
UNet/FCN PyTorch [Open in Colab](#)

This repository contains simple PyTorch implementations of U-Net and FCN, which are deep learning segmentation methods proposed by Ronneberger et al. and Long et al.

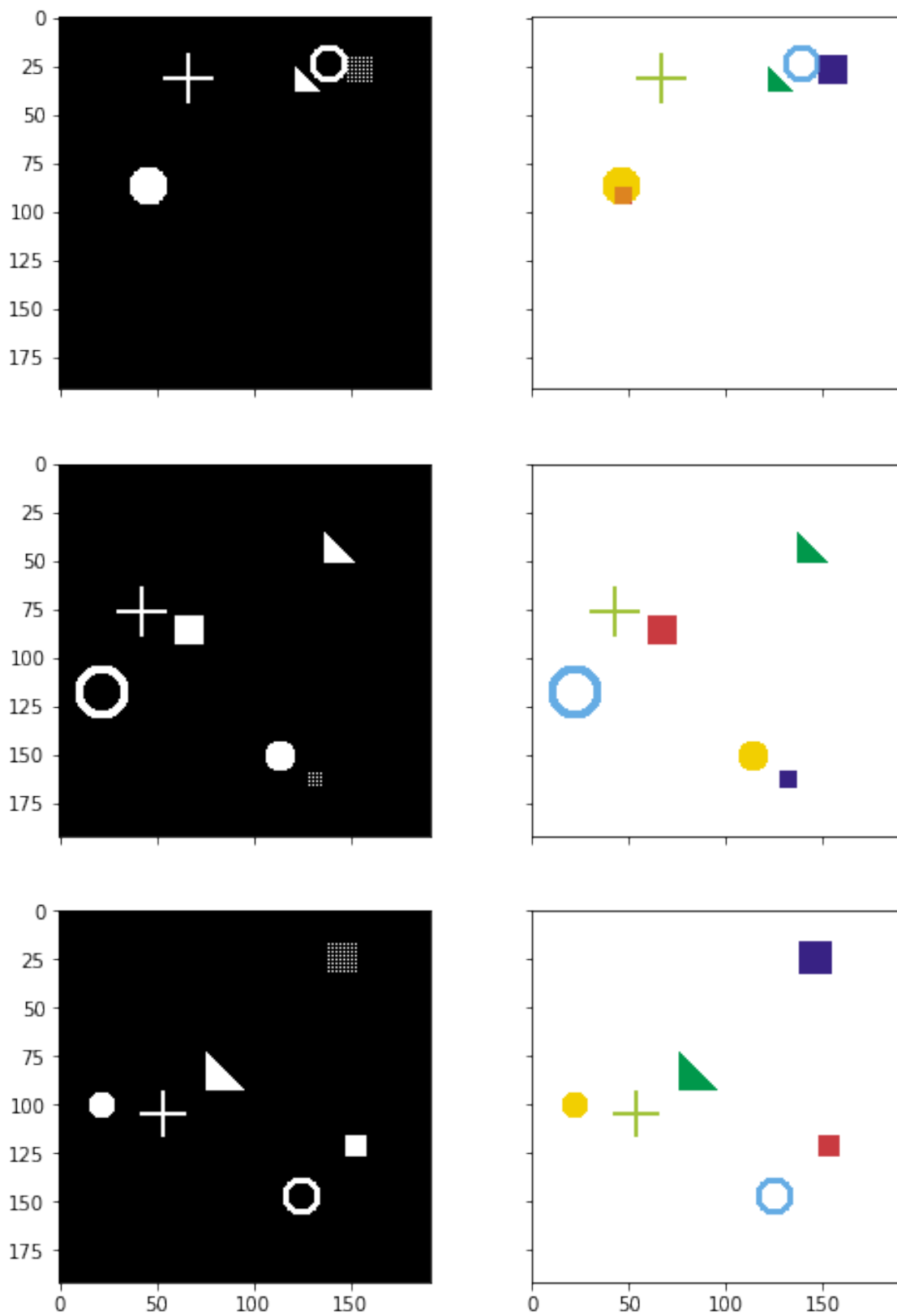
- U-Net: Convolutional Networks for Biomedical Image Segmentation
- Fully Convolutional Networks for Semantic Segmentation

