EC5303, Computer Vision, 2021 Fall

Programming Assignment #3

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

Christian Ledig et al

CVPR 2017

Presenter

20212047, Yechan Kim | yechankim@gm.gist.ac.kr

Please read ‘README.md’ before scoring.

README.md

SRGAN

Unofficial Implementation for "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Model"

Yechan Kim (20212047)

Prerequisites

- See `requirements.txt` for details.

```
torch >= 1.10.0
torchvision >= 0.11.1
glob2
numpy
matplotlib
tqdm
tensorboard
tensorboardx
```

Toy-Examples

- This repository provides DIV2K (with only few examples) and Set5 for convenience. Note that given DIV2K has only few examples, thus you need to download the full DIV2K dataset for training.
- You can download DIV2K or other datasets for super-resolution task via internet.

...

(Recap) Super-resolution?

Super-resolution

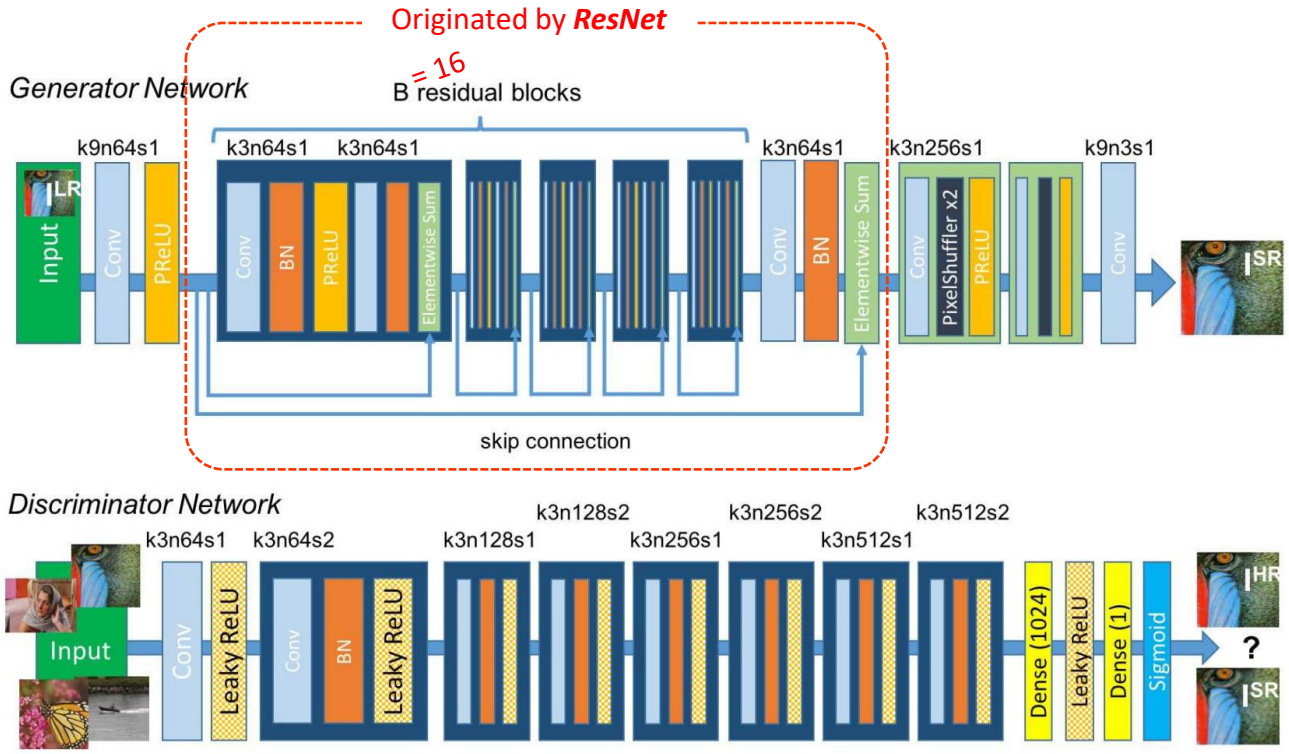
- Super-resolution is the task of taking an input of a low resolution (LR) and **upscaling it to that of a high resolution**.

- The **ill-posed nature of the underdetermined SR problem** is particularly pronounced for high upscaling factors, for which **texture detail** in the reconstructed SR images is typically absent [1].

Reference:
[1] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

(Recap) SRGAN?

- Architectures



k : kernel size, n : number of feature maps, s : strides for conv

- Avoid using any pooling layers.
- Generative network (***SRResNet***) is pre-trained alone first before adversarial training.
- Then, this work alternatively trains the discriminator and generator. → *Details on the next slide!*

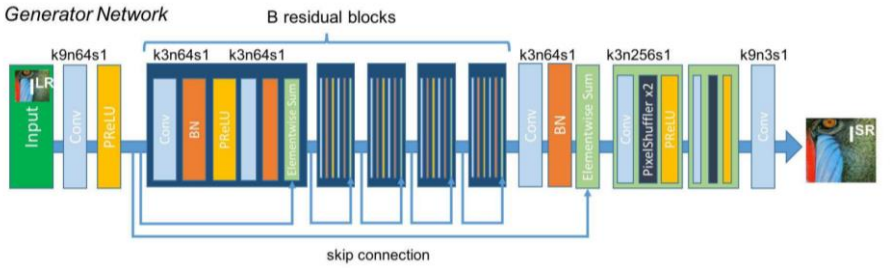
Reference:
[1] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

(Recap) SRGAN? [Code]

- SRGAN/Model/Generator.py

44 lines (41 sloc) | 2.03 KB — Raw | Blame

```python
from torch import nn
from Model.Block import ResidualBlock, list_to_sequential
from Model.Block import UpsamlingBlock
import numpy as np


class Generator(nn.Module):
    def __init__(self, in_channels=3, num_residual_blocks=16):
        super(Generator, self).__init__()                                   # (in, out)
        num_upsampling_blocks = 2
        out_channels = 64
        self.input_block = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=(9, 9),       # ( 3, 64)
                      stride=1, padding=4, bias=False),
            nn.PReLU()
        )
        self.residual_blocks = list_to_sequential(
            [ResidualBlock(in_channels=out_channels, out_channels=out_channels)                     # (64, 64)
             for _ in range(num_residual_blocks)]
        )
        self.intermediate_block = nn.Sequential(
            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=(3, 3),      # (64, 64)
                      stride=1, padding=1, bias=False),
            nn.BatchNorm2d(num_features=out_channels)
        )  # for connecting residual blocks with upsampling blocks!
        self.upsampling_blocks = list_to_sequential(
            [UpsamlingBlock(in_channels=out_channels,                   # (64, 64 * num_upsampling_blocks ^ 2)
                            scaling_factor=num_upsampling_blocks)
             for _ in range(num_upsampling_blocks)]
        )
        self.output_block = nn.Sequential(
            nn.Conv2d(in_channels=out_channels, out_channels=in_channels, kernel_size=(9, 9),
                      stride=1, padding=4, bias=False),
            # we might add extra non-linear activation here!
        )
        self.short_cut = nn.Sequential()

    def forward(self, x):
        identity = self.input_block(x)
        out = self.residual_blocks(identity)
        out = self.short_cut(identity) + self.intermediate_block(out)
        out = self.upsampling_blocks(out)
        out = self.output_block(out)
        return out
```



SRGAN/Model/Block.py

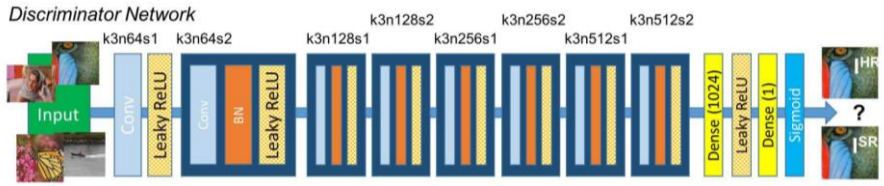
56 lines (45 sloc) | 2.11 KB — Raw | Blame

```python
from torch import nn


class ResidualBlock(nn.Module):  # for Generator (1)
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size,
                               stride=stride, padding=padding, bias=False)
        self.conv2 = nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=kernel_size,
                               stride=stride, padding=padding, bias=False)
        self.bn = nn.BatchNorm2d(num_features=out_channels)
        self.p_relu = nn.PReLU()
        self.short_cut = nn.Sequential()

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn(out)
        out = self.p_relu(out)
        out = self.conv2(out)
        out = self.bn(out)
        out = self.short_cut(identity) + out
        return out


class UpsamlingBlock(nn.Module):  # for Generator (2)
    def __init__(self, in_channels, scaling_factor, kernel_size=(3, 3), stride=1, padding=1):
        super(UpsamlingBlock, self).__init__()
        out_channels = in_channels * (scaling_factor ** 2)  # -> Why?
        self.conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=kernel_size,
                              stride=stride, padding=padding, bias=False)
        self.pixel_shuffle = nn.PixelShuffle(scaling_factor)
        self.p_relu = nn.PReLU()

    def forward(self, x):
        out = self.conv(x)
        out = self.pixel_shuffle(out)
        out = self.p_relu(out)
        return out
```

...

```python
def list_to_sequential(layer_list):
    return nn.Sequential(*layer_list)
```

(Recap) SRGAN? [Code]



- SRGAN/Model/Discriminator.py

33 lines (30 sloc) | 1.28 KB — Raw | Blame

```python
from torch import nn
from Model.Block import ConvBlock, list_to_sequential


class Discriminator(nn.Module):
    def __init__(self, in_channels=3, conv_in_out_chs_strides=None):
        super(Discriminator, self).__init__()
        self.linear_width = 1024
        if not conv_in_out_chs_strides:
            conv_in_out_chs_strides = [(64, 64, 2), (64, 128, 1), (128, 128, 2),
                                       (128, 256, 1), (256, 512, 1), (512, 512, 2)]
        self.input_block = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=64, kernel_size=(3, 3),
                      stride=1, padding=1, bias=False),
            nn.LeakyReLU(0.2)
        )
        self.conv_blocks = list_to_sequential(
            [ConvBlock(in_channels=in_ch, out_channels=out_ch, stride=stride)
             for in_ch, out_ch, stride in conv_in_out_chs_strides]
        )
        self.output_block = nn.Sequential(
            nn.LazyLinear(out_features=self.linear_width),
            nn.LeakyReLU(0.2),
            nn.AdaptiveAvgPool2d(1),
            nn.Sigmoid()
        )

    def forward(self, x):
        out = self.input_block(x)
        out = self.conv_blocks(out)
        out = out.view(-1, self.linear_width)
        out = self.output_block(out)
        return out
```

