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**

PyTorch implementation of the U-Net for image semantic segmentation with high quality images

GPL-3.0 License

3.6k stars 1.2k forks

Star Watch

Code Issues 35 Pull requests 1 Discussions Actions Security

master

Pytorch-UNet / **predict.py** / Jump to

milesial Removed dense_crf and small fixes History

4 contributors

Raw Blame

Executable File | 136 lines (102 sloc) | 4.14 KB

```
1  import argparse
2  import logging
3  import os
4
5  import numpy as np
```

```python
import torch
import torch.nn.functional as F
from PIL import Image
from torchvision import transforms

from unet import UNet
from utils.data_vis import plot_img_and_mask
from utils.dataset import BasicDataset


def predict_img(net,
                full_img,
                device,
                scale_factor=1,
                out_threshold=0.5):
    net.eval()

    img = torch.from_numpy(BasicDataset.preprocess(full_img, scale_factor))

    img = img.unsqueeze(0)
    img = img.to(device=device, dtype=torch.float32)

    with torch.no_grad():
        output = net(img)

        if net.n_classes > 1:
            probs = F.softmax(output, dim=1)
        else:
            probs = torch.sigmoid(output)

        probs = probs.squeeze(0)

        tf = transforms.Compose(
            [
                transforms.ToPILImage(),
                transforms.Resize(full_img.size[1]),
                transforms.ToTensor()
            ]
        )

        probs = tf(probs.cpu())
        full_mask = probs.squeeze().cpu().numpy()

    return full_mask > out_threshold


def get_args():
    parser = argparse.ArgumentParser(description='Predict masks from input images',
```

```
                                    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--model', '-m', default='MODEL.pth',
                        metavar='FILE',
                        help="Specify the file in which the model is stored")
    parser.add_argument('--input', '-i', metavar='INPUT', nargs='+',
                        help='filenames of input images', required=True)

    parser.add_argument('--output', '-o', metavar='INPUT', nargs='+',
                        help='Filenames of ouput images')
    parser.add_argument('--viz', '-v', action='store_true',
                        help="Visualize the images as they are processed",
                        default=False)
    parser.add_argument('--no-save', '-n', action='store_true',
                        help="Do not save the output masks",
                        default=False)
    parser.add_argument('--mask-threshold', '-t', type=float,
                        help="Minimum probability value to consider a mask pixel white",
                        default=0.5)
    parser.add_argument('--scale', '-s', type=float,
                        help="Scale factor for the input images",
                        default=0.5)

    return parser.parse_args()


def get_output_filenames(args):
    in_files = args.input
    out_files = []

    if not args.output:
        for f in in_files:
            pathsplit = os.path.splitext(f)
            out_files.append("{}_OUT{}".format(pathsplit[0], pathsplit[1]))
    elif len(in_files) != len(args.output):
        logging.error("Input files and output files are not of the same length")
        raise SystemExit()
    else:
        out_files = args.output

    return out_files


def mask_to_image(mask):
    return Image.fromarray((mask * 255).astype(np.uint8))


if __name__ == "__main__":
    args = get_args()
```

```
    in_files = args.input
    out_files = get_output_filenames(args)

    net = UNet(n_channels=3, n_classes=1)

    logging.info("Loading model {}".format(args.model))

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    logging.info(f'Using device {device}')
    net.to(device=device)
    net.load_state_dict(torch.load(args.model, map_location=device))

    logging.info("Model loaded !")

    for i, fn in enumerate(in_files):
        logging.info("\nPredicting image {} ...".format(fn))

        img = Image.open(fn)

        mask = predict_img(net=net,
                           full_img=img,
                           scale_factor=args.scale,
                           out_threshold=args.mask_threshold,
                           device=device)

        if not args.no_save:
            out_fn = out_files[i]
            result = mask_to_image(mask)
            result.save(out_files[i])

            logging.info("Mask saved to {}".format(out_files[i]))

        if args.viz:
            logging.info("Visualizing results for image {}, close to continue ...".format(fn))
            plot_img_and_mask(img, mask)
```