Synthetic images/masks for training

First clone the repository and cd into the project directory.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import helper
4 import simulation
5
6 # Generate some random images
7 input_images, target_masks = simulation.generate_random_data(192, 192,
8     count=3)
9
10 for x in [input_images, target_masks]:
11     print(x.shape)
12     print(x.min(), x.max())
13
14 # Change channel-order and make 3 channels for matplotlib
15 input_images_rgb = [x.astype(np.uint8) for x in input_images]
16
17 # Map each channel (i.e. class) to each color
18 target_masks_rgb = [helper.masks_to_colorimg(x) for x in target_masks]
19
20 # Left: Input image (black and white), Right: Target mask (6ch)
21 helper.plot_side_by_side([input_images_rgb, target_masks_rgb])
```

Left: Input image (black and white), Right: Target mask (6ch)



Prepare Dataset and DataLoader

```
1 from torch.utils.data import Dataset, DataLoader
2 from torchvision import transforms, datasets, models
3
4 class SimDataset(Dataset):
5     def __init__(self, count, transform=None):
6         self.input_images, self.target_masks = simulation.
7             generate_random_data(192, 192, count=count)
8         self.transform = transform
9
10    def __len__(self):
11        return len(self.input_images)
12
13    def __getitem__(self, idx):
14        image = self.input_images[idx]
15        mask = self.target_masks[idx]
16        if self.transform:
17            image = self.transform(image)
18
19        return [image, mask]
20
21 # use the same transformations for train/val in this example
22 trans = transforms.Compose([
23     transforms.ToTensor(),
24     transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
25     # imagenet
26 ])
27
28 train_set = SimDataset(2000, transform = trans)
29 val_set = SimDataset(200, transform = trans)
30
31 image_datasets = {
32     'train': train_set, 'val': val_set
33 }
34
35 batch_size = 25
36
37 dataloaders = {
38     'train': DataLoader(train_set, batch_size=batch_size, shuffle=True,
39         num_workers=0),
40     'val': DataLoader(val_set, batch_size=batch_size, shuffle=True,
41         num_workers=0)
42 }
```

Check the outputs from DataLoader

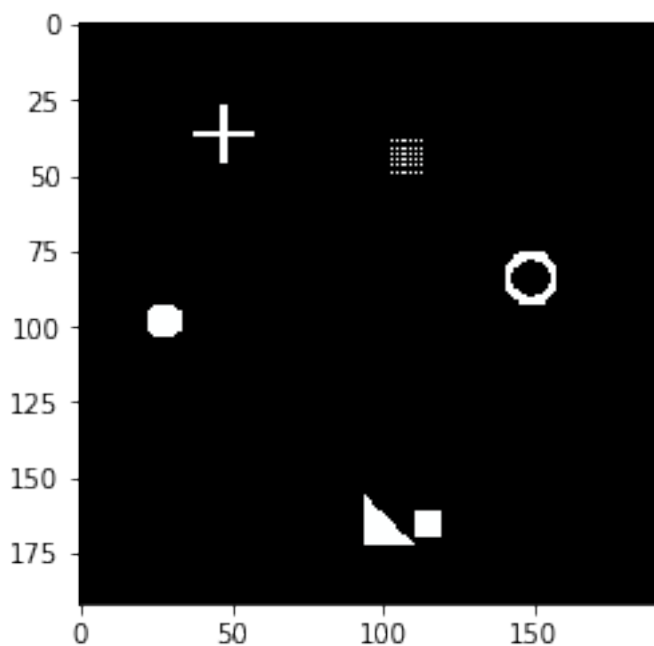
```
1 import torchvision.utils
```

```

2
3 def reverse_transform(inp):
4     inp = inp.numpy().transpose((1, 2, 0))
5     mean = np.array([0.485, 0.456, 0.406])
6     std = np.array([0.229, 0.224, 0.225])
7     inp = std * inp + mean
8     inp = np.clip(inp, 0, 1)
9     inp = (inp * 255).astype(np.uint8)
10
11     return inp
12
13 # Get a batch of training data
14 inputs, masks = next(iter(data loaders['train']))
15
16 print(inputs.shape, masks.shape)
17
18 plt.imshow(reverse_transform(inputs[3]))

```