SRGAN/Model/Block.py

56 lines (45 sloc) | 2.11 KB — Raw | Blame

```python
from torch import nn
```

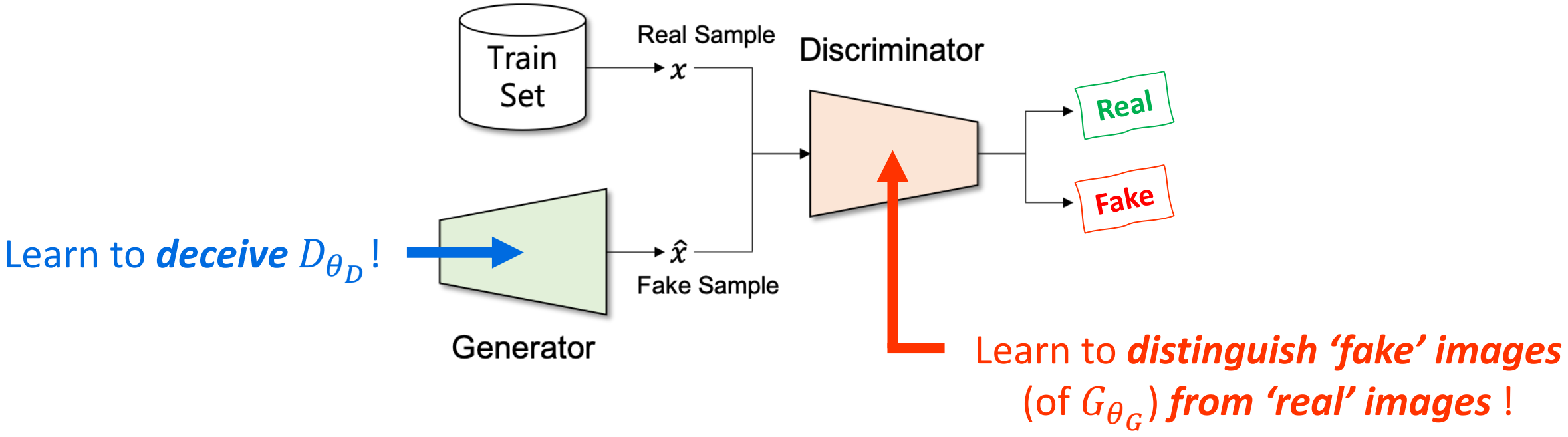
...

```python
class ConvBlock(nn.Module):  # for Discriminator
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super(ConvBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(0.2)
        )

    def forward(self, x):
        return self.block(x)


def list_to_sequential(layer_list):
    return nn.Sequential(*layer_list)
```

(Recap) SRGAN?

- Training Objective
  - Let I^{LR} and I^{HR} be images with low resolution and high resolution, respectively.
  - Let G_{θ_G} and D_{θ_D} be the generative and discriminator networks, respectively.
  - The optimization goal is to solve the adversarial min-max problem as follows:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right].$$



Reference:
[2] https://kh-kim.gitbook.io/natural-language-processing-with-pytorch/00-cover-11/01-intro

(Recap) SRGAN?

- Training Objective (cont.)

Discriminator

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$$

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}\left[\underbrace{\log D_{\theta_D}(I^{HR})}\right] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$$

Maximum when $D_{\theta_D}(I^{HR}) = 1$

($\because$ Range of $D_{\theta_D}(\cdot)$: [0, 1].)

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\underbrace{\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))}\right]$$

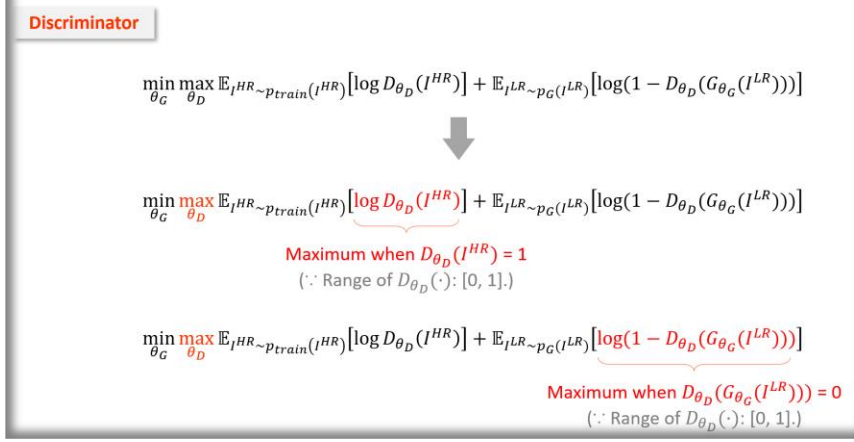
Maximum when $D_{\theta_D}(G_{\theta_G}(I^{LR}))) = 0$

($\because$ Range of $D_{\theta_D}(\cdot)$: [0, 1].)

(Recap) SRGAN?

- Training Objective (cont.)

Generator



$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})]+\mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR})))]$$

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})]+\mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR})))]$$

→ G_{θ_G} is independent of this term.

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})]+\mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\underbrace{\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR})))}]$$

Minimum when $D_{\theta_D}(G_{\theta_G}(I^{LR}))=1$
(∵ $\log 0 = -\infty$.)

(Recap) Perceptual Loss?

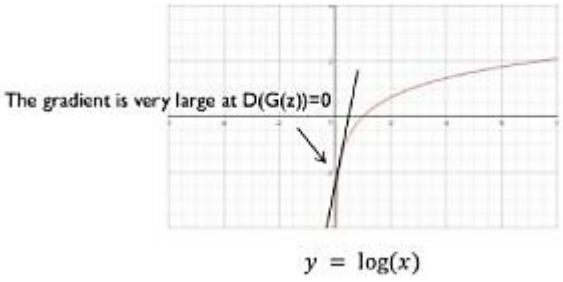
- [1] adopts a **perceptual loss** function for training the generator as follows:

$$l^{SR} = l_X^{SR} + 10^{-3}\, l_{gen}^{SR}.$$

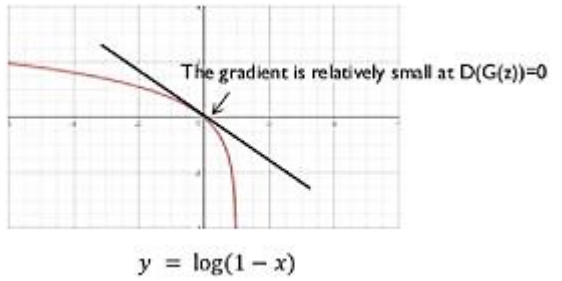
  - l_X^{SR}: Content Loss
    - This work uses $l_{VGG\,/\,i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y}\right)^2$,

      where $W_{i,j}$ and $H_{i,j}$ are the dimensions of the respective feature map $\phi_{i,j}$
      which is obtained by j-th convolution (after activation) before i-th max-pooling layer within VGG-19.

  - l_{GEN}^{SR}: Adversarial Loss
    - $I_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$



$I_{Gen}^{SR} = \sum_{n=1}^{N} \log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))$

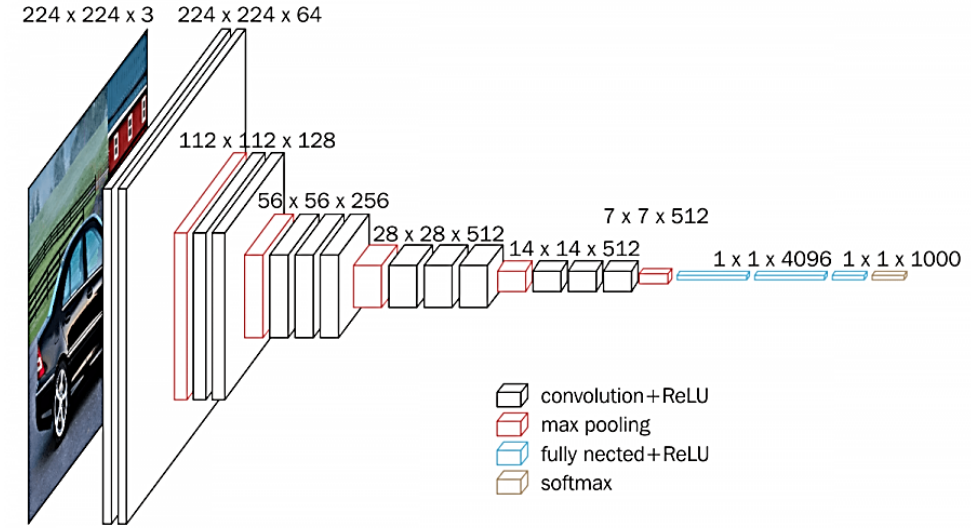


Reference:
[1] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
[3] https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

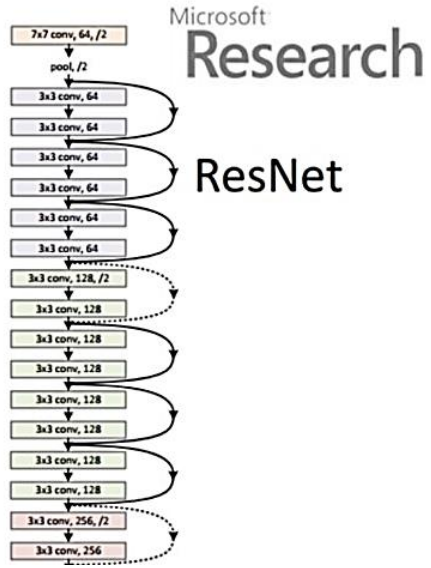
(+α) How to Enhance the Content Loss?

- l_X^{SR}: Content Loss
  - $l_{VGG\ /\ i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y} \right)^2$.

→ How about designing the 'content loss' based on ***ResNet* [4]**?



