

note1

November 23, 2025

- batch_size=4, resize=0.5, epoch=5.
- batch_size=1, resize=1, epoch=5.

0.0.1

```
cuda12.1 pip install torch==2.3.0 torchvision==0.18.0 torchaudio==2.3.0 --index-url
https://download.pytorch.org/whl/cu121
```

```
cuda11.7 pip install torch==1.13.1+cu117 torchvision==0.14.1+cu117 torchaudio==0.13.1 --extra-
index-url https://download.pytorch.org/whl/cu117
```

- trian.py
- scale=0.25 (20161512 => 504303), 1
- batch_size (<=4)
- scale=0.25, batch size=4
-

```
conda create --name cuda117 python=3.10
```

```
pip install torch==1.13.1+cu117 torchvision==0.14.1+cu117 torchaudio==0.13.1 --extra-index-url
```

```
pip install -r requirements.txt
```

```
python .\train.py
```

```
[7]: # hash
import hashlib
import time
# hash_time = time.time()
# hash_time = str(hash_time).encode('utf-8')
# hash_time = hashlib.md5(hash_time).hexdigest()
hash_time = hashlib.md5(str(time.time()).encode('utf-8')).hexdigest()
print(hash_time)
```

```
8d505b8bba8af9e73ebb5815bea1401a
```

```
[20]: #
import time
date = time.strftime('%Y-%m-%d', time.localtime())
t = time.strftime('%H%M%S', time.localtime())
file_name = '{}T{}.txt'.format(date, t)
print(file_name)
```

```
2024-07-19T221247.txt
```

```
[7]: #
import torch
from torchvision import models
from unet import UNet
from utils.data_loading import BasicDataset, CarvanaDataset
from utils.dice_score import dice_loss

file_path = './checkpoints/checkpoint_epoch1.pth'
model = UNet(n_channels=3, n_classes=18, bilinear=False)
model = model.to(memory_format=torch.channels_last)

state_dict = torch.load(file_path)
del state_dict['mask_values']

model.load_state_dict(state_dict)
print(model)

UNet(
  (inc): DoubleConv(
    (double_conv): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
    )
  )
  (down1): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (1): DoubleConv(
        (double_conv): Sequential(
          (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (5): ReLU(inplace=True)
```

```

        )
    )
)
)
(down2): Down(
    (maxpool_conv): Sequential(
        (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (1): DoubleConv(
            (double_conv): Sequential(
                (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
                (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
                (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
                (5): ReLU(inplace=True)
            )
        )
    )
)
)
(down3): Down(
    (maxpool_conv): Sequential(
        (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (1): DoubleConv(
            (double_conv): Sequential(
                (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
                (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
                (2): ReLU(inplace=True)
                (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
                (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
                (5): ReLU(inplace=True)
            )
        )
    )
)
)
(down4): Down(
    (maxpool_conv): Sequential(
        (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)

```

```

        (1): DoubleConv(
          (double_conv): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU(inplace=True)
            (3): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (5): ReLU(inplace=True)
          )
        )
      )
    )
  )
  (up1): Up(
    (up): ConvTranspose2d(1024, 512, kernel_size=(2, 2), stride=(2, 2))
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )
  (up2): Up(
    (up): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )

```

```

    )
)
(up3): Up(
  (up): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (conv): DoubleConv(
    (double_conv): Sequential(
      (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
    )
  )
)
(up4): Up(
  (up): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv): DoubleConv(
    (double_conv): Sequential(
      (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
    )
  )
)
(outc): OutConv(
  (conv): Conv2d(64, 18, kernel_size=(1, 1), stride=(1, 1))
)
)

```

0.0.2

- 18

```

asphalt 1 #e6194b  other-terrain
dirt 2 #3cb44b
mud 3 #ffe119

```

```

water 4 #0082c8
gravel 5 #911eb4
other-terrain 6 #46f0f0
tree-trunk 7 #f032e6
tree-foliage 8 #d2f53c
bush 9 #fabebe
fence 10 #008080
structure 11 #aa6e28
pole 12 #fffac8 other-object
vehicle 13 #800000 (-> 0)
rock 14 #aaffc3
log 15 #808000
other-object 16 #ffd7b4
sky 17 #000080
grass 18 #808080

