



Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

[Read the guide](#)

 [milesial / Pytorch-UNet](#)

[Code](#) [Issues 35](#) [Pull requests 1](#) [Discussions](#) [Actions](#) [Security](#) [Insights](#)

 master ▾ [Pytorch-UNet / train.py](#) / [Code](#) Jump to ▾

[Go to file](#)



 milesial Pretrained model in the README

Latest commit 67bf11b on Aug 12, 2020 [History](#)

 3 contributors 

187 lines (157 sloc) | 7.36 KB

[Raw](#) [Blame](#)   

```
1 import argparse
2 import logging
3 import os
4 import sys
5
6 import numpy as np
7 import torch
8 import torch.nn as nn
9 from torch import optim
10 from tqdm import tqdm
11
12 from eval import eval_net
13 from unet import UNet
14
15 from torch.utils.tensorboard import SummaryWriter
16 from utils.dataset import BasicDataset
17 from torch.utils.data import DataLoader, random_split
18
19 dir_img = 'data/imgs/'
20 dir_mask = 'data/masks/'
21 dir_checkpoint = 'checkpoints/'
22
23
24 def train_net(net,
25              device,
26              epochs=5,
27              batch_size=1,
28              lr=0.001,
29              val_percent=0.1,
30              save_cp=True,
31              img_scale=0.5):
32
33     dataset = BasicDataset(dir_img, dir_mask, img_scale)
34     n_val = int(len(dataset) * val_percent)
35     n_train = len(dataset) - n_val
36     train, val = random_split(dataset, [n_train, n_val])
37     train_loader = DataLoader(train, batch_size=batch_size, shuffle=True, num_workers=8, pin_memory=True)
38     val_loader = DataLoader(val, batch_size=batch_size, shuffle=False, num_workers=8, pin_memory=True, drop_last=True)
39
40     writer = SummaryWriter(comment=f'LR_{lr}_BS_{batch_size}_SCALE_{img_scale}')
41     global_step = 0
42
43     logging.info(f'''Starting training:
44         Epochs:        {epochs}
45         Batch size:    {batch_size}
46         Learning rate:  {lr}
47         Training size:  {n_train}
48         Validation size: {n_val}
```

```

49     Checkpoints:      {save_cp}
50     Device:           {device.type}
51     Images scaling:   {img_scale}
52     '''
53
54     optimizer = optim.RMSprop(net.parameters(), lr=lr, weight_decay=1e-8, momentum=0.9)
55     scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min' if net.n_classes > 1 else 'max', patience=2)
56     if net.n_classes > 1:
57         criterion = nn.CrossEntropyLoss()
58     else:
59         criterion = nn.BCEWithLogitsLoss()
60
61     for epoch in range(epochs):
62         net.train()
63
64         epoch_loss = 0
65         with tqdm(total=n_train, desc=f'Epoch {epoch + 1}/{epochs}', unit='img') as pbar:
66             for batch in train_loader:
67                 imgs = batch['image']
68                 true_masks = batch['mask']
69                 assert imgs.shape[1] == net.n_channels, \
70                     f'Network has been defined with {net.n_channels} input channels, ' \
71                     f'but loaded images have {imgs.shape[1]} channels. Please check that ' \
72                     f'the images are loaded correctly.'
73
74                 imgs = imgs.to(device=device, dtype=torch.float32)
75                 mask_type = torch.float32 if net.n_classes == 1 else torch.long
76                 true_masks = true_masks.to(device=device, dtype=mask_type)
77
78                 masks_pred = net(imgs)
79                 loss = criterion(masks_pred, true_masks)
80                 epoch_loss += loss.item()
81                 writer.add_scalar('Loss/train', loss.item(), global_step)
82
83                 pbar.set_postfix(**{'loss (batch)': loss.item()})
84
85                 optimizer.zero_grad()
86                 loss.backward()
87                 nn.utils.clip_grad_value_(net.parameters(), 0.1)
88                 optimizer.step()
89
90                 pbar.update(imgs.shape[0])
91                 global_step += 1
92                 if global_step % (n_train // (10 * batch_size)) == 0:
93                     for tag, value in net.named_parameters():
94                         tag = tag.replace('.', '/')
95                         writer.add_histogram('weights/' + tag, value.data.cpu().numpy(), global_step)
96                         writer.add_histogram('grads/' + tag, value.grad.data.cpu().numpy(), global_step)
97                     val_score = eval_net(net, val_loader, device)
98                     scheduler.step(val_score)
99                     writer.add_scalar('learning_rate', optimizer.param_groups[0]['lr'], global_step)
100
101                 if net.n_classes > 1:
102                     logging.info('Validation cross entropy: {}'.format(val_score))
103                     writer.add_scalar('Loss/test', val_score, global_step)
104                 else:
105                     logging.info('Validation Dice Coeff: {}'.format(val_score))
106                     writer.add_scalar('Dice/test', val_score, global_step)
107
108                 writer.add_images('images', imgs, global_step)
109                 if net.n_classes == 1:
110                     writer.add_images('masks/true', true_masks, global_step)
111                     writer.add_images('masks/pred', torch.sigmoid(masks_pred) > 0.5, global_step)
112
113         if save_cp:
114             try:
115                 os.mkdir(dir_checkpoint)
116                 logging.info('Created checkpoint directory')
117             except OSError:
118                 pass
119             torch.save(net.state_dict(),
120                         dir_checkpoint + f'CP_epoch{epoch + 1}.pth')
121             logging.info(f'Checkpoint {epoch + 1} saved !')
122
123     writer.close()
124
125
126 def get_args():
127     parser = argparse.ArgumentParser(description='Train the UNet on images and target masks',
128                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
129     parser.add_argument('-e', '--epochs', metavar='E', type=int, default=5,