VGGNet



ResNet

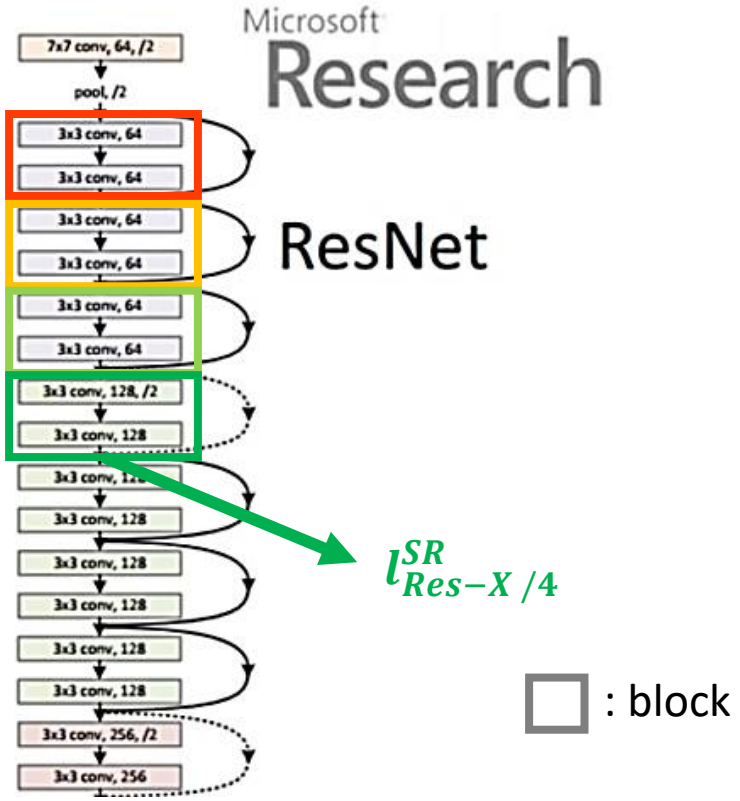
Reference:
[1] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
[4] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

(+α) How to Enhance the Content Loss? (cont.)

$$I^{SR}_{VGG\,/\,i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y} \right)^2$$

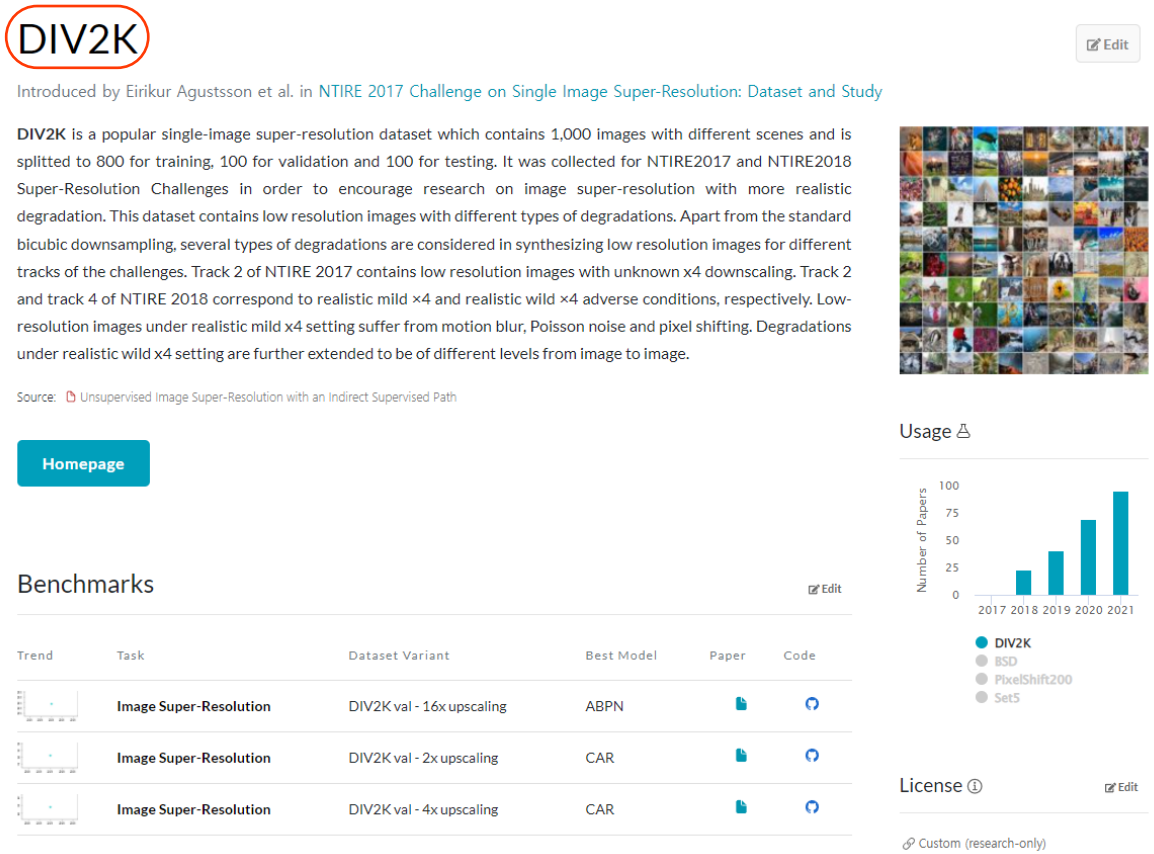
- l^{SR}_{X}: Content Loss (**Proposed**)
  - $l^{SR}_{Res-X\,/\,i} = \frac{1}{W_i H_i} \sum_{x=1}^{W_i} \sum_{y=1}^{H_i} \left(\phi_i(I^{HR})_{x,y} - \phi_i\left(G_{\theta_G}(I^{LR})\right)_{x,y} \right)^2$,

    where W_i and H_i are the dimensions of the respective feature map ϕ_i which is obtained by 'i-th res-block' within ResNet-X.



Datasets

- For training the SRResNet and SRGAN models,



DIV2K

Edit

Introduced by Eirikur Agustsson et al. in NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study

DIV2K is a popular single-image super-resolution dataset which contains 1,000 images with different scenes and is splitted to 800 for training, 100 for validation and 100 for testing. It was collected for NTIRE2017 and NTIRE2018 Super-Resolution Challenges in order to encourage research on image super-resolution with more realistic degradation. This dataset contains low resolution images with different types of degradations. Apart from the standard bicubic downsampling, several types of degradations are considered in synthesizing low resolution images for different tracks of the challenges. Track 2 of NTIRE 2017 contains low resolution images with unknown x4 downscaling. Track 2 and track 4 of NTIRE 2018 correspond to realistic mild ×4 and realistic wild ×4 adverse conditions, respectively. Low-resolution images under realistic mild x4 setting suffer from motion blur, Poisson noise and pixel shifting. Degradations under realistic wild x4 setting are further extended to be of different levels from image to image.

Source: Unsupervised Image Super-Resolution with an Indirect Supervised Path

Homepage

Benchmarks

Edit

Trend	Task	Dataset Variant	Best Model	Paper	Code
	Image Super-Resolution	DIV2K val - 16x upscaling	ABPN		
	Image Super-Resolution	DIV2K val - 2x upscaling	CAR		
	Image Super-Resolution	DIV2K val - 4x upscaling	CAR		

Usage

License

Edit

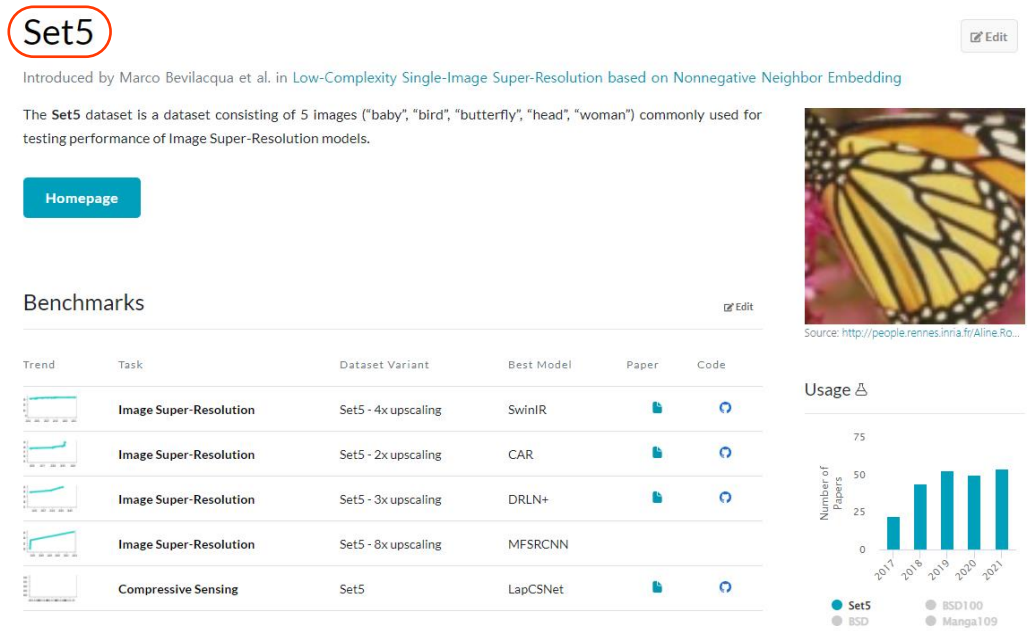
Custom (research-only)

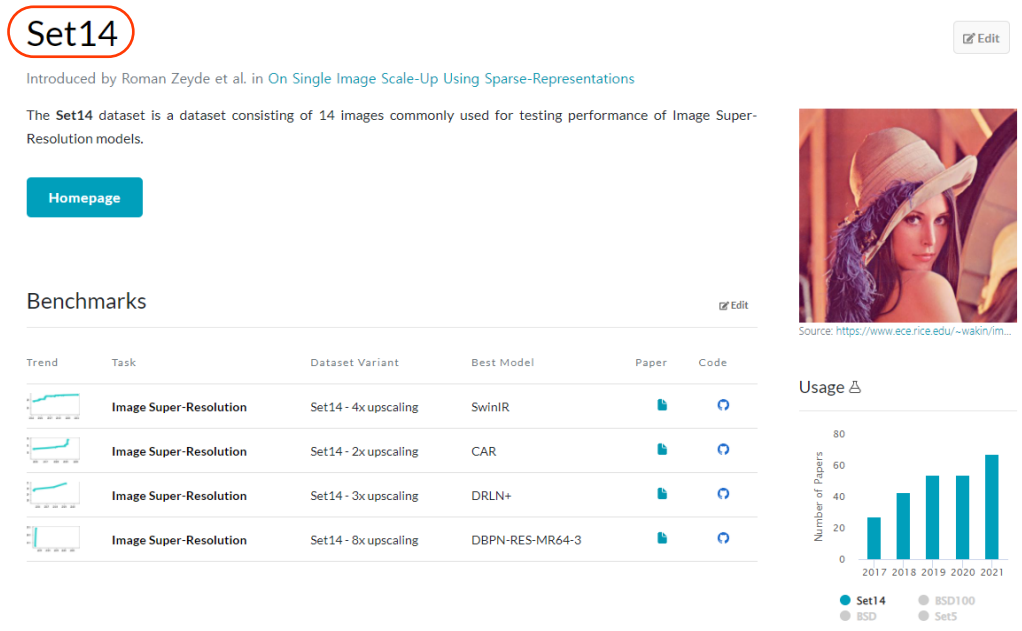
- Train: DIV2K_train_HR/*.png
- Validation: DIV2K_valid_HR/*.png