```

•

```

merge_classes = {
    1: 6, # asphalt -> other-terrain
    12: 16, # pole -> other-object
    13: 0, # exclude from evaluation
}

```

```

[3]: origin_classes = ['bg', 'asphalt', 'dirt', 'mud', 'water', 'gravel',
    ↪ 'other-terrain', 'tree-trunk', 'tree-foliage', 'brush', 'fence',
    ↪ 'structure', 'pole', 'vehicle', 'rock', 'log', 'other-object', 'sky',
    ↪ 'grass']
merge_dict = {1:6, 12:16, 13:0}
def compute_map(nb_of_classes, merge_dict):
    nb_of_classes += len(merge_dict)
    map_list = [i for i in range(nb_of_classes)]
    shift = 0
    for i in range(nb_of_classes):
        if i in merge_dict:
            shift += 1
        map_list[i] -= shift
    for k, v in merge_dict.items():
        map_list[k] = map_list[v]
    return map_list

original_dict = {origin_classes[i]: i for i in range(len(origin_classes))}
print(original_dict)
""" # 18      (      0) 1-18
{'asphalt': 1, 'dirt': 2, 'mud': 3, 'water': 4, 'gravel': 5, 'other-terrain':
↪ 6, 'tree-trunk': 7, 'tree-foliage': 8, 'brush': 9, 'fence': 10, 'structure':
↪ 11, 'pole': 12, 'vehicle': 13, 'rock': 14, 'log': 15, 'other-object': 16,
↪ 'sky': 17, 'grass': 18}

```

```

"""
new_map = compute_map(16, merge_dict)
print(new_map)
new_dict = {origin_classes[i]: new_map[i] for i in range(len(origin_classes))}
print(new_dict)
""" #      16      1-18 -> 0-15
{'bg': 0, 'asphalt': 5, 'dirt': 1, 'mud': 2, 'water': 3, 'gravel': 4,
  ↪ 'other-terrain': 5, 'tree-trunk': 6, 'tree-foliage': 7, 'brush': 8, 'fence':
  ↪ 9, 'structure': 10, 'pole': 13, 'vehicle': 0, 'rock': 11, 'log': 12,
  ↪ 'other-object': 13, 'sky': 14, 'grass': 15}
"""

```

```

{'bg': 0, 'asphalt': 1, 'dirt': 2, 'mud': 3, 'water': 4, 'gravel': 5, 'other-
terrain': 6, 'tree-trunk': 7, 'tree-foliage': 8, 'brush': 9, 'fence': 10,
'structure': 11, 'pole': 12, 'vehicle': 13, 'rock': 14, 'log': 15, 'other-
object': 16, 'sky': 17, 'grass': 18}
[0, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 0, 11, 12, 13, 14, 15]
{'bg': 0, 'asphalt': 5, 'dirt': 1, 'mud': 2, 'water': 3, 'gravel': 4, 'other-
terrain': 5, 'tree-trunk': 6, 'tree-foliage': 7, 'brush': 8, 'fence': 9,
'structure': 10, 'pole': 13, 'vehicle': 0, 'rock': 11, 'log': 12, 'other-
object': 13, 'sky': 14, 'grass': 15}

```

```

[3]: " #      16      1-18 -> 0-15\n{'bg': 0, 'asphalt': 5, 'dirt': 1, 'mud':
2, 'water': 3, 'gravel': 4, 'other-terrain': 5, 'tree-trunk': 6, 'tree-foliage':
7, 'brush': 8, 'fence': 9, 'structure': 10, 'pole': 13, 'vehicle': 0, 'rock':
11, 'log': 12, 'other-object': 13, 'sky': 14, 'grass': 15}\n"

```

0.0.3 evaluate.py

```

def evaluate(model, val_loader, device, amp):
    model.eval()
    n_val = len(val_loader) # the number of batch
    accuracys = []
    ious = []
    precisions = []
    recalls = []

    with torch.no_grad(): #
        for batch in tqdm(val_loader, total=n_val, desc='Validation round', unit='batch', leave=False):
            images, mask_true = batch['image'], batch['mask']
            images = images.to(device=device, dtype=torch.float32, memory_format=torch.channels_last)
            mask_true = mask_true.to(device=device, dtype=torch.long)

            with torch.autocast(device.type if device.type != 'mps' else 'cpu', enabled=amp):
                mask_pred = model(images)
                assert mask_true.min() >= 0 and mask_true.max() < model.n_classes, 'True mask is out of range'
                # convert to one-hot format