```
1 torch.Size([25, 3, 192, 192]) torch.Size([25, 6, 192, 192])
```



Create the UNet module

```

1 import torch
2 import torch.nn as nn
3 from torchvision import models
4

```

```

5 def convrelu(in_channels, out_channels, kernel, padding):
6     return nn.Sequential(
7         nn.Conv2d(in_channels, out_channels, kernel, padding=padding),
8         nn.ReLU(inplace=True),
9     )
10
11
12 class ResNetUNet(nn.Module):
13     def __init__(self, n_class):
14         super().__init__()
15
16         self.base_model = models.resnet18(pretrained=True)
17         self.base_layers = list(self.base_model.children())
18
19         self.layer0 = nn.Sequential(*self.base_layers[:3]) # size=(N,
20             64, x.H/2, x.W/2)
21         self.layer0_1x1 = convrelu(64, 64, 1, 0)
22         self.layer1 = nn.Sequential(*self.base_layers[3:5]) # size=(N,
23             64, x.H/4, x.W/4)
24         self.layer1_1x1 = convrelu(64, 64, 1, 0)
25         self.layer2 = self.base_layers[5] # size=(N, 128, x.H/8, x.W
26             /8)
27         self.layer2_1x1 = convrelu(128, 128, 1, 0)
28         self.layer3 = self.base_layers[6] # size=(N, 256, x.H/16, x.W
29             /16)
30         self.layer3_1x1 = convrelu(256, 256, 1, 0)
31         self.layer4 = self.base_layers[7] # size=(N, 512, x.H/32, x.W
32             /32)
33         self.layer4_1x1 = convrelu(512, 512, 1, 0)
34
35         self.upsample = nn.Upsample(scale_factor=2, mode='bilinear',
36             align_corners=True)
37
38         self.conv_up3 = convrelu(256 + 512, 512, 3, 1)
39         self.conv_up2 = convrelu(128 + 512, 256, 3, 1)
40         self.conv_up1 = convrelu(64 + 256, 256, 3, 1)
41         self.conv_up0 = convrelu(64 + 256, 128, 3, 1)
42
43         self.conv_original_size0 = convrelu(3, 64, 3, 1)
44         self.conv_original_size1 = convrelu(64, 64, 3, 1)
45         self.conv_original_size2 = convrelu(64 + 128, 64, 3, 1)
46
47         self.conv_last = nn.Conv2d(64, n_class, 1)
48
49     def forward(self, input):
50         x_original = self.conv_original_size0(input)
51         x_original = self.conv_original_size1(x_original)
52
53         layer0 = self.layer0(input)
54         layer1 = self.layer1(layer0)
55         layer2 = self.layer2(layer1)

```

```

50     layer3 = self.layer3(layer2)
51     layer4 = self.layer4(layer3)
52
53     layer4 = self.layer4_1x1(layer4)
54     x = self.upsample(layer4)
55     layer3 = self.layer3_1x1(layer3)
56     x = torch.cat([x, layer3], dim=1)
57     x = self.conv_up3(x)
58
59     x = self.upsample(x)
60     layer2 = self.layer2_1x1(layer2)
61     x = torch.cat([x, layer2], dim=1)
62     x = self.conv_up2(x)
63
64     x = self.upsample(x)
65     layer1 = self.layer1_1x1(layer1)
66     x = torch.cat([x, layer1], dim=1)
67     x = self.conv_up1(x)
68
69     x = self.upsample(x)
70     layer0 = self.layer0_1x1(layer0)
71     x = torch.cat([x, layer0], dim=1)
72     x = self.conv_up0(x)
73
74     x = self.upsample(x)
75     x = torch.cat([x, x_original], dim=1)
76     x = self.conv_original_size2(x)
77
78     out = self.conv_last(x)
79
80     return out

```

Model summary