Reference:
[5] https://paperswithcode.com/dataset/div2k

Datasets

- For evaluating the SRResNet and SRGAN models,





- HR: Set5/GTmod12/*.png
- LR: Set5/LRbicx4/*.png

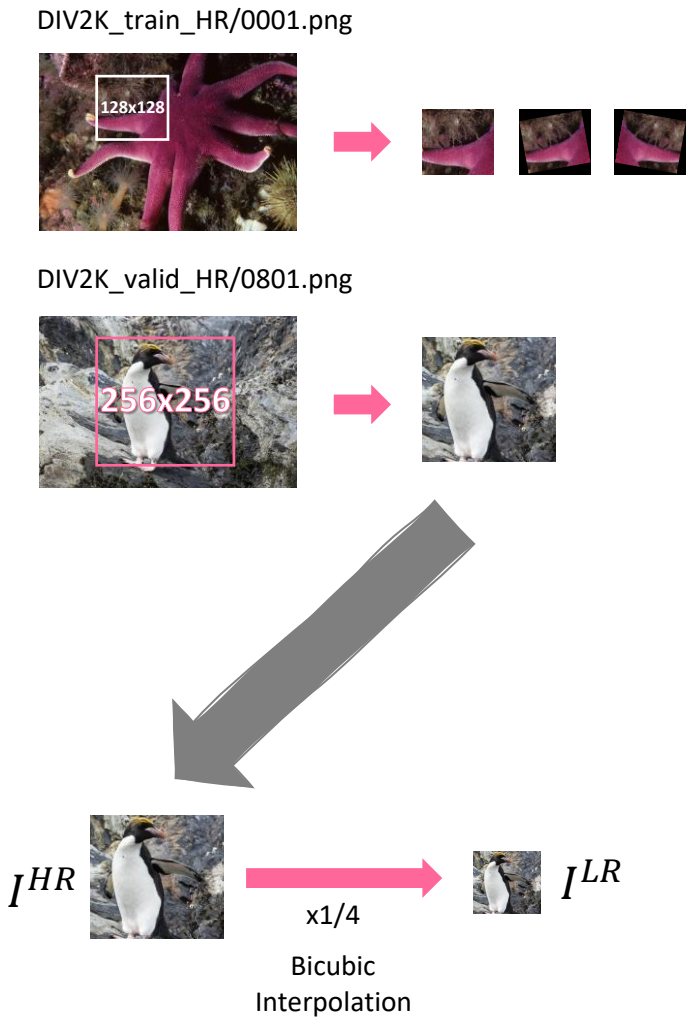
- HR: Set14/GTmod12/*.png
- LR: Set14/LRbicx4/*.png

Reference:
[6] https://paperswithcode.com/dataset/set5
[7] https://paperswithcode.com/dataset/set11

How to Load HR and LR Images?

- HR (High Resolution) images
  - This work only downloads **DIV2K_train_HR** and **DIV2K_valid_HR**.
  - Zero-padding, Random-cropping, Random-rotation, and horizontal-flipping are applied to training set of DIV2K.
    - 128x128 random-cropping
    - 10-degree random-rotation
    - Horizontal flipping with probability 0.5
  - Only center-cropping is applied to validation set of DIV2K.
    - 256x256 center-cropping

- LR (Low Resolution) images
  - LR images are be generated via resizing pre-processed HR images by a quarter (1/4) for both training and validation.
  - Bicubic interpolation is used for reducing the size of images.



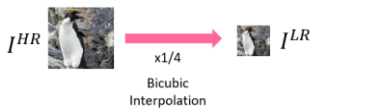
How to Load HR and LR Images?

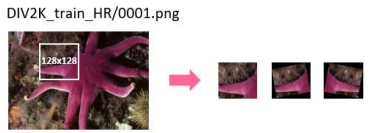
[Code]

SRGAN/Dataset/dataset.py

99 lines (88 sloc) | 3.71 KB — Raw | Blame

```python
from PIL import Image

import glob
from torch.utils.data import Dataset
from torchvision.transforms import transforms, InterpolationMode


class HR_to_HR_LR_PairDataset(Dataset):
    def __init__(self, data_path, mode, crop_size=128, rescale_ratio=4):
        self.img_files = sorted(glob.glob(f'{data_path}/*.*'))
        assert len(self.img_files) > 0, "No images exist in the given dataset path."
        self.LR_transforms = self.get_LR_transforms(crop_size, rescale_ratio)
        self.HR_transforms = self.get_HR_transforms(mode, crop_size)

    def get_LR_transforms(self, crop_size, rescale_ratio):
        return transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize(crop_size // rescale_ratio, interpolation=InterpolationMode.BICUBIC),
            transforms.ToTensor()
        ])

    def get_HR_transforms(self, mode, crop_size):
        if mode == 'train':
            return transforms.Compose([
                transforms.RandomCrop(crop_size),
                transforms.RandomRotation(10),
                transforms.RandomHorizontalFlip(0.5),
                transforms.ToTensor()
            ])
        else:
            return transforms.Compose([
                transforms.CenterCrop(crop_size),
                transforms.ToTensor()
            ])

    def __len__(self):
        return len(self.img_files)

    def __getitem__(self, idx):
        hr_img = self.HR_transforms(Image.open(self.img_files[idx]))
        lr_img = self.LR_transforms(hr_img)
        return lr_img, hr_img
```





Training Details for SRResNet

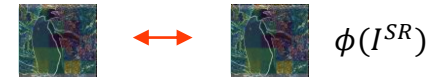
- SRResNet models are trained for 250 epochs on DIV2K.
- During training, the learning rate is 0.0001 and no learning rate scheduler is adopted.
- ADAM optimizer is used with betas=(0.9, 0.999).
- This work considers various training objectives as follows:
  - [Default] MSE Loss

$$l_{MSE}^{SR} = \frac{1}{WH}\sum_{x=1}^{W}\sum_{y=1}^{H}\left((I^{HR})_{x,y} - \left(G_{\theta_G}(I^{LR})\right)_{x,y}\right)^2,$$

where W and H are the dimensions of the images.

  - [**+a**] Content Loss – VGG Loss

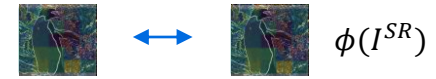
$$l_{VGG\,/\,i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}\left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y}\right)^2,$$



where $W_{i,j}$ and $H_{i,j}$ are the dimensions of the respective feature map $\phi_{i,j}$
which is obtained by j-th convolution (after activation) before i-th max-pooling layer within VGG-19.

  - [**+a**] Content Loss – ResNet Loss

$$l_{Res-X\,/\,i}^{SR} = \frac{1}{W_iH_i}\sum_{x=1}^{W_i}\sum_{y=1}^{H_i}\left(\phi_i(I^{HR})_{x,y} - \phi_i\left(G_{\theta_G}(I^{LR})\right)_{x,y}\right)^2,$$



where W_i and H_i are the dimensions of the respective feature map ϕ_i which is obtained by 'i-th res-block' within ResNet-X.

Training Details for SRResNet

- [**+a**] Content Loss – VGG Loss **[Code]**

$\phi(I^{HR})$   $\phi(I^{SR})$

$$l^{SR}_{VGG\,/\,i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}\left(G_{\theta_G}(I^{LR})\right)_{x,y} \right)^2,$$

where $W_{i,j}$ and $H_{i,j}$ are the dimensions of the respective feature map $\phi_{i,j}$

which is obtained by j-th convolution (after activation) before i-th max-pooling layer within VGG-19.

SRGAN/ContentLoss/VGGLoss.py

42 lines (37 sloc) | 1.62 KB Raw Blame

```python
import torch
import torch.nn as nn


def get_extractor_for_VGG(torch_model, i=5, j=4):
    ''' [Logic]
    extract the feature extractor from torch_model (vgg-model)
    here, the extractor should include from 1-st layer to the 'j'-th convolutional layer
          before 'i'-th max-pooling layer.
    '''
    layers = list(torch_model.children())[0]
    last_conv_idx_list = []
    last_maxpool_cnt = 0
    for idx, layer in enumerate(layers):
        layer.requires_grad = False  # freezing!
        if last_maxpool_cnt == i:
            break
        if 'MaxPool2d' in str(layer):
            last_maxpool_cnt += 1
        elif 'Conv2d' in str(layer):
            last_conv_idx_list.append(idx + 1)
    final_layers = layers[0:last_conv_idx_list[-(j + 1)]]
    return nn.Sequential(*final_layers)
```

```python
class VGGLoss(nn.Module):
    def __init__(self, i=5, j=4, device='cuda:0', vgg_model_name='vgg19_bn'):
        ''' [Logic]
        1) extract the feature map from the 'j'-th convolutional layer before 'i'-th max-pooling layer
           for both hr_image (ground_truth) and sr_image (prediction).
        2) compute and return mse_loss between features maps of hr_image & sr_image.
        '''
        super(VGGLoss, self).__init__()
        vgg = torch.hub.load('pytorch/vision:v0.10.0', vgg_model_name, pretrained=True).to(device).eval()
        self.extractor = get_extractor_for_VGG(vgg, i, j)
        self.mse_loss = nn.MSELoss()

    def forward(self, hr_img, sr_img):
        hr_feature_map = self.extractor(hr_img)
        sr_feature_map = self.extractor(sr_img)
        vgg_loss = self.mse_loss(hr_feature_map, sr_feature_map)
        return vgg_loss
```

Training Details for SRResNet

- [**+a**] Content Loss – ResNet Loss **[Code]**

$$l^{SR}_{Res-X\ /\ i} = \frac{1}{W_i H_i} \sum_{x=1}^{W_i} \sum_{y=1}^{H_i} \left(\phi_i(I^{HR})_{x,y} - \phi_i\left(G_{\theta_G}(I^{LR})\right)_{x,y} \right)^2,$$

$\phi(I^{HR})$  $\leftrightarrow$  $\phi(I^{SR})$

where W_i and H_i are the dimensions of the respective feature map ϕ_i which is obtained by 'i-th res-block' within ResNet-X.

