Weekly Work Report

Author Name

VISION@OUC

August 19, 2018

1 Research problem

I am working on the paper written by Chao Wang *et al.* [3] to add some codes to it for realizing more functions and making the generated images look more reality.

2 Research approach

I need to read papers and codes about GAN, DCGAN and dilation convolution generator *et al.*.

3 Research progress

Before this week, I have written codes about autoencoder, GAN and DCGAN. Then, I have read the pix2pix paper and codes cloned from github [2]. Cloning the DRPAN codes [1] to run it at the server to see the final results. The latest work is writting codes about the dilation convolution.

4 Progress in this week

The main work needed to do is writting a generator code using the dilation convolution.

4.1 The first dilation convolution network

The first attempt writting the dilation convolution codes is using a thin but deep network proposed at [4] just like Figure 1.

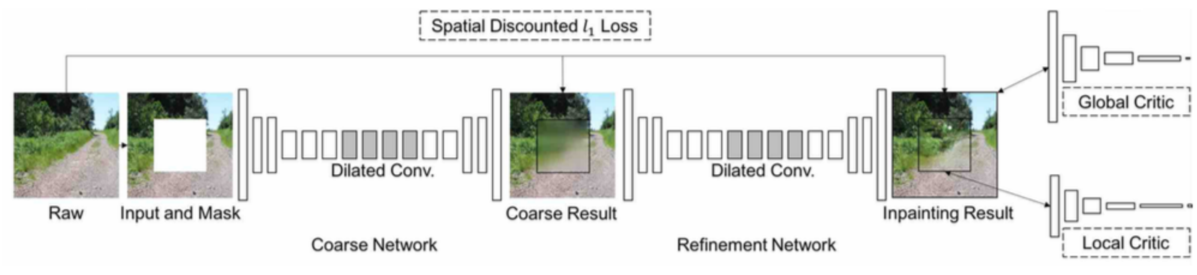


Figure 1: Overview of the improved generative inpainting framework. The coarse network is trained with reconstruction loss explicitly, while the refinement network is trained with reconstruction loss, global and local WGAN-GP adversarial loss.

The reason why we want to use the dilation convolution as the consists of the generator is that the dilation convolution is a powerful tool that can enlarge the receptive field of feature points without reducing the resolution of the feature maps.

When I writed the codes and run it at the server, however, the final result does not work perfectly. Generated images are all pink pictures like Figure 2.

After reading and analysing the paper, I finded a possible reason: the loss we used does not match the network. The loss used in the paper is WGAN-GP loss which benefits greatly to their inpainting framework as validated by its learning curves and faster/stabler convergence behaviors. But the same model trained with DCGAN sometimes collapses to limited modes for the inpainting task. Because the network (see Figure 1) is thin but deep, so it hard to train and easy to collapse.

In view of this situation, there are two methods: **(i)** the first is changing the training process. **(ii)** the second is changing a generator network. I chose the second one.

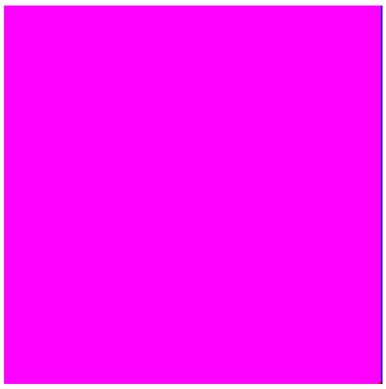


Figure 2: The generated fakeB image at the epoch 001.