```

1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2 model = ResNetUNet(n_class=6)
3 model = model.to(device)
4
5 # check keras-like model summary using torchsummary
6 from torchsummary import summary
7 summary(model, input_size=(3, 224, 224))

```

1	-----		
2	Layer (type)	Output Shape	Param #
3	=====		
4	Conv2d-1	[-1, 64, 224, 224]	1,792
5	ReLU-2	[-1, 64, 224, 224]	0
6	Conv2d-3	[-1, 64, 224, 224]	36,928
7	ReLU-4	[-1, 64, 224, 224]	0

8	Conv2d-5	[-1, 64, 112, 112]	9,408
9	BatchNorm2d-6	[-1, 64, 112, 112]	128
10	ReLU-7	[-1, 64, 112, 112]	0
11	MaxPool2d-8	[-1, 64, 56, 56]	0
12	Conv2d-9	[-1, 64, 56, 56]	4,096
13	BatchNorm2d-10	[-1, 64, 56, 56]	128
14	ReLU-11	[-1, 64, 56, 56]	0
15	Conv2d-12	[-1, 64, 56, 56]	36,864
16	BatchNorm2d-13	[-1, 64, 56, 56]	128
17	ReLU-14	[-1, 64, 56, 56]	0
18	Conv2d-15	[-1, 256, 56, 56]	16,384
19	BatchNorm2d-16	[-1, 256, 56, 56]	512
20	Conv2d-17	[-1, 256, 56, 56]	16,384
21	BatchNorm2d-18	[-1, 256, 56, 56]	512
22	ReLU-19	[-1, 256, 56, 56]	0
23	Bottleneck-20	[-1, 256, 56, 56]	0
24	Conv2d-21	[-1, 64, 56, 56]	16,384
25	BatchNorm2d-22	[-1, 64, 56, 56]	128
26	ReLU-23	[-1, 64, 56, 56]	0
27	Conv2d-24	[-1, 64, 56, 56]	36,864
28	BatchNorm2d-25	[-1, 64, 56, 56]	128
29	ReLU-26	[-1, 64, 56, 56]	0
30	Conv2d-27	[-1, 256, 56, 56]	16,384
31	BatchNorm2d-28	[-1, 256, 56, 56]	512
32	ReLU-29	[-1, 256, 56, 56]	0
33	Bottleneck-30	[-1, 256, 56, 56]	0
34	Conv2d-31	[-1, 64, 56, 56]	16,384
35	BatchNorm2d-32	[-1, 64, 56, 56]	128
36	ReLU-33	[-1, 64, 56, 56]	0
37	Conv2d-34	[-1, 64, 56, 56]	36,864
38	BatchNorm2d-35	[-1, 64, 56, 56]	128
39	ReLU-36	[-1, 64, 56, 56]	0
40	Conv2d-37	[-1, 256, 56, 56]	16,384
41	BatchNorm2d-38	[-1, 256, 56, 56]	512
42	ReLU-39	[-1, 256, 56, 56]	0
43	Bottleneck-40	[-1, 256, 56, 56]	0
44	Conv2d-41	[-1, 128, 56, 56]	32,768
45	BatchNorm2d-42	[-1, 128, 56, 56]	256
46	ReLU-43	[-1, 128, 56, 56]	0
47	Conv2d-44	[-1, 128, 28, 28]	147,456
48	BatchNorm2d-45	[-1, 128, 28, 28]	256
49	ReLU-46	[-1, 128, 28, 28]	0
50	Conv2d-47	[-1, 512, 28, 28]	65,536
51	BatchNorm2d-48	[-1, 512, 28, 28]	1,024
52	Conv2d-49	[-1, 512, 28, 28]	131,072
53	BatchNorm2d-50	[-1, 512, 28, 28]	1,024
54	ReLU-51	[-1, 512, 28, 28]	0
55	Bottleneck-52	[-1, 512, 28, 28]	0
56	Conv2d-53	[-1, 128, 28, 28]	65,536
57	BatchNorm2d-54	[-1, 128, 28, 28]	256
58	ReLU-55	[-1, 128, 28, 28]	0

59	Conv2d-56	[-1, 128, 28, 28]	147,456
60	BatchNorm2d-57	[-1, 128, 28, 28]	256
61	ReLU-58	[-1, 128, 28, 28]	0
62	Conv2d-59	[-1, 512, 28, 28]	65,536
63	BatchNorm2d-60	[-1, 512, 28, 28]	1,024
64	ReLU-61	[-1, 512, 28, 28]	0
65	Bottleneck-62	[-1, 512, 28, 28]	0
66	Conv2d-63	[-1, 128, 28, 28]	65,536
67	BatchNorm2d-64	[-1, 128, 28, 28]	256
68	ReLU-65	[-1, 128, 28, 28]	0
69	Conv2d-66	[-1, 128, 28, 28]	147,456