SRGAN/ContentLoss/ResNetLoss.py

54 lines (48 sloc) | 1.79 KB — Raw | Blame

```python
import torch
import torch.nn as nn


def get_extractor_for_ResNet(torch_model, i=6):
    ''' [Logic]
    extract the feature extractor from torch_model (ResNet-model)
    here, the extractor should include from 1-st layer to 'i'-th residual block.
    '''
    layers = list(torch_model.children())
    pre_layers = []
    post_blocks = []

    # for pre_layers
    for layer in layers:
        if type(layer) == nn.Sequential:
            break
        pre_layers.append(layer)
    # for post_blocks
    cnt = 0
    for sequence in layers[5:]:
        if type(sequence) == nn.Sequential:
            for block in sequence:
                post_blocks.append(block)
                cnt += 1
                if cnt > i:
                    break
    # freezing (1)
    for layer in pre_layers:
        layer.requires_grad = False
    # freezing (2)
    for block in post_blocks:
        block.requires_grad = False
    final_layers = pre_layers + post_blocks
    return nn.Sequential(*final_layers)
```

```python
class ResNetLoss(nn.Module):
    def __init__(self, i=5, device='cuda:0', resnet_model_name='resnet34'):
        ''' [Logic]
        1) extract the feature map from the 'i'-th residual block
           for both hr_image (ground_truth) and sr_image (prediction).
        2) compute and return mse_loss between features maps of hr_image & sr_image.
        '''
        super(ResNetLoss, self).__init__()
        resnet = torch.hub.load('pytorch/vision:v0.10.0', resnet_model_name, pretrained=True).to(device).eval()
        self.extractor = get_extractor_for_ResNet(resnet, i)
        self.mse_loss = nn.MSELoss()

    def forward(self, hr_img, sr_img):
        hr_feature_map = self.extractor(hr_img)
        sr_feature_map = self.extractor(sr_img)
        vgg_loss = self.mse_loss(hr_feature_map, sr_feature_map)
        return vgg_loss
```

SRResNet [Code]

SRGAN/1_SRResNet/SRResNet_train.py

61 lines (52 sloc) | 2.56 KB

Raw Blame

```python
import sys
sys.path.append('../')

import datetime

from torch import optim
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

from Dataset.dataset import HR_to_HR_LR_PairDataset
from Model.Generator import Generator
from SRResNet_parser import Parser
from SRResNet_utils import *


if __name__ == '__main__':
    # Parser
    my_parser = Parser(mode='train')
    my_args = my_parser.parser.parse_args()
    print('*', my_args)

    # Tag
    cur_time = datetime.datetime.now().strftime('%y-%m-%d-%H-%M-%S')
    tag_name = f'{my_args.tag}-{cur_time}' if my_args.tag else f'{cur_time}'
    print('[SRResNet]')
    print(f'{tag_name} experiment has been started.')

    # Parameter
    train_dir = my_args.train_dir
    valid_dir = my_args.valid_dir
    device = f'cuda:{my_args.gpu_index}' if my_args.gpu_index >= 0 else 'cpu'

    # Loader (Train / Valid)
    train_dataset = HR_to_HR_LR_PairDataset(train_dir, mode='train', crop_size=128)
    valid_dataset = HR_to_HR_LR_PairDataset(valid_dir, mode='eval', crop_size=256)
    train_loader = DataLoader(train_dataset, my_args.batch_size, shuffle=True, pin_memory=True)
    valid_loader = DataLoader(valid_dataset, my_args.batch_size, shuffle=False, pin_memory=True)
    # Train Generator
    # if my_args.train_generator:
    print('[Train a generator]')
    generator = Generator(in_channels=3, scaling_factor=4, num_residual_blocks=16)
    generator_best_model_state = generator.state_dict()
    g_optimizer = optim.Adam(generator.parameters(), lr=my_args.g_lr, betas=(0.9, 0.999))
    db_losses = {'train': [], 'valid': []}
    tensorboard_writer = SummaryWriter(f'./log_dir/{tag_name}')
    for epoch in range(my_args.epochs):
        train_mse, valid_mse = \
            train_and_validate_generator(generator, g_optimizer, epoch, device,
                                         train_loader, valid_loader, my_args.loss_function, tensorboard_writer)
        if db_losses['valid'] and \
                valid_mse < min(db_losses['valid']):  # store the best model so far.
            generator_best_model_state = generator.state_dict()
        db_losses['train'].append(train_mse)
        db_losses['valid'].append(valid_mse)
    torch.save(generator_best_model_state, f'SRResNet-generator-{tag_name}.pt')
    print(f'SRResNet-generator-{tag_name}.pt is stored.')
    write_log_for_SRResNet(db_losses, f'SRResNet-generator-{tag_name}-log.txt')
    print(f'SRResNet-generator-{tag_name}-log.txt is stored.')
    plot_log_for_SRResNet(db_losses, f'SRResNet-generator-{tag_name}-log.png')
    print(f'SRResNet-generator-{tag_name}-log.png is stored.')
```

SRResNet [Code]

SRGAN/1_SRResNet/SRResNet_utils.py

```python
def train_and_validate_generator(generator, g_optimizer, epoch, device,
                                 train_loader, valid_loader, loss_function_name, tensorboard_writer):
    generator.to(device)
    loss_function = nn.MSELoss().to(device)
    if loss_function_name == 'vgg_loss_19_5_4':
        loss_function = VGGLoss(i=5, j=4, vgg_model_name='vgg19_bn')
    elif loss_function_name == 'res_loss_18_4':
        loss_function = ResNetLoss(i=4, resnet_model_name='resnet18')
    elif loss_function_name == 'res_loss_34_3':
        loss_function = ResNetLoss(i=3, resnet_model_name='resnet34')
    elif loss_function_name == 'res_loss_34_5':
        loss_function = ResNetLoss(i=5, resnet_model_name='resnet34')

    # (1) train phase
    train_mse = 0
    generator.train()
    train_tqdm_loader = tqdm.tqdm(train_loader)
    for i, (lr_img, hr_img) in enumerate(train_tqdm_loader):
        lr_img = lr_img.to(device)
        hr_img = hr_img.to(device)

        sr_img = generator(lr_img)
        train_loss = loss_function(hr_img, sr_img)
        train_mse += train_loss.item()

        # optimization
        g_optimizer.zero_grad()
        train_loss.backward()
        g_optimizer.step()

        train_tqdm_loader.set_description(f'Train-SRResNet | Epoch: {epoch + 1} | Loss: {train_mse / (i + 1): .4f}')

        # tensorboard log
    train_mse /= len(train_tqdm_loader)
    tensorboard_writer.add_scalar('Loss/train', train_mse, epoch + 1)
    time.sleep(0.01)
    # (2) validation phase
    valid_mse = 0
    generator.eval()
    valid_tqdm_loader = tqdm.tqdm(valid_loader)
    with torch.no_grad():
        for i, (lr_img, hr_img) in enumerate(valid_tqdm_loader):
            lr_img = lr_img.to(device)
            hr_img = hr_img.to(device)

            sr_img = generator(lr_img)
            valid_loss = loss_function(hr_img, sr_img)
            valid_mse += valid_loss.item()

            valid_tqdm_loader.set_description(f'Valid-SRResNet | Epoch: {epoch + 1} | '
                                              f'Loss: {valid_mse / (i + 1): .4f}')

    valid_mse /= len(valid_tqdm_loader)
    tensorboard_writer.add_scalar('Loss/valid', valid_mse, epoch + 1)

    return train_mse, valid_mse
```

Training Details for SRGAN

- During training, the learning rate is 0.0001 and no learning rate scheduler is adopted.
- Two ADAM optimizers are used for both discriminator and generator.
- ADAM optimizer is used with betas=(0.9, 0.999).

- SRGAN models are trained for 100 / 200 epochs on DIV2K.
  - We consider following content losses as follows:
    - (1) MSE [+a]
    - (2) VGG Loss [Default]
    - (3) ResNet Loss [+a]

→ *See the above slide for more details!*

- Discriminator
  - Training objective
    - $\max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))]$
    - $\Leftrightarrow \min_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}[-\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[-\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))]$ (For PyTorch Implementation)

- Generator

  ⇔ SRResNet
  - Generator is initialized with **pretrained weights** of SRResNet (w/ MSE Loss).
  - Training objective
    - $\min_{\theta_G} \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))]$
    - $\Leftrightarrow \min_{\theta_G} \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[-\log(D_{\theta_D}(G_{\theta_G}(I^{LR})))]$ (For PyTorch Implementation)

Training Details for SRGAN

- Discriminator
  - Training objective **[Code]**
    - $\max_{\theta_D} \mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}\left[\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$

      $\Leftrightarrow \min_{\theta_D} \mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}\left[-\log D_{\theta_D}(I^{HR})\right] + \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}\left[-\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$

SRGAN/2_SRGAN/SRGAN_utils.py

```
55      discriminator.train()
56      generator.train()
57      train_tqdm_loader = tqdm.tqdm(train_loader)
58      for i, (lr_img, hr_img) in enumerate(train_tqdm_loader):
59          lr_img = lr_img.to(device)
60          hr_img = hr_img.to(device)
61
62          # (1-1) train a discriminator model.
63          d_optimizer.zero_grad()
64          sr_img = generator(lr_img).detach()     # detach() <- for not back-propagation
65          d_i_hr = discriminator(hr_img).flatten()
66          d_i_sr = discriminator(sr_img).flatten()
67          d_loss_real = bce_loss_function(d_i_hr, torch.ones_like(d_i_hr))    # Goal: D[I^{HR}]    ~= 1 (for real image)
68          d_loss_fake = bce_loss_function(d_i_sr, torch.zeros_like(d_i_sr))   # Goal: D[G(I^{LR})] ~= 0 (for fake image)
69          d_loss = d_loss_real + d_loss_fake      # maximize      E[log[D[I^{HR}]]] + E[log[(1 - D[G(I^{LR})])]]
70                                                  # <=> maximize  E[log[D[I^{HR}]]] - E[log[D[G(I^{LR})]]]
71                                                  # for PyTorch implementation (PyTorch only has a minimizer!!!),
72                                                  # <=> minimize  - E[log[D[I^{HR}]]] + E[log[D[G(I^{LR})]]]
73          train_d_loss += d_loss.item()
74          d_loss.backward(retain_graph=True)
75          d_optimizer.step()
```

Binary Cross Entropy

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

if $y = 1$, $L = -\log \hat{y}$

otherwise ($y = 0$), $L = -\log(1 - \hat{y})$

Training Details for SRGAN

- Generator

  ⇔ SRResNet

  - Generator is initialized with **pretrained weights** of SRResNet (w/ MSE Loss).
  - Training objective **[Code]**
    - $\min_{\theta_G} \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$

      ⇔ $\min_{\theta_G} \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}\left[-\log(D_{\theta_D}(G_{\theta_G}(I^{LR})))\right]$

SRGAN/2_SRGAN/SRGAN_utils.py

```
77          # (1-2) train a generator model.
78          g_optimizer.zero_grad()
79          hr_img = hr_img.detach()                # detach() <- for not back-propagation
80          sr_img = generator(lr_img)
81          d_i_sr = d_i_sr.detach()
82          content_loss = content_loss_function(hr_img, sr_img)
83          adversarial_loss = bce_loss_function(d_i_sr, torch.ones_like(d_i_sr))  # minimize  E[log[(1 - D[G(I^{LR})])]]
84                                                                                 # <=> minimize  - E[log[D[G(I^{LR})]]]
85          g_loss = 1 * content_loss + 1e-3 * adversarial_loss
86
87          train_g_loss += g_loss.item()
88          g_loss.backward()
89          g_optimizer.step()
```

Binary Cross Entropy

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

if $y = 1$, $L = -\log \hat{y}$

otherwise ($y = 0$), $L = -\log(1 - \hat{y})$

SRGAN [Code]