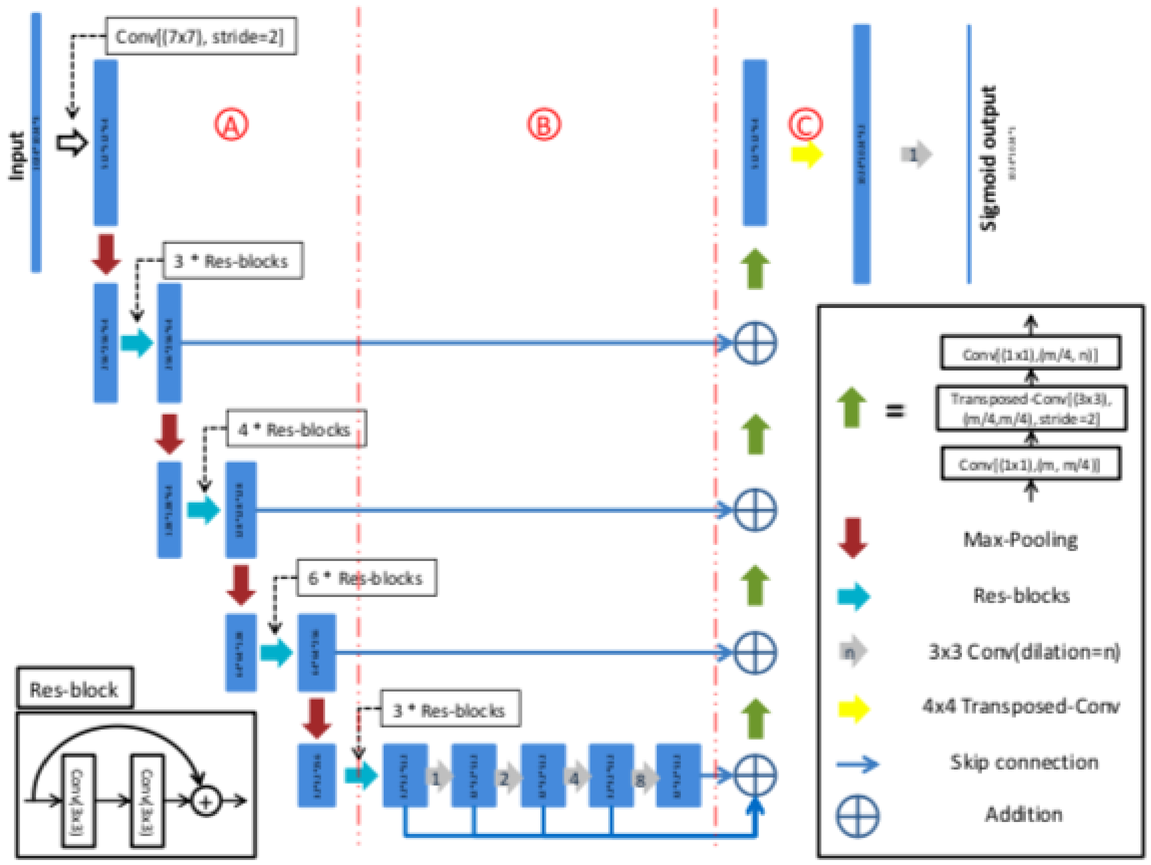
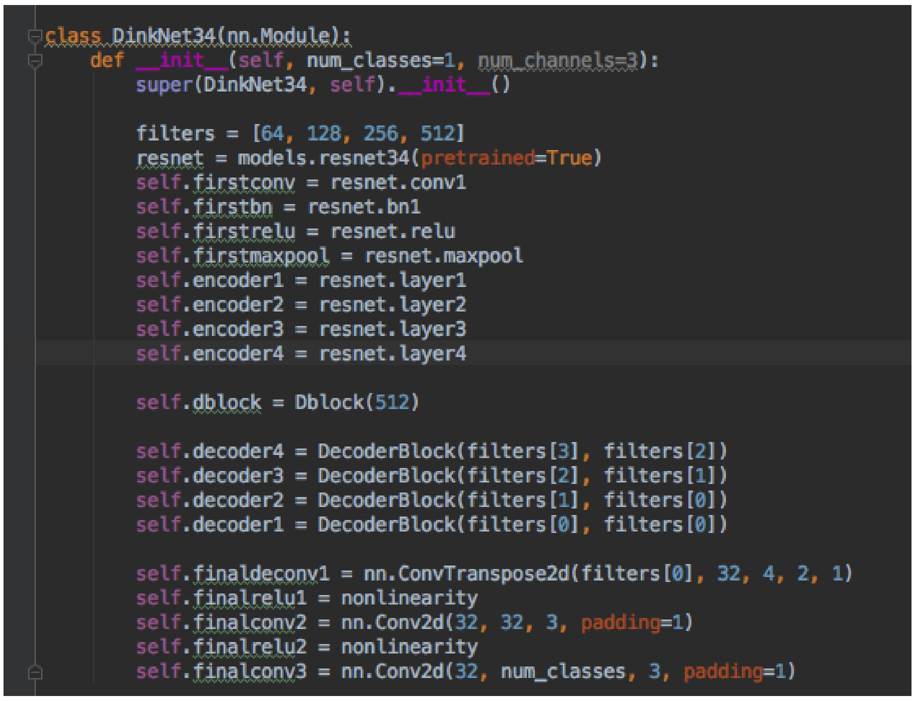


