Programming Assignment 3

Jun Seo Ha
Artificial Intelligence Graduate School
Gwangju Institute of Science and Technology

이 글은 ***Image Harmonization***에 대해, network architecture generator G에서, RAIN adaptation을 통해 기존의 Instance normalization (without RAIN)보다 성능이 우수함을 보여주는 글입니다.

1. Overview

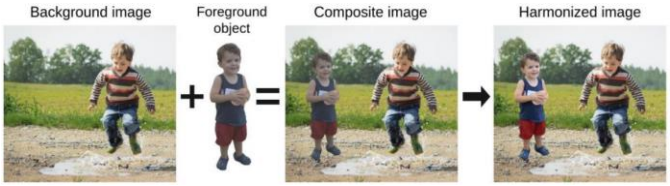

Figure1. Image Harmonization: Adjust the color consistency between the background image and the foreground object using foreground mask.

Background image와 foreground image를 합성할 때에 두 이미지 색상의 일관성을 맞춰주는 것이 중요합니다. 기존의 합성 이미지를 조화시키기 위한 deep learning method (AdaIN)은 background image와 foreground image간의 합성 이미지와 실제 이미지 사이에서 visual style consistency의 명시적인 탐색 없이 학습을 진행합니다. 이에 대해 RAIN은 Background image와 foreground image 사이의 visual style consistency를 보장하기 이미지 조화 문제를 style transfer problem으로 다룹니다.

2. Step1: Build a network architecture with IN

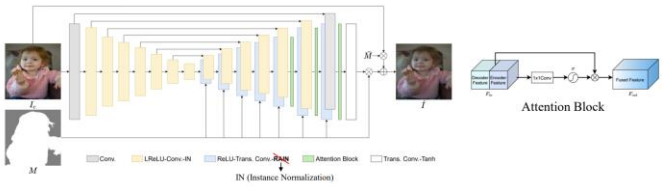

Figure 2. Overview of the proposed generator.

AdaIN은 실제 시간 image stylization을 제시했습니다. 이 모델은 pre-trained VGG network를 이용하여 style code를 추출했습니다. 이 모델의 문제점으로, foreground 영역이 제거된 배경 이미지는 pre-trained network로 추출할 수 없으므로 이에 대해, 평균 및 분산 이 동이라는 문제가 발생하게 됩니다.

3. Step2: Build a network architecture with RAIN

위에서 언급한 IN network architecture의 문제점을 해결하기 위해, Region-aware Adaptive Instance Normalization(RAIN)을 통해 background image와 foreground image를 연관시키는 방법을 제시합니다. 다시 말해서, 이미지 조화 문제를 background image에서 foreground image로 new style transfer하는 task로 여깁니다.

4. Result

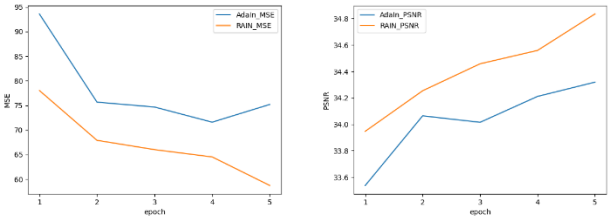

Figure3. Mean Square Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) comparison of AdaIN and RAIN according to epoch

Fig.3.을 보면, 1번째 학습에 대해서부터 RAIN은 AdaIN보다 MSE와 PSNR에서 더 좋은 성능을 보임을 알 수 있다. 다른 전체 외부 이미지 feature의 style을 고려하는 AdaIN과 다르게, 동일한 image에서만 background에서 foreground로 style transfer하는 RAIN이 효과적이라는 것을 알 수 있습니다. 그래서 초기 학습 상태에서도 RAIN은 AdaIN보다 훨씬 좋은 성능을 보이며, 이후 학습이 진행됨에 따라서도, AdaIN과 달리 일관되게 성능이 향상되는 것을 보입니다.

References

[1] Ling, Jun, et al. "Region-aware adaptive instance normalization for image harmonization." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[2] Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." *Proceedings of the IEEE international conference on computer vision*. 2017.