SRGAN/2_SRGAN/SRGAN_train.py

```python
if __name__ == '__main__':
    # Parser
    my_parser = Parser(mode='train')
    my_args = my_parser.parser.parse_args()

    # Tag
    cur_time = datetime.datetime.now().strftime('%y-%m-%d-%H-%M-%S')
    tag_name = f'{my_args.tag}-{cur_time}' if my_args.tag else f'{cur_time}'
    print('[SRGAN]')
    print(f'{tag_name} experiment has been started.')

    # Parameter
    train_dir = my_args.train_dir
    valid_dir = my_args.valid_dir
    device = f'cuda:{my_args.gpu_index}' if my_args.gpu_index >= 0 else 'cpu'

    # Loader (Train / Valid)
    train_dataset = HR_to_HR_LR_PairDataset(train_dir, mode='train', crop_size=128)
    valid_dataset = HR_to_HR_LR_PairDataset(valid_dir, mode='eval', crop_size=256)
    train_loader = DataLoader(train_dataset, my_args.batch_size, shuffle=True, pin_memory=True)
    valid_loader = DataLoader(valid_dataset, my_args.batch_size, shuffle=False, pin_memory=True)

    discriminator = Discriminator(in_channels=3)
    discriminator_best_psnr_model_state = discriminator.state_dict()
    discriminator_best_ssim_model_state = discriminator.state_dict()
    d_optimizer = optim.Adam(discriminator.parameters(), lr=my_args.d_lr, betas=(0.9, 0.999))

    generator = Generator(in_channels=3, scaling_factor=4, num_residual_blocks=16)
    generator.load_state_dict(torch.load(my_args.generator_pt_path))
    generator_best_psnr_model_state = generator.state_dict()
    generator_best_ssim_model_state = generator.state_dict()
    g_optimizer = optim.Adam(generator.parameters(), lr=my_args.g_lr, betas=(0.9, 0.999))

    db_valid_PSNR_SSIM = {'psnr': [], 'ssim': []}
    tensorboard_writer = SummaryWriter(f'./log_dir/{tag_name}')
```

```python
    for epoch in range(my_args.epochs):
        valid_psnr, valid_ssim = \
            train_and_validate_SRGAN(generator, g_optimizer, discriminator, d_optimizer, epoch, device,
                                     train_loader, valid_loader, my_args.content_loss_function, tensorboard_writer)
        if db_valid_PSNR_SSIM['psnr'] and \
                valid_ssim > max(db_valid_PSNR_SSIM['psnr']):  # store the best model so far.
            generator_best_psnr_model_state = generator.state_dict()
            discriminator_best_psnr_model_state = discriminator.state_dict()
        if db_valid_PSNR_SSIM['ssim'] and \
                valid_ssim > max(db_valid_PSNR_SSIM['ssim']):  # store the best model so far.
            generator_best_ssim_model_state = generator.state_dict()
            discriminator_best_ssim_model_state = discriminator.state_dict()

        db_valid_PSNR_SSIM['psnr'].append(valid_psnr)
        db_valid_PSNR_SSIM['ssim'].append(valid_ssim)

    generator_last_model_state = generator.state_dict()
    discriminator_last_model_state = discriminator.state_dict()

    torch.save(generator_best_psnr_model_state, f'SRGAN-generator-best-psnr-{tag_name}.pt')
    torch.save(generator_best_ssim_model_state, f'SRGAN-generator-best-ssim-{tag_name}.pt')
    torch.save(generator_last_model_state, f'SRGAN-generator-last-{tag_name}.pt')
    torch.save(discriminator_best_psnr_model_state, f'SRGAN-discriminator-best-psnr-{tag_name}.pt')
    torch.save(discriminator_best_ssim_model_state, f'SRGAN-discriminator-best-ssim-{tag_name}.pt')
    torch.save(discriminator_last_model_state, f'SRGAN-discriminator-last-{tag_name}.pt')
    print(f'SRGAN-generator-best-psnr-{tag_name}.pt is stored.')
    print(f'SRGAN-generator-best-ssim-{tag_name}.pt is stored.')
    print(f'SRGAN-generator-last-{tag_name}.pt is stored.')
    print(f'SRGAN-discriminator-best-psnr-{tag_name}.pt is stored.')
    print(f'SRGAN-discriminator-best-ssim-{tag_name}.pt is stored.')
    print(f'SRGAN-discriminator-last-{tag_name}.pt is stored.')
    write_log_for_SRGAN(db_valid_PSNR_SSIM, f'SRGAN-{tag_name}-log.txt')
    print(f'SRGAN-{tag_name}-log.txt is stored.')
    plot_log_for_SRGAN(db_valid_PSNR_SSIM, f'SRGAN-{tag_name}-log.png')
    print(f'SRGAN-{tag_name}-log.png is stored.')
```

Evaluation Metric (1)

- PSNR (Peak Signal-to-Noise Ratio)
  - PSNR is formulated as follows:

$$PSNR(I_a, I_b) = 10 \log \frac{s^2}{MSE(I_a, I_b)}.$$

This work sets s to 1.
($\because$ Normalized image tensors are used.)

  - I_a and I_b denote the pair of images of same dimension.
    - The dimension of the image can be represented as (C, H, W).
      - C: Number of **C**hannels
      - H: **H**eight
      - W: **W**idth
  - s denotes the maximum possible pixel value of the image.
  - $MSE(I_a, I_b) = \sum_{i=1}^{C} \sum_{j=1}^{H} \sum_{k=1}^{W} |I_a(i,j,k) - I_b(i,j,k)|^2$.

Reference:
[8] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
[9] https://cvnote.ddlee.cc/2019/09/12/psnr-ssim-python

Evaluation Metric (2)

- SSIM (Structural Similarity Index Map)
  - SSIM is formulated as follows:

$$SSIM(I_a, I_b) = \left[l(I_a, I_b)^{\alpha} \cdot c(I_a, I_b)^{\beta} \cdot s(I_a, I_b)^{\gamma} \right].$$

  - I_a and I_b denote the pair of images of same dimension.
    - The dimension of the image can be represented as (C, H, W).
      - C: Number of **C**hannels
      - H: **H**eight
      - W: **W**idth
  - Luminance, $l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}$
    - μ_i: Average of i.
  - Contrast, $c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$
    - σ_i^2: Variance of i.
  - Structure, $s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}$
    - σ_{xy}: Covariance of x and y
  - In addition to above definitions, this work sets c_3 to $\frac{c_2}{2}$ and α, β, γ to 1,
    - $SSIM(I_a, I_b) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$. → In this work, c_1 and c_2 are set to 0.01^2 and 0.03^2, respectively as in [4].

Reference:
[9] https://cvnote.ddlee.cc/2019/09/12/psnr-ssim-python
[10] https://en.wikipedia.org/wiki/Structural_similarity

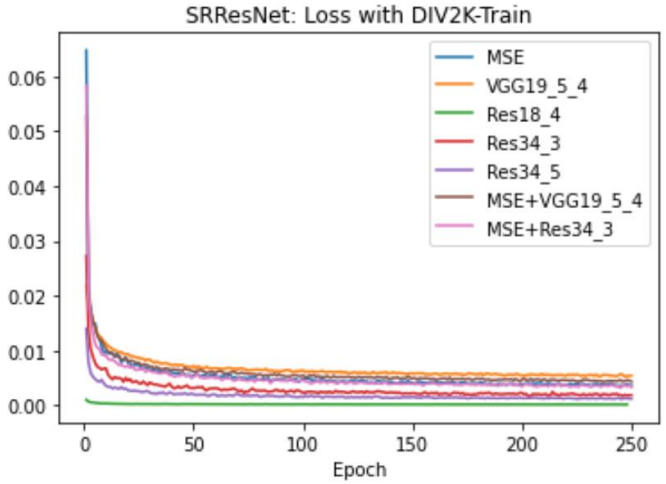
Experimental Setup

- OS / CPU: Linux Ubuntu 20.04 (LTS) 64bit, AMD Ryzen 5 3700x
- GPU: Nvidia RTX 3090 (24GB)
- CUDA: v11.2

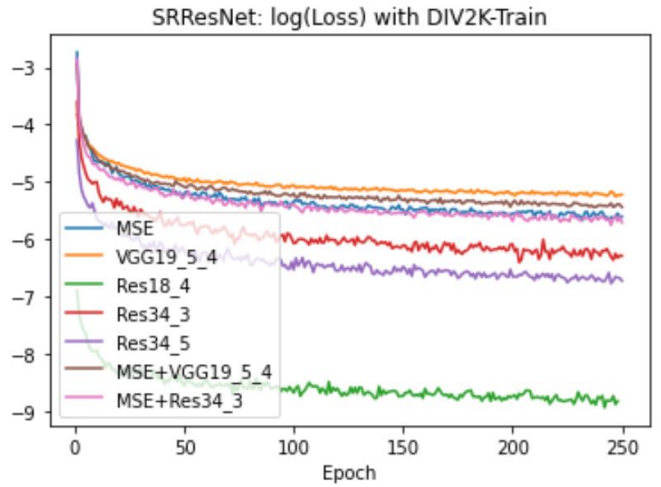
- Python 3.7
  - PyTorch: Torch 1.10.0, Torchvision 0.11.1
  - Glob2
  - Tqdm
  - Tensorboard
  - Matplotlib
  - Pytorch-msssim
    - For **SSIM** → Rather than implementing SSIM myself, I used Jorge-pessoa's code.
    - Link: https://github.com/jorge-pessoa/pytorch-msssim
    - We can simply install and use Pytorch-msssim via pip installer (**pip install pytorch-msssim**).