Figure 3: D-LinkNet architecture. Each blue rectangular block represents a multi-channel features map. Part A is the encoder of D-LinkNet. D-LinkNet uses ResNet34 as encoder. Part C is the decoder of D-LinkNet, it is set the same as LinkNet decoder. Original LinkNet only has Part A and Part C. D-LinkNet has an additional Part B which can enlarge the receptive field and as well as preserve the detailed spatial information. Each convolution layer is followed by a ReLU activation except the last convolution layer which use sigmoid activation.



```
class DinkNet34(nn.Module):
    def __init__(self, num_classes=1, num_channels=3):
        super(DinkNet34, self).__init__()

        filters = [64, 128, 256, 512]
        resnet = models.resnet34(pretrained=True)
        self.firstconv = resnet.conv1
        self.firstbn = resnet.bn1
        self.firstrelu = resnet.relu
        self.firstmaxpool = resnet.maxpool
        self.encoder1 = resnet.layer1
        self.encoder2 = resnet.layer2
        self.encoder3 = resnet.layer3
        self.encoder4 = resnet.layer4

        self.dblock = Dblock(512)

        self.decoder4 = DecoderBlock(filters[3], filters[2])
        self.decoder3 = DecoderBlock(filters[2], filters[1])
        self.decoder2 = DecoderBlock(filters[1], filters[0])
        self.decoder1 = DecoderBlock(filters[0], filters[0])

        self.finaldeconv1 = nn.ConvTranspose2d(filters[0], 32, 4, 2, 1)
        self.finalrelu1 = nonlinearity
        self.finalconv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.finalrelu2 = nonlinearity
        self.finalconv3 = nn.Conv2d(32, num_classes, 3, padding=1)
```

Figure 4: The first part of the ResNet34 code.

4.2 The second dilation convolution

The second network drawn lessons from [5] is using the dilation convolution with resnet shown at Figure 3.

Consideration of avoiding a thin but deep network, I chose a network with complex but shallow network. The whole network is divided by three parts. The A part is encoder (see Figure 4 and Figure 5). D-LinkNet uses ResNet34 pretrained on ImageNet dataset as its encoder. ResNet34 is originally designed for classification task on mid-resolution images of size 256×256.

Because we want to increase the receptive field of feature points in the center part of the network as well as keep the detailed information. And using pooling layers could multiply increase the receptive field of feature points, but may reduce the resolution of center features maps and drop spacial information. The dilated convolution layer can be desirabel alternative of pooling layer. Thus, D-LinkNet uses several dilated convolution layers with skip connections in the center part (see Figure 6).

The decoder (see Figure 7) of D-LinkNet remains the same as the original LinkNet which is computationally efficient. The decoder part uses transposed convolution layers to do umsampling, restoring the resolution of feature map from 32×32 to 1024×1024.

I have to add codes to our codes, but I don not run it successfully.

References

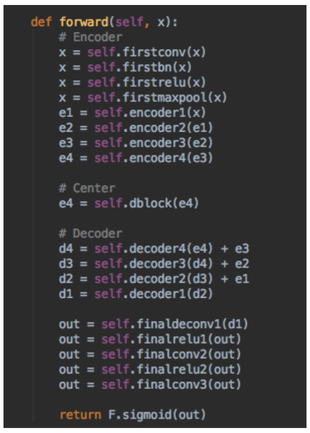
[1] GitHub_godisboy. https://github.com/godisboy/DRPAN. 1

[2] GitHub_junyanz. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix. 1

[3] C. Wang, H. Zheng, Z. Yu, Z. Zheng, Z. Gu, and B. Zheng. Discriminative region proposal adversarial networks for high-quality image-to-image translation. In *ECCV*, 2018. 1

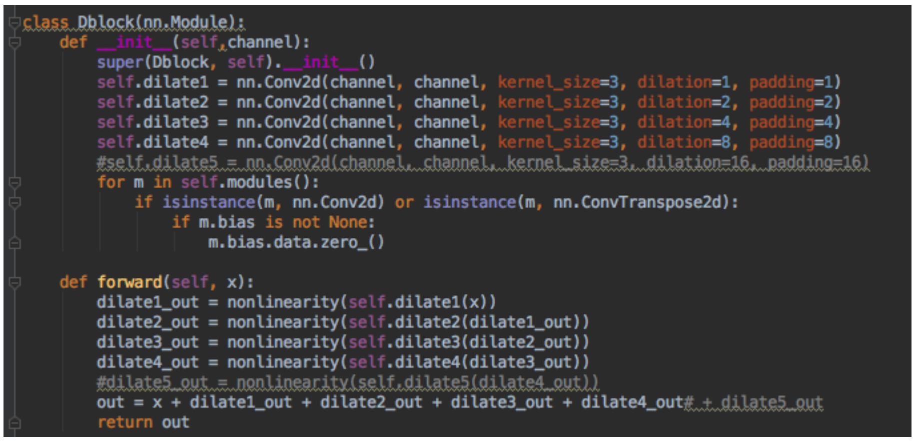
[4] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018. 1

[5] L. Zhou, C. Zhang, and M. Wu. D-LinkNet: LinkNet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction. In *CVPR*, 2018. 3



