VDSR

Accurate Image Super-Resolution Using Very Deep Convolutional Networks / CVPR 2016

깊은 LAYER의 CNN 구조를 SR problem에 성공적으로 적용하였다는 의미가 있는 논문

SRCNN의 3가지 관점에서의 문제점과 해결책을 제시한다.

1) LAYER가 얕아 좁은 이미지 영역에 대한 정보만을 사용한다는 것

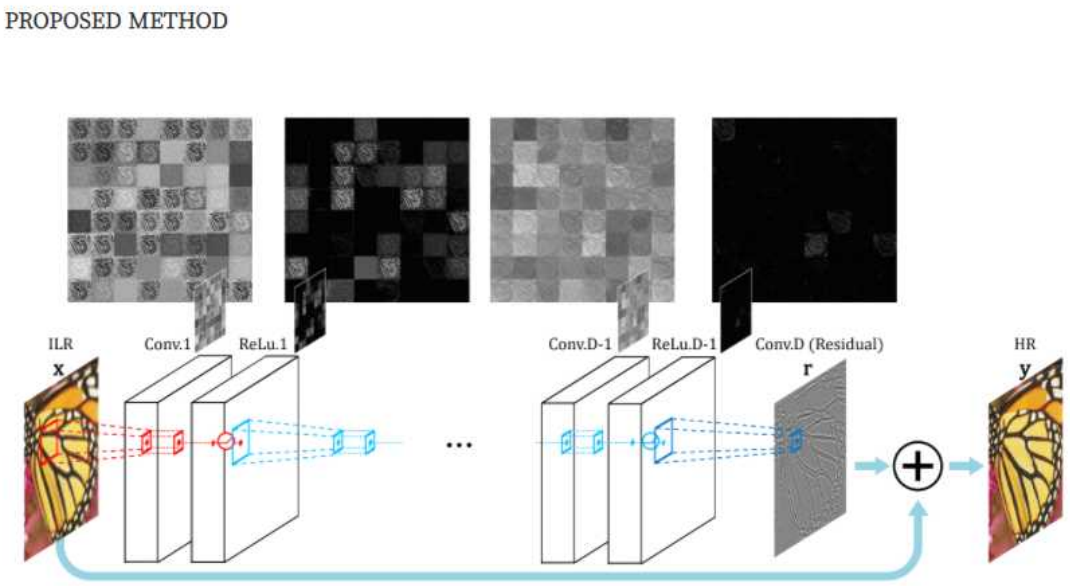
-> VDSR은 LAYER가 깊음으로 인해 넓은 Receptive filed 덕에 넓은 영역에 퍼져있는 Contextual information을 사용하여 높은 scale에서도 안정적인 성능을 보인다.

2) Convergence가 매우 느리다느 것

-> 두가지 방법을 제시했는데, 첫째는 Residual Learning 방식을 사용하였고, 두 번째는 매우 높은 Learning rate를 사용했다느 것이다. Residual learning 방식은 LR 이미지와 HR 이미지가 매우 높은 유사도를 보임으로 인해 높은 효율을 보여주며, 높은 Learning rate를 이용한 빠른 Convergence는 Residual learning과 Gradient clipping 덕에 가능하게 된다.

3) 단일 scale에 대해서만 가능하는 점

-> VDSR은 단일 모델이지만 다양한 Scale에 대응할 수 있다.