70	BatchNorm2d-67	[-1, 128, 28, 28]	256
71	ReLU-68	[-1, 128, 28, 28]	0
72	Conv2d-69	[-1, 512, 28, 28]	65,536
73	BatchNorm2d-70	[-1, 512, 28, 28]	1,024
74	ReLU-71	[-1, 512, 28, 28]	0
75	Bottleneck-72	[-1, 512, 28, 28]	0
76	Conv2d-73	[-1, 128, 28, 28]	65,536
77	BatchNorm2d-74	[-1, 128, 28, 28]	256
78	ReLU-75	[-1, 128, 28, 28]	0
79	Conv2d-76	[-1, 128, 28, 28]	147,456
80	BatchNorm2d-77	[-1, 128, 28, 28]	256
81	ReLU-78	[-1, 128, 28, 28]	0
82	Conv2d-79	[-1, 512, 28, 28]	65,536
83	BatchNorm2d-80	[-1, 512, 28, 28]	1,024
84	ReLU-81	[-1, 512, 28, 28]	0
85	Bottleneck-82	[-1, 512, 28, 28]	0
86	Conv2d-83	[-1, 256, 28, 28]	131,072
87	BatchNorm2d-84	[-1, 256, 28, 28]	512
88	ReLU-85	[-1, 256, 28, 28]	0
89	Conv2d-86	[-1, 256, 14, 14]	589,824
90	BatchNorm2d-87	[-1, 256, 14, 14]	512
91	ReLU-88	[-1, 256, 14, 14]	0
92	Conv2d-89	[-1, 1024, 14, 14]	262,144
93	BatchNorm2d-90	[-1, 1024, 14, 14]	2,048
94	Conv2d-91	[-1, 1024, 14, 14]	524,288
95	BatchNorm2d-92	[-1, 1024, 14, 14]	2,048
96	ReLU-93	[-1, 1024, 14, 14]	0
97	Bottleneck-94	[-1, 1024, 14, 14]	0
98	Conv2d-95	[-1, 256, 14, 14]	262,144
99	BatchNorm2d-96	[-1, 256, 14, 14]	512
100	ReLU-97	[-1, 256, 14, 14]	0
101	Conv2d-98	[-1, 256, 14, 14]	589,824
102	BatchNorm2d-99	[-1, 256, 14, 14]	512
103	ReLU-100	[-1, 256, 14, 14]	0
104	Conv2d-101	[-1, 1024, 14, 14]	262,144
105	BatchNorm2d-102	[-1, 1024, 14, 14]	2,048
106	ReLU-103	[-1, 1024, 14, 14]	0
107	Bottleneck-104	[-1, 1024, 14, 14]	0
108	Conv2d-105	[-1, 256, 14, 14]	262,144
109	BatchNorm2d-106	[-1, 256, 14, 14]	512

110	ReLU-107	[-1, 256, 14, 14]	0
111	Conv2d-108	[-1, 256, 14, 14]	589,824
112	BatchNorm2d-109	[-1, 256, 14, 14]	512
113	ReLU-110	[-1, 256, 14, 14]	0
114	Conv2d-111	[-1, 1024, 14, 14]	262,144
115	BatchNorm2d-112	[-1, 1024, 14, 14]	2,048
116	ReLU-113	[-1, 1024, 14, 14]	0
117	Bottleneck-114	[-1, 1024, 14, 14]	0
118	Conv2d-115	[-1, 256, 14, 14]	262,144
119	BatchNorm2d-116	[-1, 256, 14, 14]	512
120	ReLU-117	[-1, 256, 14, 14]	0
121	Conv2d-118	[-1, 256, 14, 14]	589,824
122	BatchNorm2d-119	[-1, 256, 14, 14]	512
123	ReLU-120	[-1, 256, 14, 14]	0
124	Conv2d-121	[-1, 1024, 14, 14]	262,144
125	BatchNorm2d-122	[-1, 1024, 14, 14]	2,048
126	ReLU-123	[-1, 1024, 14, 14]	0
127	Bottleneck-124	[-1, 1024, 14, 14]	0
128	Conv2d-125	[-1, 256, 14, 14]	262,144
129	BatchNorm2d-126	[-1, 256, 14, 14]	512
130	ReLU-127	[-1, 256, 14, 14]	0
131	Conv2d-128	[-1, 256, 14, 14]	589,824
132	BatchNorm2d-129	[-1, 256, 14, 14]	512
133	ReLU-130	[-1, 256, 14, 14]	0
134	Conv2d-131	[-1, 1024, 14, 14]	262,144
135	BatchNorm2d-132	[-1, 1024, 14, 14]	2,048
136	ReLU-133	[-1, 1024, 14, 14]	0
137	Bottleneck-134	[-1, 1024, 14, 14]	0
138	Conv2d-135	[-1, 256, 14, 14]	262,144
139	BatchNorm2d-136	[-1, 256, 14, 14]	512