```

```

130         help='Number of epochs', dest='epochs')
131     parser.add_argument('-b', '--batch-size', metavar='B', type=int, nargs='?', default=1,
132         help='Batch size', dest='batchsize')
133     parser.add_argument('-l', '--learning-rate', metavar='LR', type=float, nargs='?', default=0.0001,
134         help='Learning rate', dest='lr')
135     parser.add_argument('-f', '--load', dest='load', type=str, default=False,
136         help='Load model from a .pth file')
137     parser.add_argument('-s', '--scale', dest='scale', type=float, default=0.5,
138         help='Downscaling factor of the images')
139     parser.add_argument('-v', '--validation', dest='val', type=float, default=10.0,
140         help='Percent of the data that is used as validation (0-100)')
141
142     return parser.parse_args()
143
144
145 if __name__ == '__main__':
146     logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
147     args = get_args()
148     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
149     logging.info(f'Using device {device}')
150
151     # Change here to adapt to your data
152     # n_channels=3 for RGB images
153     # n_classes is the number of probabilities you want to get per pixel
154     # - For 1 class and background, use n_classes=1
155     # - For 2 classes, use n_classes=1
156     # - For N > 2 classes, use n_classes=N
157     net = UNet(n_channels=3, n_classes=1, bilinear=True)
158     logging.info(f'Network:\n'
159         f'\t\t{net.n_channels} input channels\n'
160         f'\t\t{net.n_classes} output channels (classes)\n'
161         f'\t\t"Bilinear" if net.bilinear else "Transposed conv" upscaling')
162
163     if args.load:
164         net.load_state_dict(
165             torch.load(args.load, map_location=device)
166         )
167         logging.info(f'Model loaded from {args.load}')
168
169     net.to(device=device)
170     # faster convolutions, but more memory
171     # cudnn.benchmark = True
172
173     try:
174         train_net(net=net,
175             epochs=args.epochs,
176             batch_size=args.batchsize,
177             lr=args.lr,
178             device=device,
179             img_scale=args.scale,
180             val_percent=args.val / 100)
181     except KeyboardInterrupt:
182         torch.save(net.state_dict(), 'INTERRUPTED.pth')
183         logging.info('Saved interrupt')
184     try:
185         sys.exit(0)
186     except SystemExit:
187         os._exit(0)

```