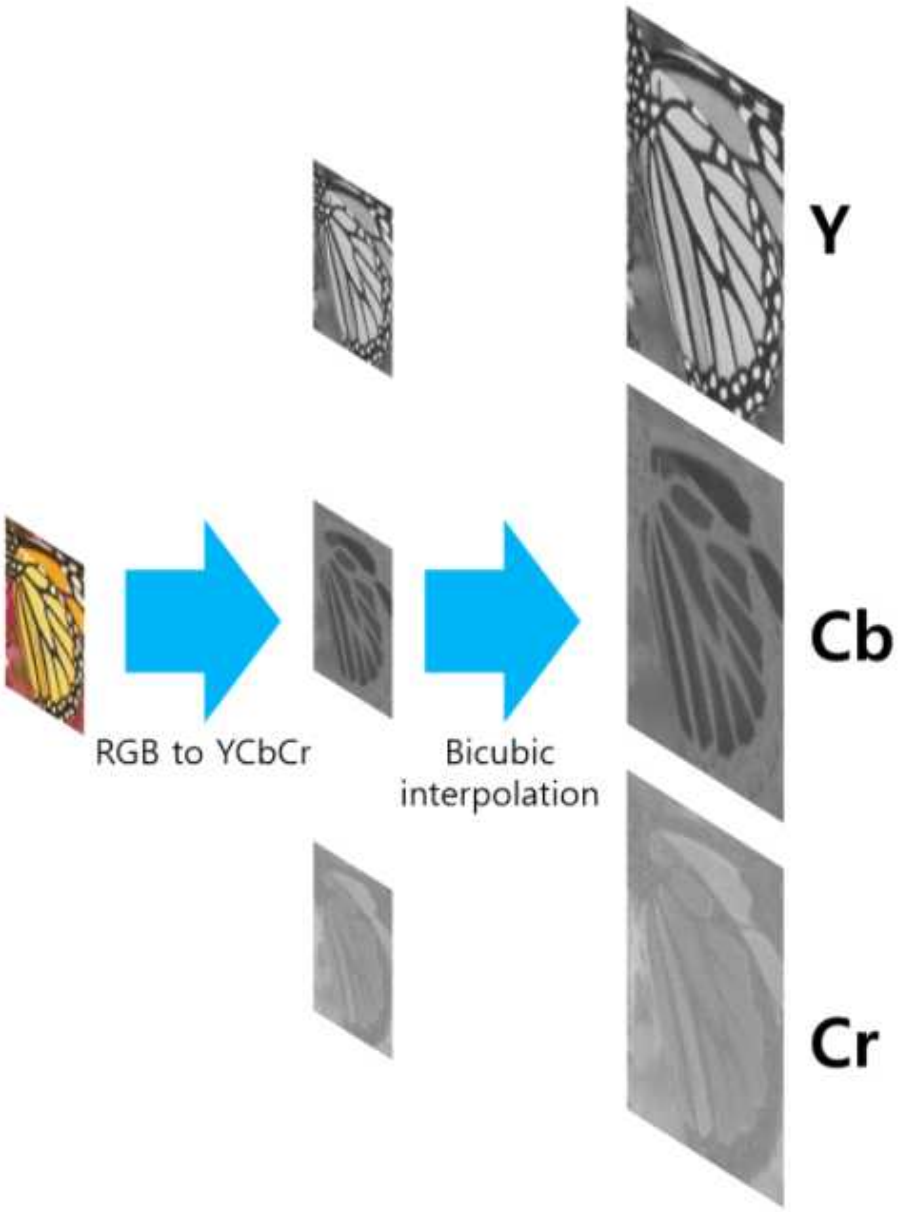


VDSR 구조는 20개의 LAYER로 구성되어 있다.

VDSR의 LR 이미지는 HR 이미지와 해상도가 같다. VDSR에서 표현하는 LR 이미지는 단지 Bicubic interpolation 된 이미지임을 생각하고 논문을 읽자 !

먼저, input RGB inage를 YCbCr로 변환을 한다.

YCbCr은 Y(휘도)/Cb,Cr(색채,크로마)의 값으로 이루어져 있다.



변환 후 Y, Cb, Cr 각 채널을 Bicubic interpolation으로 원하는 scale factor(어떤 양을 늘리거나 줄이거나 곱하는 수)로 upscale한다.
이때 upscale된 Y 채널 이미지만 VDSR 네트워크의 입력으로 사용한다.
(LR image의 Y 채널 이미지가 VDSR 네트워크를 통과한 이미지)
+(LR image의 Y 채널 이미지) == Output image
Output image와 HR 이미지의 차이를 줄여가기 위해 MSE Loss를 사용하여 학습하게 되며 **VDSR 네트워크의 output은 자연스럽게 Residual image가 된다.(?)**
각각 3*3사이즈의 커널로 이루어져 있다

코드 리뷰
㉠ 모델 코드
- Conv_ReLU_Block 정의

```
class Conv_ReLU_Block(nn.Module):
    def __init__(self):
        super(Conv_ReLU_Block, self).__init__()
        self.conv = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
stride=1, padding=1, bias=False)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        return self.relu(self.conv(x))
```