```

```

        mask_true = F.one_hot(mask_true, model.n_classes).permute(0, 3, 1, 2).float()
        mask_pred = F.one_hot(mask_pred.argmax(dim=1), model.n_classes).permute(0, 3, 1, 2).float()
        accuracy, ious, precisions, recalls = calculate_metrics(mask_pred[:, 1:], mask_true[:, 1:])
        # accuracys_mean = np.nanmean(np.array(accuracys), axis=0)
        ious_mean = np.nanmean(np.array(ious), axis=0)
        precisions_mean = np.nanmean(np.array(precisions), axis=0)
        recalls_mean = np.nanmean(np.array(recalls), axis=0)

    return accuracy, ious_mean, precisions_mean, recalls_mean

• mask_pred = F.one_hot(mask_pred.argmax(dim=1), model.n_classes).permute(0,
    3, 1, 2).float()
    - mask_pred:          shape:(batch_size, height, width)
    - F.one_hot(mask_pred.argmax(dim=1), model.n_classes):          one-
      hot shape:(batch_size, n_classes, height, width)

```

```

[9]: import torch
      # batch_size = 1
      # num_classes = 3
      # height = 2
      # width = 3
      # predict shape: batch_size * num_classes * height * width
      mask_pred = torch.tensor([
          [
              [[0.1, 0.4, 0.1], [0.4, 0.5, 0.1]],
              [[0.3, 0.5, 0.6], [0.6, 0.3, 0.5]],
              [[0.6, 0.1, 0.3], [0.0, 0.1, 0.4]]
          ],
      ])
      print(mask_pred.shape)

```

```

torch.Size([1, 3, 2, 3])

```

```

[13]: import torch
      import torch.nn.functional as F
      # argmax
      # shape = (batch_size, height, width)
      indices = mask_pred.argmax(dim=1)
      # ans = F.one_hot(mask_pred.argmax(dim=1), 3).permute(0, 3, 1, 2).float()
      print(indices)
      ans = F.one_hot(indices, 3)
      # print(ans)
      # print(indices)
      print(ans)

```

```

tensor([[[[2, 1, 1],
           [1, 0, 1]]]])
tensor([[[[0, 0, 1],
           [0, 1, 0],

```

```

        [0, 1, 0]],

        [[0, 1, 0],
         [1, 0, 0],
         [0, 1, 0]]]])

```

```

[17]: import torch
mask_pred = torch.tensor([
    [
        [[0.1, 0.4, 0.1], [0.4, 0.5, 0.1]],
        [[0.3, 0.5, 0.6], [0.6, 0.3, 0.5]],
        [[0.6, 0.1, 0.3], [0.0, 0.1, 0.4]],
    ],
    [
        [[0.1, 0.4, 0.1], [0.4, 0.5, 0.1]],
        [[0.3, 0.5, 0.6], [0.6, 0.3, 0.5]],
        [[0.6, 0.1, 0.3], [0.0, 0.1, 0.4]],
    ],
])
mask_true = torch.tensor([
    [
        [2, 1, 0],
        [1, 2, 0],
    ],
    [
        [1, 2, 0],
        [0, 1, 2],
    ],
])
print(mask_pred.shape)
indices = mask_pred.argmax(dim=1)
print(indices)
print(len(indices))

```

```

torch.Size([2, 3, 2, 3])
tensor([[[[2, 1, 1],
          [1, 0, 1]],

         [[2, 1, 1],
          [1, 0, 1]]]])

```

2

```

[3]: print(indices.shape)
print(len(indices))
print(len(indices[0]))
print(len(indices[0][0]))