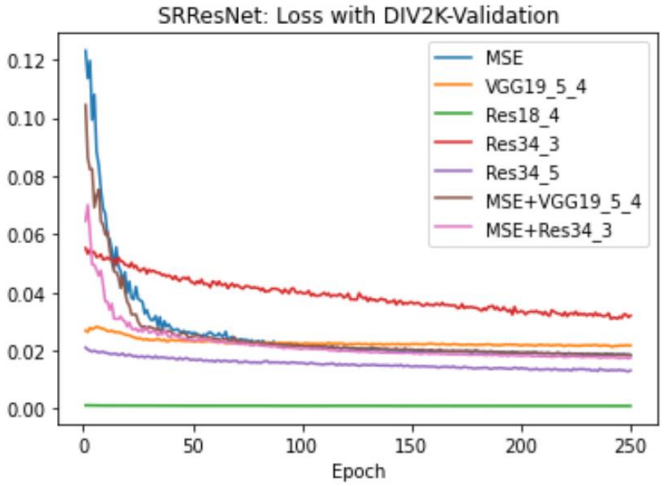
Experimental Results (1) SRResNet

- Loss per each epoch

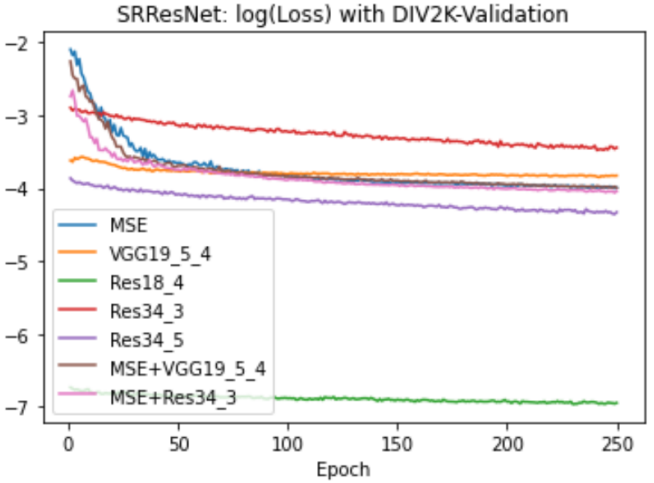


log(·)



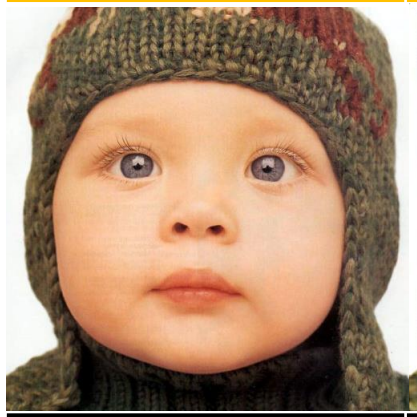
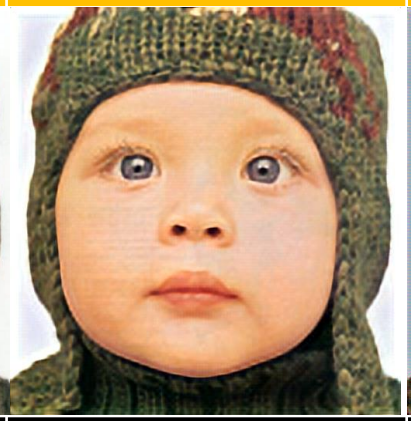
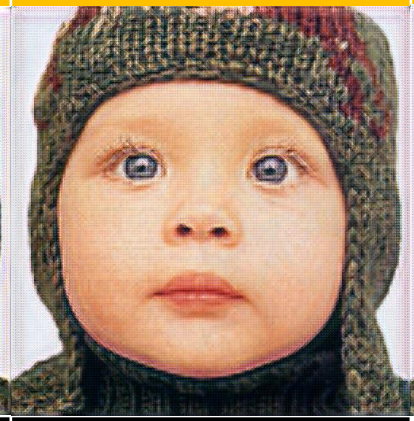
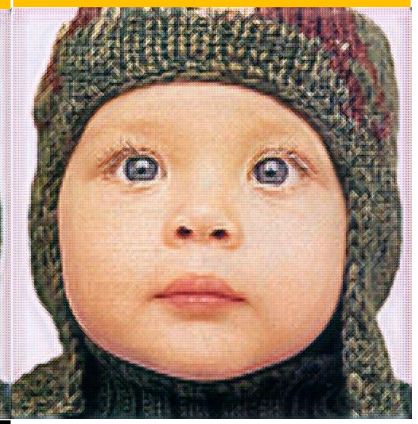


log(·)



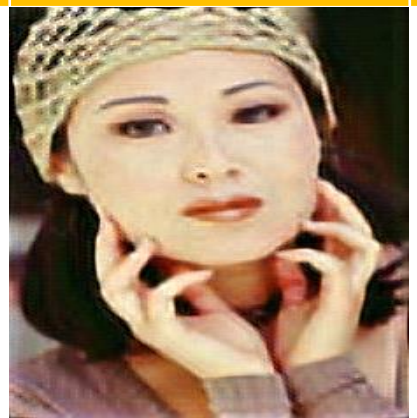
Experimental Results (1) SRResNet

- Visual Results on Set5 - Baby

GT	SRResNet-				
Set5/GTmod12/baby.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.7 * MSE + 0.3 * VGG19_5_4	0.7 * MSE + 0.3 * Res18_4	0.7 * MSE + 0.3 * Res34_3	0.7 * MSE + 0.3 * Res34_5	Set5/LRbicx4/baby.png	
	[redacted]		[redacted]		

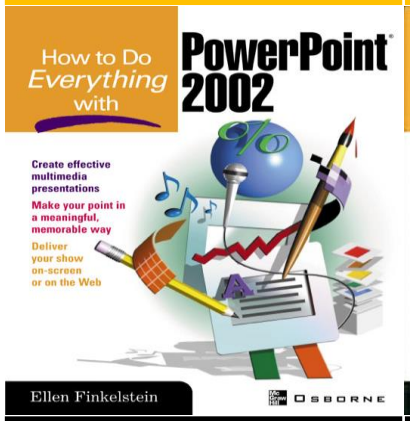
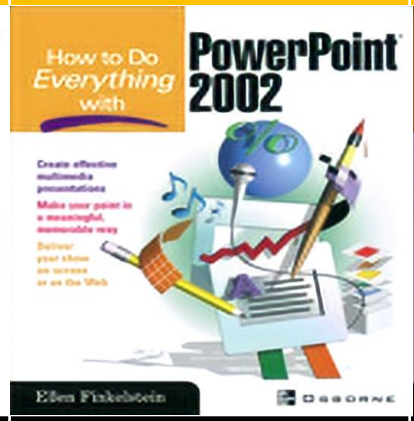
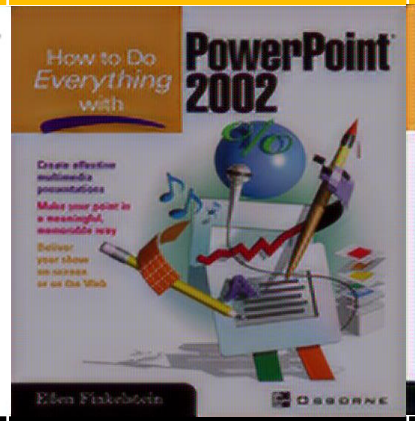
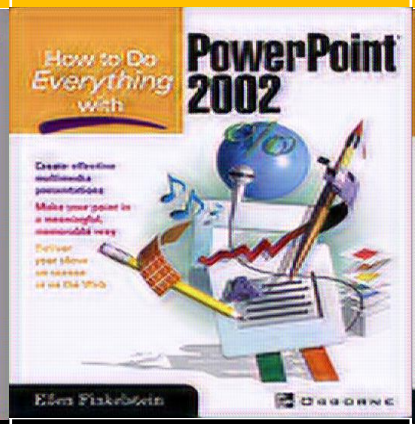
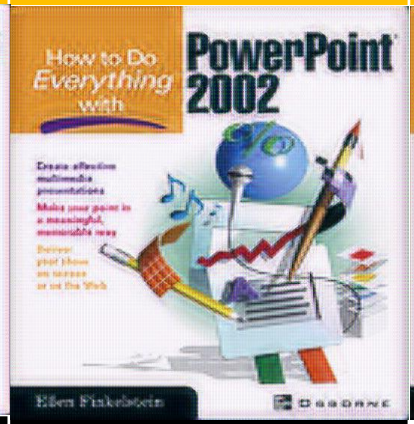
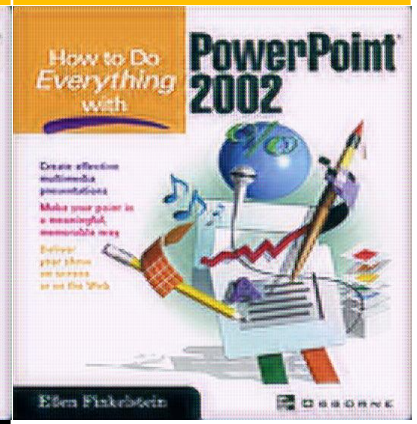
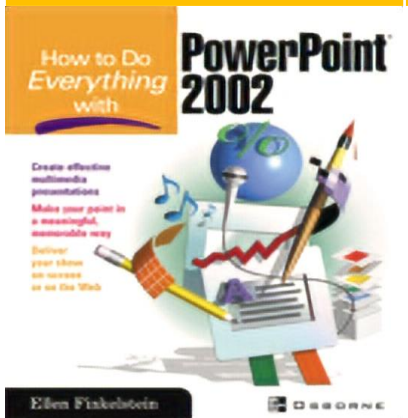
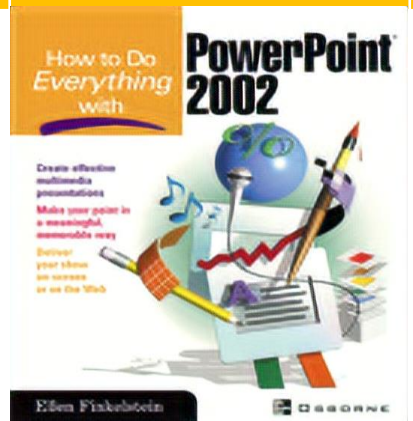
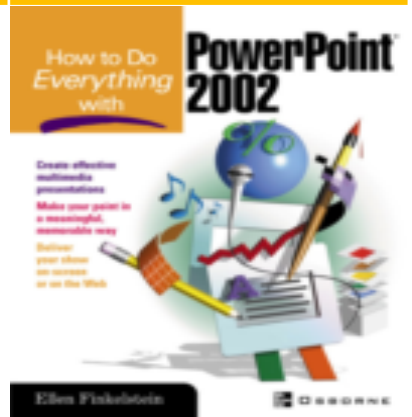
Experimental Results (1) SRResNet

- Visual Results on Set5 - Woman

GT	SRResNet-				
Set5/GTmod12/woman.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.7 * MSE + 0.3 * VGG19_5_4	0.7 * MSE + 0.3 * Res18_4	0.7 * MSE + 0.3 * Res34_3	0.7 * MSE + 0.3 * Res34_5	Set5/LRbicx4/woman.png	
	[redacted]		[redacted]		

Experimental Results (1) SRResNet

- Visual Results on Set14 - Ppt3

GT	SRResNet-				
Set14/GTmod12/ppt3.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.7 * MSE + 0.3 * VGG19_5_4	0.7 * MSE + 0.3 * Res18_4	0.7 * MSE + 0.3 * Res34_3	0.7 * MSE + 0.3 * Res34_5	Set5/LRbicx4/ppt3.png	
	-		-		