140	ReLU-137	[-1, 256, 14, 14]	0
141	Conv2d-138	[-1, 256, 14, 14]	589,824
142	BatchNorm2d-139	[-1, 256, 14, 14]	512
143	ReLU-140	[-1, 256, 14, 14]	0
144	Conv2d-141	[-1, 1024, 14, 14]	262,144
145	BatchNorm2d-142	[-1, 1024, 14, 14]	2,048
146	ReLU-143	[-1, 1024, 14, 14]	0
147	Bottleneck-144	[-1, 1024, 14, 14]	0
148	Conv2d-145	[-1, 512, 14, 14]	524,288
149	BatchNorm2d-146	[-1, 512, 14, 14]	1,024
150	ReLU-147	[-1, 512, 14, 14]	0
151	Conv2d-148	[-1, 512, 7, 7]	2,359,296
152	BatchNorm2d-149	[-1, 512, 7, 7]	1,024
153	ReLU-150	[-1, 512, 7, 7]	0
154	Conv2d-151	[-1, 2048, 7, 7]	1,048,576
155	BatchNorm2d-152	[-1, 2048, 7, 7]	4,096
156	Conv2d-153	[-1, 2048, 7, 7]	2,097,152
157	BatchNorm2d-154	[-1, 2048, 7, 7]	4,096
158	ReLU-155	[-1, 2048, 7, 7]	0
159	Bottleneck-156	[-1, 2048, 7, 7]	0
160	Conv2d-157	[-1, 512, 7, 7]	1,048,576

161	BatchNorm2d-158	[-1, 512, 7, 7]	1,024
162	ReLU-159	[-1, 512, 7, 7]	0
163	Conv2d-160	[-1, 512, 7, 7]	2,359,296
164	BatchNorm2d-161	[-1, 512, 7, 7]	1,024
165	ReLU-162	[-1, 512, 7, 7]	0
166	Conv2d-163	[-1, 2048, 7, 7]	1,048,576
167	BatchNorm2d-164	[-1, 2048, 7, 7]	4,096
168	ReLU-165	[-1, 2048, 7, 7]	0
169	Bottleneck-166	[-1, 2048, 7, 7]	0
170	Conv2d-167	[-1, 512, 7, 7]	1,048,576
171	BatchNorm2d-168	[-1, 512, 7, 7]	1,024
172	ReLU-169	[-1, 512, 7, 7]	0
173	Conv2d-170	[-1, 512, 7, 7]	2,359,296
174	BatchNorm2d-171	[-1, 512, 7, 7]	1,024
175	ReLU-172	[-1, 512, 7, 7]	0
176	Conv2d-173	[-1, 2048, 7, 7]	1,048,576
177	BatchNorm2d-174	[-1, 2048, 7, 7]	4,096
178	ReLU-175	[-1, 2048, 7, 7]	0
179	Bottleneck-176	[-1, 2048, 7, 7]	0
180	Conv2d-177	[-1, 1024, 7, 7]	2,098,176
181	ReLU-178	[-1, 1024, 7, 7]	0
182	Upsample-179	[-1, 1024, 14, 14]	0
183	Conv2d-180	[-1, 512, 14, 14]	524,800
184	ReLU-181	[-1, 512, 14, 14]	0
185	Conv2d-182	[-1, 512, 14, 14]	7,078,400
186	ReLU-183	[-1, 512, 14, 14]	0
187	Upsample-184	[-1, 512, 28, 28]	0
188	Conv2d-185	[-1, 512, 28, 28]	262,656
189	ReLU-186	[-1, 512, 28, 28]	0
190	Conv2d-187	[-1, 512, 28, 28]	4,719,104
191	ReLU-188	[-1, 512, 28, 28]	0
192	Upsample-189	[-1, 512, 56, 56]	0
193	Conv2d-190	[-1, 256, 56, 56]	65,792
194	ReLU-191	[-1, 256, 56, 56]	0
195	Conv2d-192	[-1, 256, 56, 56]	1,769,728
196	ReLU-193	[-1, 256, 56, 56]	0
197	Upsample-194	[-1, 256, 112, 112]	0
198	Conv2d-195	[-1, 64, 112, 112]	4,160
199	ReLU-196	[-1, 64, 112, 112]	0
200	Conv2d-197	[-1, 128, 112, 112]	368,768
201	ReLU-198	[-1, 128, 112, 112]	0
202	Upsample-199	[-1, 128, 224, 224]	0
203	Conv2d-200	[-1, 64, 224, 224]	110,656
204	ReLU-201	[-1, 64, 224, 224]	0
205	Conv2d-202	[-1, 6, 224, 224]	390
206	=====		
207	Total params: 40,549,382		
208	Trainable params: 40,549,382		
209	Non-trainable params: 0		
210	-----		

Define the main training loop