-Net 정의

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.residual_layer = self.make_layer(Conv_ReLU_Block, 18) # 18개 층 생성
        self.input = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=3,
stride=1, padding=1, bias=False)
        self.output = nn.Conv2d(in_channels=64, out_channels=1, kernel_size=3,
stride=1, padding=1, bias=False)
        self.relu = nn.ReLU(inplace=True)

        # 초기화 방법
        # 모델의 모듈을 차례로 불러온다.
        for m in self.modules():
            if isinstance(m, nn.Conv2d): # isinstance : 비교
                # Filter에서 나오는 부난 root(2/n)로 normalize 한다.
                 n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                 m.weight.data.normal_(0, sqrt(2. / n))

    def make_layer(self, block, num_of_layer):
        layers = []
        for _ in range(num_of_layer):
            layers.append(block())
        return nn.Sequential(*layers)

    def forward(self, x):
        residual = x
        out = self.relu(self.input(x))
```

```
        out = self.residual_layer(out)
        out = self.output(out)
        out = torch.add(out,residual)
        return out
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 256]             576
              ReLU-2         [-1, 64, 256, 256]               0
            Conv2d-3         [-1, 64, 256, 256]          36,864
              ReLU-4         [-1, 64, 256, 256]               0
  Conv_ReLU_Block-5          [-1, 64, 256, 256]               0
            Conv2d-6         [-1, 64, 256, 256]          36,864
              ReLU-7         [-1, 64, 256, 256]               0
  Conv_ReLU_Block-8          [-1, 64, 256, 256]               0
            Conv2d-9         [-1, 64, 256, 256]          36,864
             ReLU-10         [-1, 64, 256, 256]               0
 Conv_ReLU_Block-11          [-1, 64, 256, 256]               0
           Conv2d-12         [-1, 64, 256, 256]          36,864
             ReLU-13         [-1, 64, 256, 256]               0
 Conv_ReLU_Block-14          [-1, 64, 256, 256]               0
           Conv2d-15         [-1, 64, 256, 256]          36,864
             ReLU-16         [-1, 64, 256, 256]               0
...
 Conv_ReLU_Block-50          [-1, 64, 256, 256]               0
           Conv2d-51         [-1, 64, 256, 256]          36,864
             ReLU-52         [-1, 64, 256, 256]               0
 Conv_ReLU_Block-53          [-1, 64, 256, 256]               0
           Conv2d-54         [-1, 64, 256, 256]          36,864
             ReLU-55         [-1, 64, 256, 256]               0
 Conv_ReLU_Block-56          [-1, 64, 256, 256]               0
           Conv2d-57          [-1, 1, 256, 256]             576
================================================================
Total params: 664,704
Trainable params: 664,704
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.25
Forward/backward pass size (MB): 1792.50
Params size (MB): 2.54
Estimated Total Size (MB): 1795.29
----------------------------------------------------------------
```

input, output, hidden 합쳐서 20개의 layer로 계산 후 제일 마지막에 자기 자신과 더해주고 출력해준다.

㉡ 수행 과정
1. 필요한 패키지 불러오기

```
# Load the package you are going to use
import torch
from torch.autograd import Variable
from PIL import Image
import numpy as np
import time, math
import matplotlib.pyplot as plt
%matplotlib inline

# Load the pretrained model
model = torch.load("model/model_epoch_50.pth")["model"]
```

2. 이미지 불러오고, y만 잘라주기 (input 준비)

```
# Load the groundtruth image and the low-resolution image (downscaled with a factor of 4)
im_gt = Image.open("Set5/butterfly_GT.bmp").convert("RGB")
im_b = Image.open("Set5/butterfly_GT_scale_4.bmp").convert("RGB")
```

im_gt 가 원본 이미지
im_b가 bicubic scale 4배로 만든 이미지

```
# Convert the images into YCbCr mode and extraction the Y channel
(for PSNR calculation)
im_gt_ycbcr = np.array(im_gt.convert("YCbCr"))
im_b_ycbcr = np.array(im_b.convert("YCbCr"))
im_gt_y = im_gt_ycbcr[:,:,0].astype(float)
im_b_y = im_b_ycbcr[:,:,0].astype(float)
```

RGB 채널을 YCbCr 채널로 바꾸어준 다음, input 데이터를 만들어준다.

3. PSNR 정의

```
# Here is the function for PSNR calculation
def PSNR(pred, gt, shave_border=0):
    height, width = pred.shape[:2]
    pred = pred[shave_border:height - shave_border, shave_border:width - shave_border]
    gt = gt[shave_border:height - shave_border, shave_border:width - shave_border]
    imdff = pred - gt
    rmse = math.sqrt(np.mean(imdff ** 2))
    if rmse == 0:
```

```
        return 100
    return 20 * math.log10(255.0 / rmse)
```

단순히 BICUBIC만 쓴 이미지에 대해서는 PSNR을 구하면 다음과 같다.