Code

networks.py

Class RainNet(nn.Module)

```python
class RainNet(nn.Module):
    def __init__(self, input_nc=3, output_nc=3, ngf=64, norm_layer=RAIN,                ### input_nc=3, output_nc=3
                 norm_type_indicator=[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],    # RAIN
##                 norm_type_indicator=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],   # IN

                 use_dropout=False, use_attention=True):
        """

        Parameters:
            input_nc (int) -- the number of channels in input images
            output_nc (int) -- the number of channels in output images
            ngf (int) -- the number of filters in the last conv layer
            netG (str) -- the architecture's name: rainnet
            norm (str) -- the name of normalization layers used in the network: batch | instance | none
            use_dropout (bool) -- if use dropout layers.
            init_type (str)    -- the name of our initialization method.
            init_gain (float)  -- scaling factor for normal, xavier and orthogonal.
            gpu_ids (int list) -- which GPUs the network runs on: e.g., 0,1,2

        """
        super(RainNet, self).__init__()
        self.input_nc = input_nc
        self.norm_namebuffer = ['RAIN']
        self.use_dropout = use_dropout
        self.use_attention = use_attention
        norm_type_list = [get_norm_layer('instance'), get_norm_layer('rain')]  # elif norm_type.startswith('rain'):   # from models.normalize import RAIN
        # --------------------------Network Settings--------------------------
        # fill the blank
        ##############################################################################
        # kernel size = 4 for all
        self.convlayer0 = nn.Conv2d(input_nc, ngf, kernel_size = 4, stride = 2, padding = 1)


        self.actlayer1 = nn.LeakyReLU(0.2)
        self.convlayer1 = nn.Conv2d(ngf, ngf*2, kernel_size = 4, stride = 2, padding = 1)
        self.normlayer1 = norm_type_list[norm_type_indicator[0]](ngf*2)  # RAIN adaptation?      # norm_type_indicator=[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]



        self.actlayer2 = nn.LeakyReLU(0.2)
        self.convlayer2 = nn.Conv2d(ngf*2, ngf*4, kernel_size = 4, stride = 2, padding = 1)
        self.normlayer2     = norm_type_list[norm_type_indicator[1]](ngf*4)



        self.actlayer3 = nn.LeakyReLU(0.2)
        self.convlayer3 = nn.Conv2d(ngf*4, ngf*8, kernel_size = 4, stride = 2, padding = 1)
        self.normlayer3     = norm_type_list[norm_type_indicator[2]](ngf*8)
```

```python
        unet_block = UnetBlockCodec(ngf * 8, ngf * 8, input_nc=None, submodule=None, norm_layer=norm_layer,  # norm_layer=RAIN
                                    innermost=True, enc=norm_type_indicator[6], dec=norm_type_indicator[7])  # add the innermost layer

        unet_block = UnetBlockCodec(ngf * 8, ngf * 8, input_nc=None, submodule=unet_block, norm_layer=norm_layer,
                                    use_dropout=use_dropout, enc=norm_type_indicator[5], dec=norm_type_indicator[8])

        unet_block = UnetBlockCodec(ngf * 8, ngf * 8, input_nc=None, submodule=unet_block, norm_layer=norm_layer,
                                    use_dropout=use_dropout, enc=norm_type_indicator[4], dec=norm_type_indicator[9])

        self.unet_block = UnetBlockCodec(ngf * 8, ngf * 8, input_nc=None, submodule=unet_block, norm_layer=norm_layer,
                                    use_dropout=use_dropout, enc=norm_type_indicator[3], dec=norm_type_indicator[10])

##########################################################################################


        self.actlayer11 = nn.ReLU()
        self.deconvlayer11 = nn.ConvTranspose2d(ngf*16, ngf*4, kernel_size=4, stride=2, padding=1)
        self.normlayer11 = norm_type_list[norm_type_indicator[11]](ngf*4)

        self.attconvlayer11 = nn.Conv2d(ngf*8, ngf*8, kernel_size=1, stride=1)
        self.attactlayer11 = nn.Sigmoid()



        self.actlayer12 = nn.ReLU()
        self.deconvlayer12 = nn.ConvTranspose2d(ngf*8, ngf*2, kernel_size=4, stride=2, padding=1)
        self.normlayer12 = norm_type_list[norm_type_indicator[12]](ngf*2)

        self.attconvlayer12 = nn.Conv2d(ngf*4, ngf*4, kernel_size=1, stride=1)
        self.attactlayer12 = nn.Sigmoid()



        self.actlayer13 = nn.ReLU()
        self.deconvlayer13 = nn.ConvTranspose2d(ngf*4, ngf, kernel_size=4, stride=2, padding=1)
        self.normlayer13 = norm_type_list[norm_type_indicator[13]](ngf)

        self.attconvlayer13 = nn.Conv2d(ngf*2, ngf*2, kernel_size=1, stride=1)
        self.attactlayer13 = nn.Sigmoid()



        self.actresult = nn.ReLU()
        self.deconvresult = nn.ConvTranspose2d(ngf * 2, output_nc, kernel_size=4, stride=2, padding=1)
        self.result = nn.Tanh()
```

```python
    def forward(self, x, mask):
        # fill the blank
        ##############################################################################

        l0 = self.convlayer0(x)

        l1 = self.actlayer1(l0)
        l1 = self.convlayer1(l1)

        l2 = self.actlayer2(l1)
        l2 = self.convlayer2(l2)

        l3 = self.actlayer3(l2)
        l3 = self.convlayer3(l3)


        u3 = self.unet_block(l3, mask)


        u2 = self.actlayer11(u3)
        u2 = self.deconvlayer11(u2)

        if self.normlayer11._get_name() in self.norm_namebuffer:  ## self.norm_namebuffer = RAIN     ### RAIN adaptation indicator에 따라 수행됨
            u2 = self.normlayer11(u2, mask)
        else:
            u2 = self.normlayer11(u2)
        u2 = torch.cat([l2, u2], 1)



        u2_1 = self.attconvlayer11(u2)
        u2_1 = self.attactlayer11(u2_1)
        u2 = u2_1 * u2


        u1 = self.actlayer12(u2)
        u1 = self.deconvlayer12(u1)


        if self.normlayer12._get_name() in self.norm_namebuffer:
            u1 = self.normlayer12(u1, mask)
        else:
            u1 = self.normlayer12(u1)
        u1 = torch.cat([l1, u1], 1)



        u1_1 = self.attconvlayer12(u1)
        u1_1 = self.attactlayer12(u1_1)
        u1 = u1_1 * u1
```

```python
        u0 = self.actlayer13(u1)
        u0 = self.deconvlayer13(u0)



        if self.normlayer13._get_name() in self.norm_namebuffer:
            u0 = self.normlayer13(u0, mask)
        else:
            u0 = self.normlayer13(u0)
        u0 = torch.cat([l0, u0], 1)


        u0_1 = self.attconvlayer13(u0)
        u0_1 = self.attactlayer13(u0_1)
        u0 = u0_1 * u0


        out = self.actresult(u0)
        out = self.deconvresult(out)
        out = self.result(out)

        ##############################################################################


        return out
```