```
1 from collections import defaultdict
2 import torch.nn.functional as F
3 from loss import dice_loss
4
5 def calc_loss(pred, target, metrics, bce_weight=0.5):
6     bce = F.binary_cross_entropy_with_logits(pred, target)
7
8     pred = F.sigmoid(pred)
9     dice = dice_loss(pred, target)
10
11     loss = bce * bce_weight + dice * (1 - bce_weight)
12
13     metrics['bce'] += bce.data.cpu().numpy() * target.size(0)
14     metrics['dice'] += dice.data.cpu().numpy() * target.size(0)
15     metrics['loss'] += loss.data.cpu().numpy() * target.size(0)
16
17     return loss
18
19 def print_metrics(metrics, epoch_samples, phase):
20     outputs = []
21     for k in metrics.keys():
22         outputs.append("{}: {:.4f}".format(k, metrics[k] / epoch_samples))
23
24     print("{}: {}".format(phase, ", ".join(outputs)))
25
26 def train_model(model, optimizer, scheduler, num_epochs=25):
27     best_model_wts = copy.deepcopy(model.state_dict())
28     best_loss = 1e10
29
30     for epoch in range(num_epochs):
31         print('Epoch {}/{}'.format(epoch, num_epochs - 1))
32         print('-' * 10)
33
34         since = time.time()
35
36         # Each epoch has a training and validation phase
37         for phase in ['train', 'val']:
38             if phase == 'train':
39                 scheduler.step()
40                 for param_group in optimizer.param_groups:
41                     print("LR", param_group['lr'])
42
43                 model.train() # Set model to training mode
44             else:
45                 model.eval() # Set model to evaluate mode
46
47         metrics = defaultdict(float)
```

