



Original

<https://raw.githubusercontent.com/usuyama/pytorch-unet/master/pytorch_unet.py>

import torch

import torch.nn as nn

def double\_conv(in\_channels, out\_channels):

    return nn.Sequential(

        nn.Conv2d(in\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True),

        nn.Conv2d(out\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True)

    )

class UNet(nn.Module):

    def \_\_init\_\_(self, n\_class):

        super().\_\_init\_\_()

        self.dconv\_down1 = double\_conv(3, 64)

        self.dconv\_down2 = double\_conv(64, 128)

        self.dconv\_down3 = double\_conv(128, 256)

        self.dconv\_down4 = double\_conv(256, 512)

        self.maxpool = nn.MaxPool2d(2)

        self.upsample = nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True)

        self.dconv\_up3 = double\_conv(256 + 512, 256)

        self.dconv\_up2 = double\_conv(128 + 256, 128)

        self.dconv\_up1 = double\_conv(128 + 64, 64)

        self.conv\_last = nn.Conv2d(64, n\_class, 1)

    def forward(self, x):

        conv1 = self.dconv\_down1(x)

        x = self.maxpool(conv1)

        conv2 = self.dconv\_down2(x)

        x = self.maxpool(conv2)

        conv3 = self.dconv\_down3(x)

        x = self.maxpool(conv3)

        x = self.dconv\_down4(x)

        x = self.upsample(x)

        x = torch.cat([x, conv3], dim=1)

        x = self.dconv\_up3(x)

        x = self.upsample(x)

        x = torch.cat([x, conv2], dim=1)

        x = self.dconv\_up2(x)

        x = self.upsample(x)

        x = torch.cat([x, conv1], dim=1)

        x = self.dconv\_up1(x)

        out = self.conv\_last(x)

        return out

MOdified

import torch

import torch.nn as nn

def double\_conv(in\_channels, out\_channels):

    return nn.Sequential(

        nn.Conv2d(in\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True),

        nn.Conv2d(out\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True)

    )

class UNet(nn.Module):

    def \_\_init\_\_(self, n\_class):

        super().\_\_init\_\_()

        self.dconv\_down1 = double\_conv(3, 64) # Changed from 3 to 6

        self.dconv\_down2 = double\_conv(64, 128)

        self.dconv\_down3 = double\_conv(128, 256)

        self.dconv\_down4 = double\_conv(256, 512)

        self.maxpool = nn.MaxPool2d(2)

        self.upsample = nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True)

        self.dconv\_up3 = double\_conv(256 + 512, 256)

        self.dconv\_up2 = double\_conv(128 + 256, 128)

        self.dconv\_up1 = double\_conv(128 + 64, 64)

        self.conv\_last = nn.Conv2d(64, n\_class, 1)

    def forward(self, sample):

        # bgs = sample['bg']

        bgfgs = sample['bg\_fg']

        # c = torch.cat([bgs, bgfgs], dim=1)

        '''print('input: {}'.format(input.shape))

        print('bg: {}'.format(bgs.shape))

        print('bg and fg: {}'.format(bgfgs.shape))

        print('concatenated: {}'.format(c.shape))'''

        conv1 = self.dconv\_down1(bgfgs)

        x = self.maxpool(conv1)

        conv2 = self.dconv\_down2(x)

        x = self.maxpool(conv2)

        conv3 = self.dconv\_down3(x)

        x = self.maxpool(conv3)

        x = self.dconv\_down4(x)

        x = self.upsample(x)

        x = torch.cat([x, conv3], dim=1)

        x = self.dconv\_up3(x)

        x = self.upsample(x)

        x = torch.cat([x, conv2], dim=1)

        x = self.dconv\_up2(x)

        x = self.upsample(x)

        x = torch.cat([x, conv1], dim=1)

        x = self.dconv\_up1(x)

        out = self.conv\_last(x)

        # print('output : {}'.format(out.shape))

        return out

Trial 2:

import torch

import torch.nn as nn

def double\_conv(in\_channels, out\_channels):

    return nn.Sequential(

        nn.Conv2d(in\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True),

        nn.Conv2d(out\_channels, out\_channels, 3, padding=1),

        nn.ReLU(inplace=True)

    )

class UNet(nn.Module):

    def \_\_init\_\_(self, n\_class):

        super().\_\_init\_\_()

        self.dconv\_down1 = double\_conv(3, 64) # Changed from 3 to 6

        self.dconv\_down2 = double\_conv(64, 128)

        self.dconv\_down3 = double\_conv(128, 256)

        self.dconv\_down4 = double\_conv(256, 512)

        self.maxpool = nn.MaxPool2d(2)

        self.upsample = nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True)

        self.dconv\_up3 = double\_conv(256 + 512, 256)

        self.dconv\_up2 = double\_conv(128 + 256, 128)

        self.dconv\_up1 = double\_conv(128 + 64, 64)

        self.conv\_last = nn.Conv2d(64, n\_class, 1)

    def forward(self, sample):

        # bgs = sample['bg']

        bgfgs = sample['bg\_fg']

        # c = torch.cat([bgs, bgfgs], dim=1)

        '''print('input: {}'.format(input.shape))

        print('bg: {}'.format(bgs.shape))

        print('bg and fg: {}'.format(bgfgs.shape))

        print('concatenated: {}'.format(c.shape))'''

        conv1 = self.dconv\_down1(bgfgs)

        x = self.maxpool(conv1)

        conv2 = self.dconv\_down2(x)

        x = self.maxpool(conv2)

        conv3 = self.dconv\_down3(x)

        x = self.maxpool(conv3)

        x = self.dconv\_down4(x)

        # Split from here

        x = self.upsample(x)

        x = torch.cat([x, conv3], dim=1)

        x = self.dconv\_up3(x)

        x = self.upsample(x)

        x = torch.cat([x, conv2], dim=1)

        x = self.dconv\_up2(x)

        x = self.upsample(x)

        x = torch.cat([x, conv1], dim=1)

        x = self.dconv\_up1(x)

        mout = self.conv\_last(x)

        # model for Depth

        x = self.upsample(conv4)

        x = torch.cat([x, conv3], dim=1)

        x = self.dconv\_up3(x)

        x = self.upsample(x)

        x = torch.cat([x, conv2], dim=1)

        x = self.dconv\_up2(x)

        x = self.upsample(x)

        x = torch.cat([x, conv1], dim=1)

        x = self.dconv\_up1(x)

        dout = self.conv\_last(x)

        # print('output : {}'.format(out.shape))

        # return out

        return mout, dout