```

```

torch.Size([2, 2, 3])
2

```

2
3

```
[24]: # One-hot
import torch
import torch.nn.functional as F

# shape = (batch_size, height, width, num_classes)
# permute (batch_size, num_classes, height, width)
mask_pred = F.one_hot(indices, 3).permute(0, 3, 1, 2).float()
print(mask_pred)
print(mask_pred.shape)
n_classes = mask_pred.shape[1]
print(n_classes)
mask_pred = mask_pred.flatten()
print(mask_pred)
print(len(mask_pred))
```

```
tensor([[[[0., 0., 0.],
          [0., 1., 0.]],

        [[0., 1., 1.],
          [1., 0., 1.]],

        [[1., 0., 0.],
          [0., 0., 0.]]],

       [[0., 0., 0.],
        [0., 1., 0.]],

        [[0., 1., 1.],
          [1., 0., 1.]],

        [[1., 0., 0.],
          [0., 0., 0.]]]])
torch.Size([2, 3, 2, 3])
3
tensor([0., 0., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0., 0.])
36
```

```
[29]: indices = indices.flatten()
print(indices)
ignore_index = [0, 2]
# indices ignore_index
counts = sum([1 if i not in ignore_index else 0 for i in indices])
print(counts)
```

```
count = torch.tensor([1 if i not in ignore_index else 0 for i in indices])
print(sum(count))
print(count)
```

```
tensor([2, 1, 1, 1, 0, 1, 2, 1, 1, 1, 0, 1])
8
tensor(8)
tensor([0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1])
```

[16]:

```
-----
RuntimeError                                Traceback (most recent call last)
Cell In[16], line 5
      2 import torch
      3 import torch.nn.functional as F
----> 5 mask_pred = F.one_hot(indices, 3).permute(0, 3, 1, 2).float()
      6 print(mask_pred)
```

```
RuntimeError: permute(sparse_coo): number of dimensions in the tensor input does
↳ not match the length of the desired ordering of dimensions i.e. input.dim() =
↳ 2 is not equal to len(dims) = 4
```

[8]: `# indices.shape[0] * indices.shape[1] * indices.shape[2]`

[13]:

```
pred_positive = (indices == 2)
print(pred_positive)
actual_positive = (mask_true == 2)
print(actual_positive)
```

```
tensor([[[ True, False, False],
         [False, False, False]],

        [[ True, False, False],
         [False, False, False]]])
tensor([[[ True, False, False],
         [False,  True, False]],

        [[False,  True, False],
         [False, False,  True]])])
```

[1]:

```
true_positive = (pred_positive & actual_positive).sum().item()
print(true_positive)
```

```
-----
NameError                                Traceback (most recent call last)
```

Cell In[1], line 1

```
----> 1 true_positive = (pred_positive & actual_positive).sum().item()
      2 print(true_positive)
```

NameError: name 'pred_positive' is not defined

```
[16]: union = torch.logical_or(pred_positive, actual_positive).sum().item()
      false_positive = pred_positive.sum().item() - true_positive
      false_negative = actual_positive.sum().item() - true_positive
      print(union, false_positive, false_negative)
```

5 1 3

0.0.4

```
[21]: pixel_counts = torch.tensor([2.3e5, 2.5e9, 1.5e7, 5.8e7, 1.1e8, 4.64e6, 3.9e9,
    ↪ 1.6e10, 3.4e8, 1e7, 9.8e7, 1.6e7, 1.2e8, 3.15e7, 2e9, 3e9])
      pixel_counts = pixel_counts[1:]

      # relative_weights
      total_pixel_count = pixel_counts.sum()
      num_classes = len(pixel_counts)
      relative_weights = total_pixel_count / (num_classes * pixel_counts)
      relative_weights = relative_weights / relative_weights.max() # normalize
      print(relative_weights)

      # log weights
      epsilon = 1e-6
      log_weights = torch.log(total_pixel_count / (pixel_counts + epsilon))
      log_weights = log_weights / log_weights.max() # normalize
      print(log_weights)
```

```
tensor([1.8560e-03, 3.0933e-01, 8.0000e-02, 4.2182e-02, 1.0000e+00, 1.1897e-03,
        2.9000e-04, 1.3647e-02, 4.6400e-01, 4.7347e-02, 2.9000e-01, 3.8667e-02,
        1.4730e-01, 2.3200e-03, 1.5467e-03])
tensor([0.2781, 0.8653, 0.7101, 0.6366, 1.0000, 0.2271, 0.0651, 0.5071, 0.9119,
        0.6499, 0.8579, 0.6267, 0.7802, 0.3037, 0.2572])
```

[]: