varify_model

November 23, 2025

0.0.1

```
[8]: from utils.data_loading import WSDataset
from unet import UNet
from utils.utils import plot_img_and_mask
import torch
import torch.nn.functional as F
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from torchvision import transforms

def predict_img(net,
                full_img,
                device,
                scale_factor=1,
                out_threshold=0.5):
    net.eval()
    new_size = (full_img.size[0] * scale_factor, full_img.size[1] *␣
 ↪scale_factor)
    im_transform = transforms.Compose([
        ImageResize(new_size, interpolate_mode=Image.BICUBIC),
        ImageNormalization(),
        transforms.ToTensor()
    ])
    img = im_transform(full_img).unsqueeze(0)
    img = img.to(device=device, dtype=torch.float32)
    print(img.shape)

    with torch.no_grad():
        output = net(img).cpu()
        output = F.interpolate(output, (full_img.size[1], full_img.size[0]),␣
 ↪mode='bilinear')
        mask = output.argmax(dim=1)
    return mask[0].long().squeeze().numpy()
```

```
---------------------------------------------------------------------------
```

```
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[8], line 3
      1 from utils.data_loading import WSDataset
      2 from unet import UNet
----> 3 from utils.utils import plot_img_and_mask
      4 import torch
      5 import torch.nn.functional as F

ModuleNotFoundError: No module named 'utils.utils'
```

[9]:

```python
import matplotlib.pyplot as plt
import pandas as pd
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
print(device)
checkpoint_path = './checkpoints/checkpoint_epoch7.pth'
net = UNet(n_channels=3, n_classes=16, bilinear=False)
net.to(device=device)

state_dict = torch.load(checkpoint_path, map_location=device)
mask_values = state_dict.pop('mask_values', [0, 1])
net.load_state_dict(state_dict)
test_dir = Path('.') / '..' / '..' / '..' / 'data' / 'test1'

# test_input_dir = test_dir / 'image'
# test_gt_dir = test_dir / 'indexLabel'
# inputs = test_input_dir.glob('*.png')
data_dir = Path('.') / '..' / '..' / 'data'
val_csv = pd.read_csv(data_dir / 'val.csv')
val_image_paths = val_csv['im_path'].values
val_label_paths = val_csv['label_path'].values

i = 0

# print(list(inputs))
for i in range(len(val_image_paths)):
    # gt_path = test_gt_dir / input_path.name
    gt_path = data_dir / val_label_paths[i]
    gt = Image.open(gt_path)
    image = Image.open(data_dir / val_image_paths[i])
    mask = predict_img(net=net,
                       full_img=image,
                       scale_factor=1,
                       out_threshold=0.5,
                       device=device)
    # print('mask true ', np.unique(np.array(gt)))
    # print('  mask  ', np.unique(np.array(mask)))
    plt.subplot(1, 3, 1)
    plt.imshow(image)
    plt.subplot(1, 3, 2)
    plt.imshow(mask, cmap='gray')
    plt.subplot(1, 3, 3)
    plt.imshow(gt, cmap='gray')
    plt.show()
    i += 1
    if i == 5:
        break
```

```
mps
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[9], line 10
      7 net = UNet(n_channels=3, n_classes=16, bilinear=False)
      8 net.to(device=device)
---> 10 state_dict = torch.load(checkpoint_path, map_location=device)
     11 mask_values = state_dict.pop('mask_values', [0, 1])
     12 net.load_state_dict(state_dict)

File ~/anaconda3/envs/comp9517_python310/lib/python3.11/site-packages/torch/
 ↪serialization.py:997, in load(f, map_location, pickle_module, weights_only,␣
 ↪mmap, **pickle_load_args)
    994 if 'encoding' not in pickle_load_args.keys():
    995     pickle_load_args['encoding'] = 'utf-8'
--> 997 with _open_file_like(f, 'rb') as opened_file:
    998     if _is_zipfile(opened_file):
    999         # The zipfile reader is going to advance the current file␣
 ↪position.
   1000         # If we want to actually tail call to torch.jit.load, we need t
   1001         # reset back to the original position.
   1002         orig_position = opened_file.tell()

File ~/anaconda3/envs/comp9517_python310/lib/python3.11/site-packages/torch/
 ↪serialization.py:444, in _open_file_like(name_or_buffer, mode)
    442 def _open_file_like(name_or_buffer, mode):
    443     if _is_path(name_or_buffer):
--> 444         return _open_file(name_or_buffer, mode)
    445     else:
    446         if 'w' in mode:

File ~/anaconda3/envs/comp9517_python310/lib/python3.11/site-packages/torch/
 ↪serialization.py:425, in _open_file.__init__(self, name, mode)
    424 def __init__(self, name, mode):
--> 425     super().__init__(open(name, mode))
```

```
FileNotFoundError: [Errno 2] No such file or directory: './checkpoints/
 ↪checkpoint_epoch7.pth'
```

