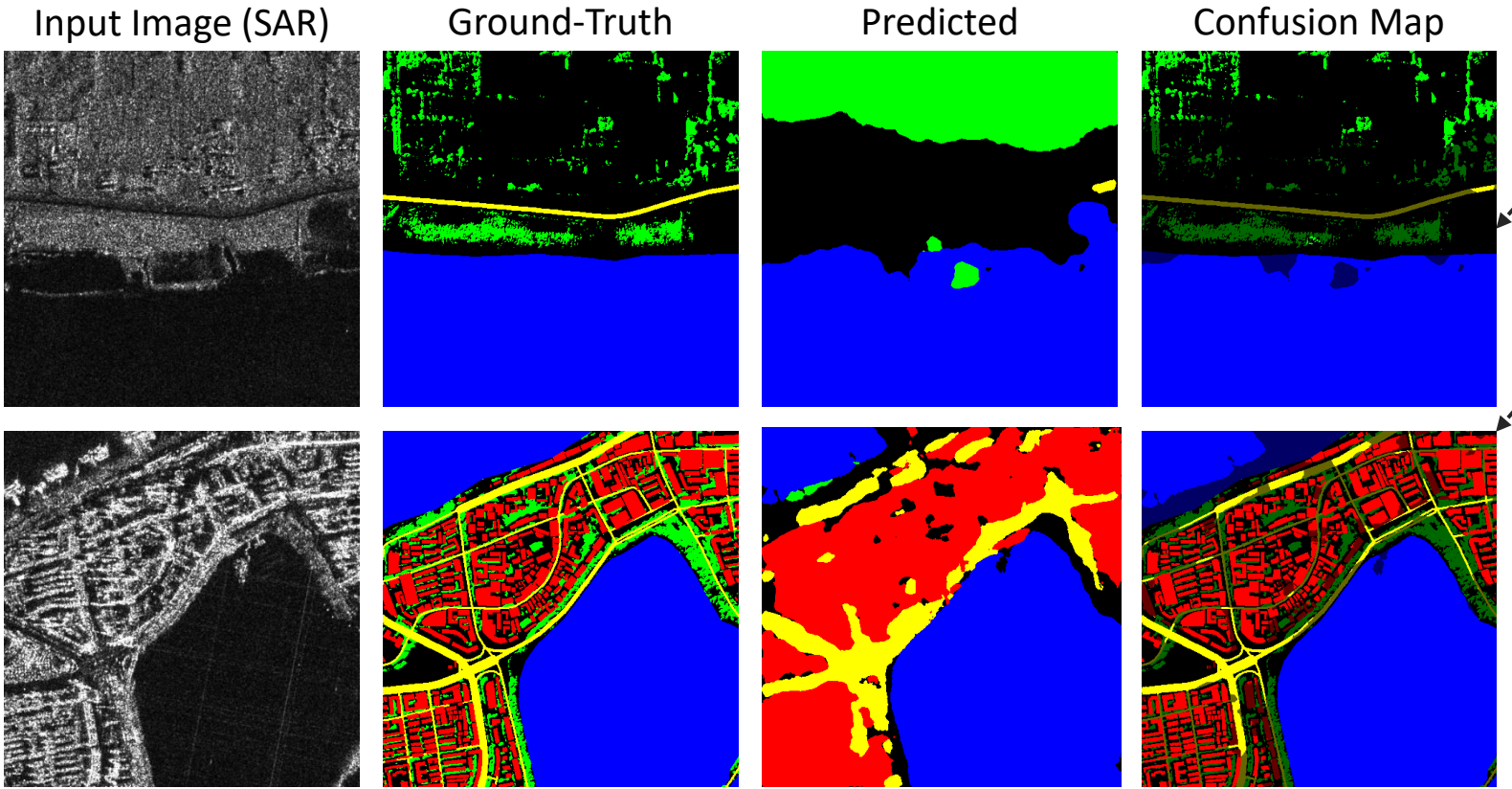
FUSAR-Map Dataset

- Batch size: 8
- Learning rate: 0.0001
- LR Scheduler: ReduceLROnPlateau

Loss function →		**Cross Entropy**						**Dice**					
Data Augmentation →		**No DA**			**Moderate DA**			**No DA**			**Moderate DA**		
Model	**Encoder**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**	**IoU**	**F1**	**Acc.**
U-Net	ResNet-50	0.3867	0.4657	0.8831	0.4059	0.4874	0.8878	0.4221	0.5434	0.8643	0.4457	0.5763	0.8582
	EfficientNet-b2	0.3964	0.4759	0.8844	0.4009	0.4759	**0.8882**	0.4462	0.5682	0.8704	**0.4471**	**0.5777**	0.8563
FPN	ResNet-50	0.3870	0.4757	0.8800	0.4080	0.4884	**0.8897**	0.4252	0.5467	0.8644	**0.4469**	**0.5762**	0.8623
	EfficientNet-b2	0.3921	0.4782	0.8810	0.4047	0.4881	0.8871	0.4244	0.5545	0.8543	0.4450	0.5751	0.8572

Macro reduction.

FUSAR-Map Dataset



Conclusion

- ✅ Learned how to perform semantic segmentation with the SMP library
- ✅ Understood the role of architectures, encoders, and transfer learning
- ✅ Built practical experiments for both binary and multiclass segmentation
  - Transparent PyTorch loops
  - Built-in SMP losses and metrics
- ✅ Explored case studies in remote sensing:
  - 🌎 Multispectral, RGB, and grayscale datasets
- ✅ Concepts are easily transferable to other domains such as medical imaging, agriculture, and industry

Future Directions

- This tutorial covered the basics of semantic segmentation with SMP. Next steps may explore:
- ⚡ **Model Efficiency**
  - Reduce encoder depth for lightweight models
  - Use **timm** encoders (GitHub link)
- 🧩 **Encoder Customization**
  - Create custom encoders
  - Freeze and unfreeze encoder layers
- 💾 **Training Workflow**
  - Save and load models with SMP utilities
  - Explore multilabel segmentation

Bibliography

- **Segmentation Models PyTorch (SMP):**
  - P. Iakubovskii, "**Segmentation models pytorch**," https://github.com/qubvel/segmentation_models.pytorch, 2019.
- **Albumentations:**
  - A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "**Albumentations: fast and flexible image augmentations**," Information, vol. 11, no. 2, p. 125, 2020.

Bibliography

- **38-Cloud Dataset:**
  - S. Mohajerani, T. A. Krammer, and P. Saeedi, "**A cloud detection algorithm for remote sensing images using fully convolutional neural networks**," in 2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP), 2018, pp. 1–5.
  - S. Mohajerani and P. Saeedi, "**Cloud-net: An end-to-end cloud detection algorithm for landsat 8 imagery**," in IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium, 2019, pp.1029–1032.
- **DeepGlobe Dataset:**
  - I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu,F. Hughes, D. Tuia, and R. Raskar, "**Deepglobe 2018: A challenge toparse the earth through satellite images**," in Proceedings of the IEEEconference on computer vision and pattern recognition workshops, 2018,pp. 172–181.
- **FUSAR-Map:**
  - X. Shi, S. Fu, J. Chen, F. Wang, and F. Xu, "**Object-level semantic segmentation on the high-resolution gaofen-3 fusar-map dataset**," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 3107–3119, 2021.

Bibliography

- **Segmentation Models:**
  - O. Ronneberger, P. Fischer, and T. Brox, "**U-net: Convolutional networks for biomedical image segmentation**," in MICCAI, 2015.
  - Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "**Unet++:A nested u-net architecture for medical image segmentation**," in Deep Learning in Medical Image Analysis, 2018.
  - T.-Y. Lin, P. Doll ´ar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "**Feature pyramid networks for object detection**," in CVPR, 2017.
  - H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "**Pyramid scene parsing network**," in CVPR, 2017.
  - L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "**Rethinking atrous convolution for semantic image segmentation**," in arXiv preprint arXiv:1706.05587, 2017.
  - L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "**Encoder-decoder with atrous separable convolution for semantic image segmentation**," in ECCV, 2018.
  - A. Chaurasia and E. Culurciello, "**Linknet: Exploiting encoder representations for efficient semantic segmentation**," in Visual Communications and Image Processing (VCIP), 2017.
  - T. Fan, G. Wang, Y. Li, and H. Wang, "**Ma-net: A multi-scale attention network for liver and tumor segmentation**," vol. 8. IEEE, 2020, pp.179 656–179 665.
  - H. Li, P. Xiong, J. An, and L. Wang, "**Pyramid attention network for semantic segmentation**," 2018.
  - T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "**Unified perceptual parsing for scene understanding**," in ECCV, 2018.
  - E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "**Segformer: Simple and efficient design for semantic segmentation with transformers**," in NeurIPS, 2021.
  - R. Ranftl, A. Bochkovskiy, and V. Koltun, "**Vision transformers for dense prediction**," in ICCV, 2021.

Acknowledgements

SIBGRAPI
38TH CONFERENCE ON GRAPHICS, PATTERNS, AND IMAGES 2025

CNPq
NVIDIA
CAPES
FAPEMIG
UFV
Universidade Federal de Viçosa
UFU
ufscar
UFV
Universidade Federal de Viçosa
DIRETORIA DE PESQUISA
E PÓS-GRADUAÇÃO

THE END!