```

48         epoch_samples = 0
49
50         for inputs, labels in dataloaders[phase]:
51             inputs = inputs.to(device)
52             labels = labels.to(device)
53
54             # zero the parameter gradients
55             optimizer.zero_grad()
56
57             # forward
58             # track history if only in train
59             with torch.set_grad_enabled(phase == 'train'):
60                 outputs = model(inputs)
61                 loss = calc_loss(outputs, labels, metrics)
62
63             # backward + optimize only if in training phase
64             if phase == 'train':
65                 loss.backward()
66                 optimizer.step()
67
68             # statistics
69             epoch_samples += inputs.size(0)
70
71             print_metrics(metrics, epoch_samples, phase)
72             epoch_loss = metrics['loss'] / epoch_samples
73
74             # deep copy the model
75             if phase == 'val' and epoch_loss < best_loss:
76                 print("saving best model")
77                 best_loss = epoch_loss
78                 best_model_wts = copy.deepcopy(model.state_dict())
79
80             time_elapsed = time.time() - since
81             print('{:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed
82                                     % 60))
83
84             print('Best val loss: {:.4f}'.format(best_loss))
85
86             # load best model weights
87             model.load_state_dict(best_model_wts)
88             return model

```

Training

```

1 import torch
2 import torch.optim as optim
3 from torch.optim import lr_scheduler
4 import time
5 import copy

```

```

6
7 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
8 print(device)
9
10 num_class = 6
11 model = ResNetUNet(num_class).to(device)
12
13 # freeze backbone layers
14 #for l in model.base_layers:
15 #    for param in l.parameters():
16 #        param.requires_grad = False
17
18 optimizer_ft = optim.Adam(filter(lambda p: p.requires_grad, model.
    parameters()), lr=1e-4)
19
20 exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=30,
    gamma=0.1)
21
22 model = train_model(model, optimizer_ft, exp_lr_scheduler, num_epochs
    =60)

```

```

1 cuda:0
2 Epoch 0/59
3 -----
4 LR 0.0001
5 train: bce: 0.070256, dice: 0.856320, loss: 0.463288
6 val: bce: 0.014897, dice: 0.515814, loss: 0.265356
7 saving best model
8 0m 51s
9 Epoch 1/59
10 -----
11 LR 0.0001
12 train: bce: 0.011369, dice: 0.309445, loss: 0.160407
13 val: bce: 0.003790, dice: 0.113682, loss: 0.058736
14 saving best model
15 0m 51s
16 Epoch 2/59
17 -----
18 LR 0.0001
19 train: bce: 0.003480, dice: 0.089928, loss: 0.046704
20 val: bce: 0.002525, dice: 0.067604, loss: 0.035064
21 saving best model
22 0m 51s
23
24 (Omitted)
25
26 Epoch 57/59
27 -----
28 LR 1e-05
29 train: bce: 0.000523, dice: 0.010289, loss: 0.005406
30 val: bce: 0.001558, dice: 0.030965, loss: 0.016261

```

```
31 0m 51s
32 Epoch 58/59
33 -----
34 LR 1e-05
35 train: bce: 0.000518, dice: 0.010209, loss: 0.005364
36 val: bce: 0.001548, dice: 0.031034, loss: 0.016291
37 0m 51s
38 Epoch 59/59
39 -----
40 LR 1e-05
41 train: bce: 0.000518, dice: 0.010168, loss: 0.005343
42 val: bce: 0.001566, dice: 0.030785, loss: 0.016176
43 0m 50s
44 Best val loss: 0.016171
```

Use the trained model

```
1 import math
2
3 model.eval() # Set model to the evaluation mode
4
5 # Create another simulation dataset for test
6 test_dataset = SimDataset(3, transform = trans)
7 test_loader = DataLoader(test_dataset, batch_size=3, shuffle=False,
8                           num_workers=0)
9
10 # Get the first batch
11 inputs, labels = next(iter(test_loader))
12 inputs = inputs.to(device)
13 labels = labels.to(device)
14
15 # Predict
16 pred = model(inputs)
17 # The loss functions include the sigmoid function.
18 pred = F.sigmoid(pred)
19 pred = pred.data.cpu().numpy()
20 print(pred.shape)
21
22 # Change channel-order and make 3 channels for matplotlib
23 input_images_rgb = [reverse_transform(x) for x in inputs.cpu()]
24
25 # Map each channel (i.e. class) to each color
26 target_masks_rgb = [helper.masks_to_colorimg(x) for x in labels.cpu().
27                     numpy()]
28 pred_rgb = [helper.masks_to_colorimg(x) for x in pred]
29
30 helper.plot_side_by_side([input_images_rgb, target_masks_rgb, pred_rgb
31 ])
```

1 (3, 6, 192, 192)

Left: Input image, Middle: Correct mask (Ground-truth), Right: Predicted mask