[ ]:

```
gt = '../../../data/test1/indexLabel/1624325291-972695058.png'
array = np.array(Image.open(gt))
print(np.unique(array))
```

```
[ 2  5  6  7  8  9 11 14 17 18]
```

[42]:

```
mask_true = [ 2, 5, 6, 7, 8, 9, 11, 14, 17, 18]
mapping = [0, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 0, 11, 12, 13, 14, 15] #
transformed_mask = [0] * len(mask_true)
for i, v in enumerate(mask_true):
    transformed_mask[i] = mapping[v]
print(transformed_mask)
# pred = [ 1  6  7 14 15]
```

```
[1, 4, 5, 6, 7, 8, 10, 11, 14, 15]
```

0.0.2 Mask

[10]:

```
import numpy as np
from PIL import Image
def unique_values_in_mask(mask):
    return np.unique(mask)
def apply_mapping(mask, mapping): # Define function to apply mapping to mask
    new_mask = np.copy(mask)
    for old_value, new_value in enumerate(mapping):
        new_mask[mask == old_value] = new_value
    return new_mask

mask_path = '../../data/processed/val/indexLabel/1624327528-048416193.png'
origin_classes =[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,␣
 ↪18]
class_mapping = [0, 5, 1, 2, 3, 4, 5, 6, 7, 8,  9, 10, 13,  0, 11, 12, 13, 14,␣
 ↪15]
classes = {'asphalt': 1, 'dirt': 2, 'mud': 3, 'water': 4, 'gravel'
: 5, 'other-terrain': 6, 'tree-trunk': 7, 'tree-foliage': 8, 'bush': 9, 'fence':
 ↪ 10, 'structure': 11, 'pole': 12, 'vehicle': 13, 'rock': 14, 'log': 15,␣
 ↪'other-object': 16, 'sky': 17, 'grass': 18}
inverse_dict = {1: 'asphalt', 2: 'dirt', 3: 'mud', 4: 'water', 5: 'gravel', 6:␣
 ↪'other-terrain', 7: 'tree-trunk', 8: 'tree-foliage', 9: 'bush', 10: 'fence',␣
 ↪11: 'structure', 12: 'pole', 13: 'vehicle', 14: 'rock', 15: 'log', 16:␣
 ↪'other-object', 17: 'sky', 18: 'grass'}
mask = np.array(Image.open(mask_path))
unique_values = unique_values_in_mask(mask)
```

```
classes_in_mask = [inverse_dict[i] for i in unique_values]
print('  mask  ', unique_values, classes_in_mask)
mapped_mask = apply_mapping(mask, class_mapping)
print('  mask  ', unique_values_in_mask(mapped_mask))
print(inverse_dict)
```

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

[21]:
```
final_dict = {k: class_mapping[v] for k, v in classes.items()}
# print(final_dict)
final_dict['bg'] = 0
from collections import defaultdict
inverse_final_dict = defaultdict(list)
for k in final_dict:
    inverse_final_dict[final_dict[k]].append(k)
print(inverse_final_dict)
# list1 = []
for k in sorted(inverse_final_dict.keys()):
    print(k, inverse_final_dict[k])
    # list1.append(inverse_final_dict[k])
```

```
defaultdict(<class 'list'>, {5: ['asphalt', 'other-terrain'], 1: ['dirt'], 2:
['mud'], 3: ['water'], 4: ['gravel'], 6: ['tree-trunk'], 7: ['tree-foliage'], 8:
['bush'], 9: ['fence'], 10: ['structure'], 13: ['pole', 'other-object'], 0:
['vehicle', 'bg'], 11: ['rock'], 12: ['log'], 14: ['sky'], 15: ['grass']})
0 ['vehicle', 'bg']
1 ['dirt']
2 ['mud']
3 ['water']
4 ['gravel']
5 ['asphalt', 'other-terrain']
6 ['tree-trunk']
7 ['tree-foliage']
8 ['bush']
9 ['fence']
10 ['structure']
11 ['rock']
12 ['log']
13 ['pole', 'other-object']
14 ['sky']
15 ['grass']
```

```
0 ['vehicle', 'bg']
```

```
1 ['dirt']
2 ['mud']
3 ['water']
4 ['gravel']
5 ['asphalt', 'other-terrain']
6 ['tree-trunk']
7 ['tree-foliage']
8 ['bush']
9 ['fence']
10 ['structure']
11 ['rock']
12 ['log']
13 ['pole', 'other-object']
14 ['sky']
15 ['grass']
```