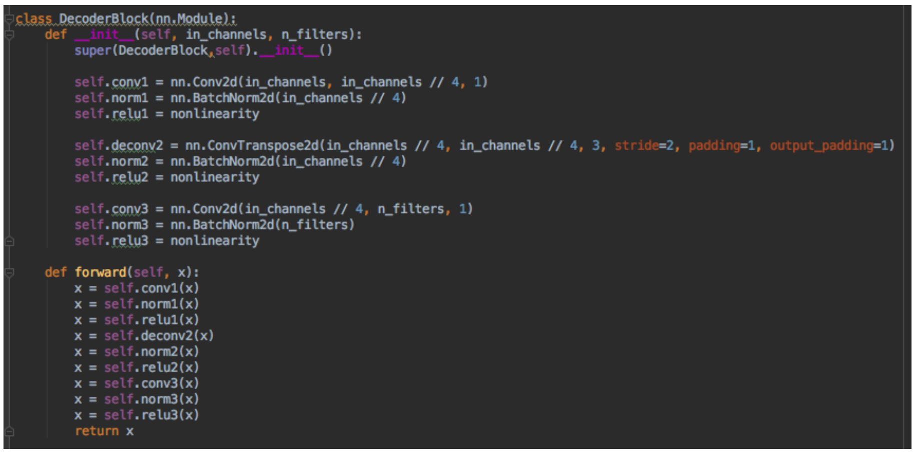
```python
    def forward(self, x):
        # Encoder
        x = self.firstconv(x)
        x = self.firstbn(x)
        x = self.firstrelu(x)
        x = self.firstmaxpool(x)
        e1 = self.encoder1(x)
        e2 = self.encoder2(e1)
        e3 = self.encoder3(e2)
        e4 = self.encoder4(e3)

        # Center
        e4 = self.dblock(e4)

        # Decoder
        d4 = self.decoder4(e4) + e3
        d3 = self.decoder3(d4) + e2
        d2 = self.decoder2(d3) + e1
        d1 = self.decoder1(d2)

        out = self.finaldeconv1(d1)
        out = self.finalrelu1(out)
        out = self.finalconv2(out)
        out = self.finalrelu2(out)
        out = self.finalconv3(out)

        return F.sigmoid(out)
```

Figure 5: The second part of the ResNet34 code.



```python
class Dblock(nn.Module):
    def __init__(self,channel):
        super(Dblock, self).__init__()
        self.dilate1 = nn.Conv2d(channel, channel, kernel_size=3, dilation=1, padding=1)
        self.dilate2 = nn.Conv2d(channel, channel, kernel_size=3, dilation=2, padding=2)
        self.dilate3 = nn.Conv2d(channel, channel, kernel_size=3, dilation=4, padding=4)
        self.dilate4 = nn.Conv2d(channel, channel, kernel_size=3, dilation=8, padding=8)
        #self.dilate5 = nn.Conv2d(channel, channel, kernel_size=3, dilation=16, padding=16)
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
                if m.bias is not None:
                    m.bias.data.zero_()

    def forward(self, x):
        dilate1_out = nonlinearity(self.dilate1(x))
        dilate2_out = nonlinearity(self.dilate2(dilate1_out))
        dilate3_out = nonlinearity(self.dilate3(dilate2_out))
        dilate4_out = nonlinearity(self.dilate4(dilate3_out))
        #dilate5_out = nonlinearity(self.dilate5(dilate4_out))
        out = x + dilate1_out + dilate2_out + dilate3_out + dilate4_out# + dilate5_out
        return out
```

Figure 6: The code for dilation convolution.



```python
class DecoderBlock(nn.Module):
    def __init__(self, in_channels, n_filters):
        super(DecoderBlock,self).__init__()

        self.conv1 = nn.Conv2d(in_channels, in_channels // 4, 1)
        self.norm1 = nn.BatchNorm2d(in_channels // 4)
        self.relu1 = nonlinearity

        self.deconv2 = nn.ConvTranspose2d(in_channels // 4, in_channels // 4, 3, stride=2, padding=1, output_padding=1)
        self.norm2 = nn.BatchNorm2d(in_channels // 4)
        self.relu2 = nonlinearity

        self.conv3 = nn.Conv2d(in_channels // 4, n_filters, 1)
        self.norm3 = nn.BatchNorm2d(n_filters)
        self.relu3 = nonlinearity

    def forward(self, x):
        x = self.conv1(x)
        x = self.norm1(x)
        x = self.relu1(x)
        x = self.deconv2(x)
        x = self.norm2(x)
        x = self.relu2(x)
        x = self.conv3(x)
        x = self.norm3(x)
        x = self.relu3(x)
        return x
```

Figure 7: The code for decoder part.