normalize.py

Class RAIN(nn.Module)

```python
class RAIN(nn.Module):
    def __init__(self, dims_in, eps=1e-5):
        '''Compute the instance normalization within only the background region, in which
           the mean and standard variance are measured from the features in background region.
        '''
        super(RAIN, self).__init__()
        self.foreground_gamma = nn.Parameter(torch.zeros(dims_in), requires_grad=True)
        self.foreground_beta = nn.Parameter(torch.zeros(dims_in), requires_grad=True)
        self.background_gamma = nn.Parameter(torch.zeros(dims_in), requires_grad=True)
        self.background_beta = nn.Parameter(torch.zeros(dims_in), requires_grad=True)
        self.eps = eps

    def forward(self, x, mask):


        # fill the blank
        ##########################################################################



        ##########################################################################

        mask = F.interpolate(mask.detach(), size = (x.shape[2],x.shape[2]))


        mean_fore, std_fore = self.get_foreground_mean_std(x * mask, mask)
        mean_back, std_back = self.get_foreground_mean_std(x * (1-mask), 1-mask) # (region_back, mask_back)


        forgamma  = self.foreground_gamma[None, :, None, None]
        forbeta   = self.foreground_beta[None, :, None, None]
        normalized_foreground = ( (forgamma+1)  *  ( (x - mean_fore) / std_fore * std_back + mean_back ) + forbeta ) * mask


        backgamma = self.background_gamma[None, :, None, None]
        backbeta  = self.background_beta[None, :, None, None]
        normalized_background = ( (backgamma+1) *  ( (x - mean_back) / std_back) + backbeta ) * (1 - mask)


        ##########################################################################




        ##########################################################################


        return normalized_foreground + normalized_background
```

```python
    def mysum(self, tensor):
        list = []
        for i in range(tensor.shape[1]):
            channelsum = torch.sum(tensor[:,i])
            list.append(channelsum)

        result = torch.stack(list)
        result = result[None,:]
        list.clear()

        return result

    def get_foreground_mean_std(self, region, mask):

        # fill the blank
##        print("\n\n###################   get_foreground_mean_std   ################################\n\n")


        ##########################################################################
        sum = self.mysum(region)

        num = self.mysum(mask)

        mu = sum / (num + self.eps)

        mean = mu[:, :, None, None]

        upper = (region + (1 - mask)*mean - mean) ** 2

        var = self.mysum(upper) / (num + self.eps)

        var = var[:, :, None, None]

        ##########################################################################




        return mean, torch.sqrt(var+self.eps)
```