[25]:

```
nan = np.nan
i_iou = [       nan, 4.0439e-01, 0.0000e+00,        nan, 0.0000e+00, 0.0000e+00,
        3.7265e-01, 6.3374e-01, 1.8124e-06,        nan, 1.7964e-04, 0.0000e+00,
        0.0000e+00, 1.1568e-04, 6.9367e-01, 2.8160e-01]
classes = [['vehicle', 'background'], ['dirt'], ['mud'], ['water'], ['gravel'],␣
 ↪['asphalt', 'other-terrain'], ['tree-trunk'], ['tree-foliage'], ['bush'],␣
 ↪['fence'], ['structure'], ['rock'], ['log'], ['pole', 'other-object'],␣
 ↪['sky'], ['grass']]
for i in range(len(i_iou)):
    print(classes[i], i_iou[i])

# 0: vehicle, 2.3 * 10^5
# 1: dirt, 2.5 * 10^9
# 2: mud, 1.5 * 10^7
# 3: water, 5.8 * 10^7
# 4: gravel, 1.1 * 10^8
# 5: asphalt/other-terrain, 2.4*10^5+4.4*10^6 = 4.64*10^6
# 6: tree-trunk, 3.9 * 10^9
# 7: tree-foliage, 1.6 * 10^10
# 8: bush, 3.4 * 10^8
# 9: fence, 1 * 10^7
# 10: structure, 9.8 * 10^7
# 11: rock, 1.6*10^7
# 12: log, 1.2* 10^8
# 13: pole/other-object, 3.5*10^6 + 2.8 * 10^7 = 3.15 * 10^7
# 14: sky, 2.0 * 10^9
# 15: grass, 3 * 10^9
```

```
['vehicle', 'background'] nan
['dirt'] 0.40439
['mud'] 0.0
['water'] nan
```

```
['gravel'] 0.0
['asphalt', 'other-terrain'] 0.0
['tree-trunk'] 0.37265
['tree-foliage'] 0.63374
['bush'] 1.8124e-06
['fence'] nan
['structure'] 0.00017964
['rock'] 0.0
['log'] 0.0
['pole', 'other-object'] 0.00011568
['sky'] 0.69367
['grass'] 0.2816
```

[ ]:

```
# 1624327528-048416193.png
# classes value in mask_pred [ 1  4  6  7 10 14 15] # 4: gravel, 10: structure
# classes value in gt.       [ 1  6  7 12 14 15] # 12: log


# 1624327505-807209646.png
# classes value in mask_pred [ 1  6  7 10 13 14 15] 13: other-object
# classes value in gt.       [ 1  6  7  8 10 14 15] 8: bush


# classes value in mask_pred [ 1  6  7 10 14 15] 10: structure
# classes value in gt.       [ 1  6  7  8 14 15] 8: bush
```

[32]:

```
import torch
#
pixel_counts = [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:]
#
weights = 1 / torch.tensor(pixel_counts)
weights = weights / weights.sum()  #      1
weights = torch.cat((torch.tensor([0.0]), weights))

# print(weights)
```

[4]:

```
import numpy as np
import torch
x = torch.tensor([[1, 2, 3, 4, 5],
              [6, 7, 8, 9, 10],
              [11, 12, 13, 14, 15],
              [16, 17, 18, 19, 20],
              [21, 22, 23, 24, 25]])
print(x.numel())
```

```
25
```