```
psnr_bicubic = PSNR(im_gt_y, im_b_y)
print('psnr for bicubic is {}dB'.format(psnr_bicubic))
psnr for bicubic is 20.82464896dB
```

4. input을 dataloader에 잘 들어갈 수 있게 가공

```
# Prepare for the input, a pytorch tensor

im_input = im_b_y/255. # 현재 type이 numpy임
im_input = Variable(torch.from_numpy(im_input).float()).\
    view(1, -1, im_input.shape[0], im_input.shape[1])

print(im_input)
```

```
torch.Size([1, 1, 256, 256])
```

5. weight가 저장된 pth를 이용하여 모델링

1) cpu()로 비교

```
# Let's try the network feedforward in cpu mode
model = model.cpu()

# Let's see how long does it take for processing
start_time = time.time()
out = model(im_input)
elapsed_time = time.time() - start_time
print("It takes {}s for processing in cpu mode".format(elapsed_time))
```

```
It takes 0.6975526809692383s for processing in cpu mode
```

2) cuda()로 비교

```
# Now let's try the network feedforward in gpu mode
model = model.cuda()
im_input = im_input.cuda()

# Let's see how long does it take for processing in gpu mode
start_time = time.time()
out = model(im_input)
elapsed_time = time.time() - start_time
print("It takes {}s for processing in gpu mode".format(elapsed_time))
```

```
It takes 0.4577970504760742s for processing in gpu mode
```

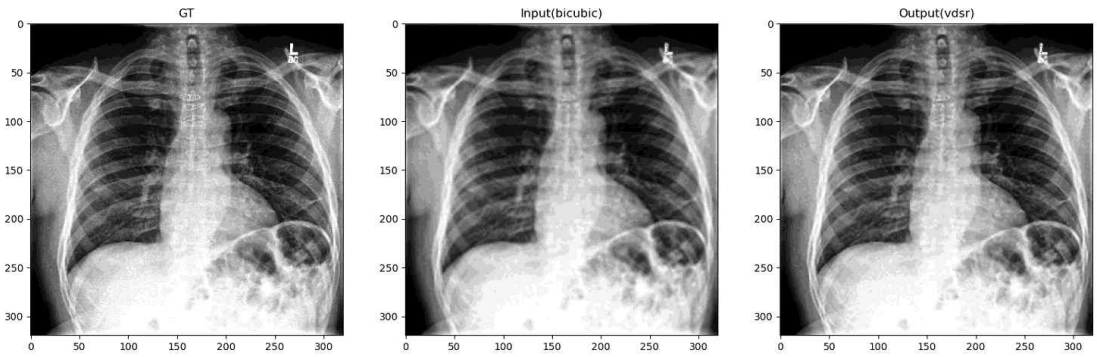
6. output 이미지에 대해서 결과값 내기

```
# Get the output image
out = out.cpu()
im_h_y = out.data[0].numpy().astype(np.float32)
im_h_y = im_h_y * 255.
# 0보다 작은 값은 0으로 255보다 큰 값은 255로 해주기
im_h_y[im_h_y < 0] = 0
im_h_y[im_h_y > 255.] = 255.
im_h_y = im_h_y[0,:,:]

# Calculate the PNSR different between bicubic interpolation and vdsr prediction
print("PSNR improvement is {}dB".format(psnr_predicted - psnr_bicubic))
```

```
PSNR improvement is 4.604248017945615dB
```

결과값 비교



PSNR_predicted= 30.93538582980297
PSNR_bicubic= 29.834132107702764
It takes 0.4311039447784424s for processing

Reference

https://github.com/2KangHo/vdsr_pytorch
https://hwangtoemat.github.io/paper-review/2019-07-17-SRCNN-%EC%BD%94%EB%93%9C/