Experimental Results (1) SRResNet

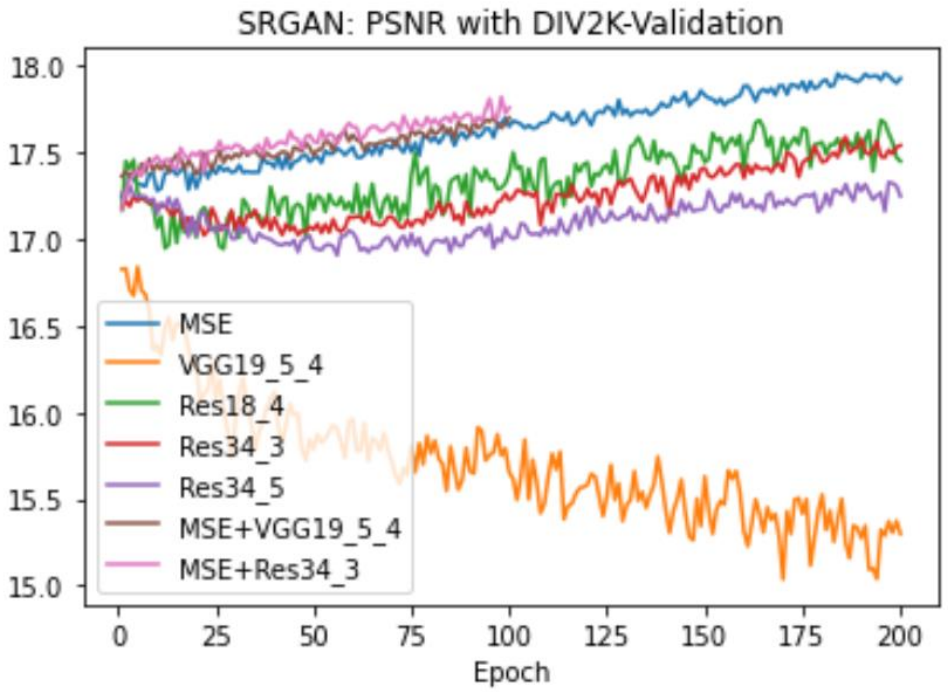
- PSNR / SSIM on Set5 and Set14
  - Mean PSNR and Mean SSIM

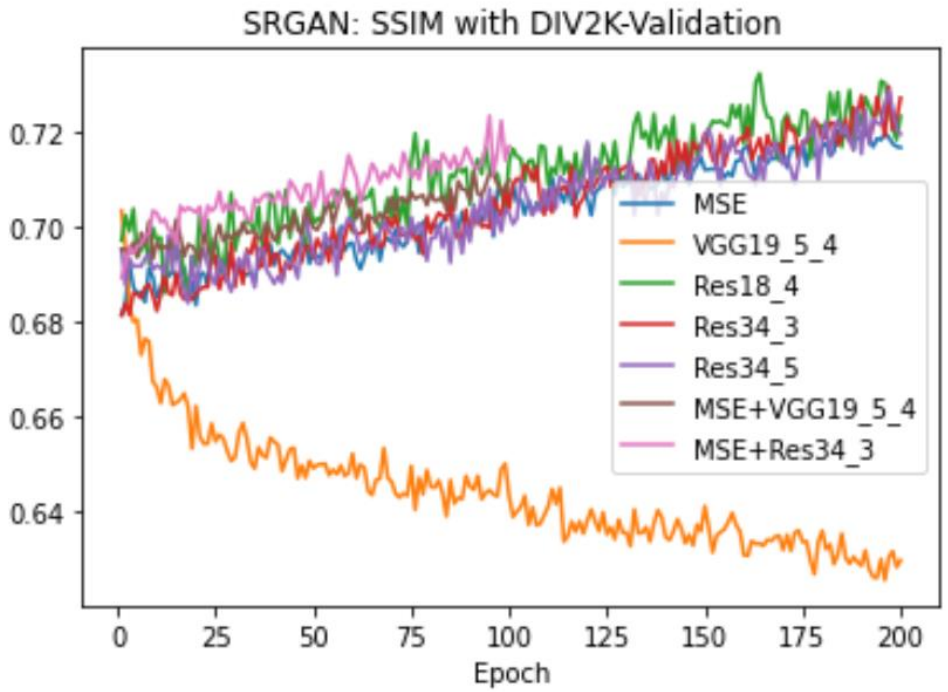
	SRResNet-								
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.7 * MSE + 0.3 * VGG19_5_4**	**0.7 * MSE + 0.3 * Res18_4**	**0.7 * MSE + 0.3 * Res34_3**	**0.7 * MSE + 0.3 * Res34_5**
PSNR	23.13	14.68	16.79	14.90	14.58	23.42	-	23.34	-
SSIM	0.9348	0.8186	0.8017	0.8287	0.8184	0.9379	-	0.9413	-

	SRResNet-								
Set14	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.7 * MSE + 0.3 * VGG19_5_4**	**0.7 * MSE + 0.3 * Res18_4**	**0.7 * MSE + 0.3 * Res34_3**	**0.7 * MSE + 0.3 * Res34_5**
PSNR	21.73	14.00	16.29	14.67	14.37	22.16	-	21.99	-
SSIM	0.9090	0.8064	0.7812	0.8104	0.7989	0.9112	-	0.9164	-

Experimental Results (2) SRGAN

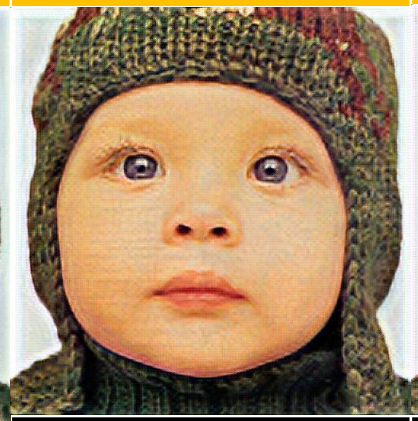
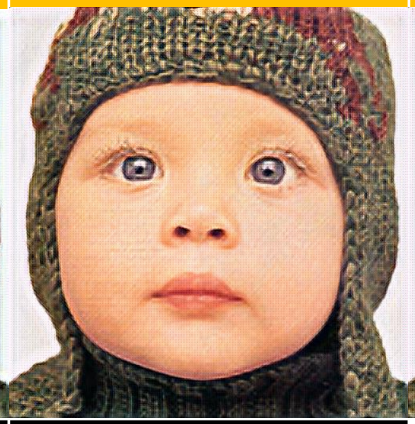
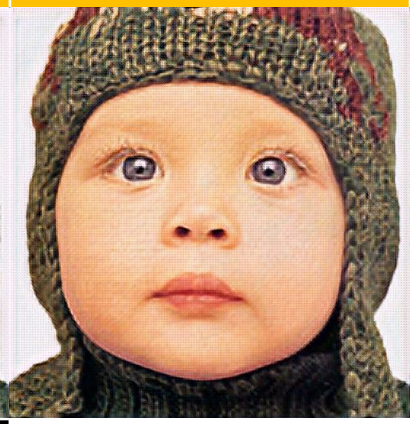
- Validation PSNR / SSIM per each epoch





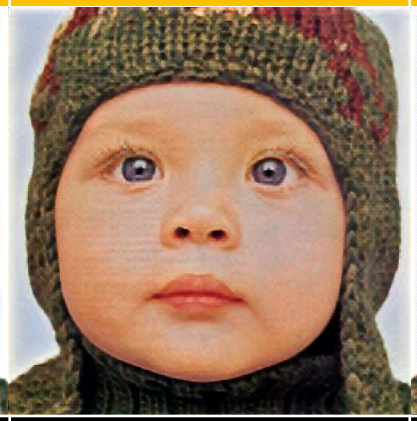
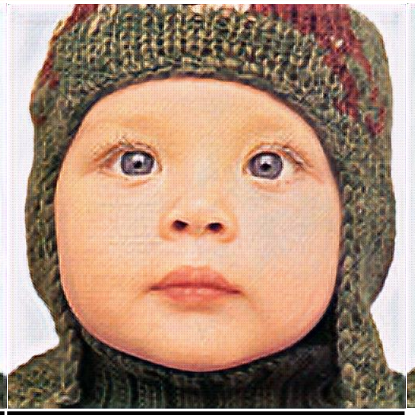
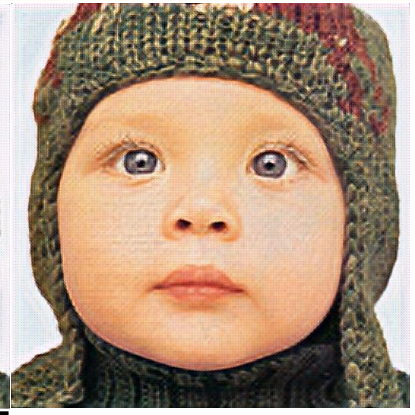
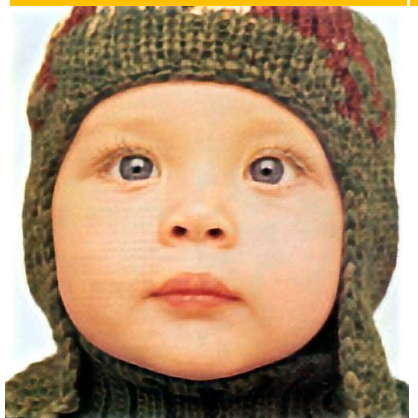
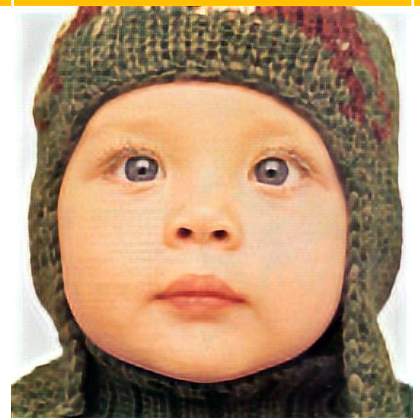
Experimental Results (2) SRGAN

- Visual Results on Set5 - Baby

GT	SRGAN-	(200 epochs)			
Set5/GTmod12/baby.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/baby.png	
[redacted]	[redacted]	[redacted]	[redacted]		

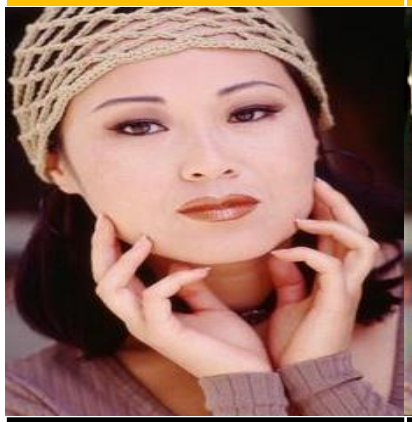
Experimental Results (2) SRGAN

- Visual Results on Set5 - Baby

GT	SRGAN-	(100 epochs)			
Set5/GTmod12/baby.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/baby.png	
	[redacted]		[redacted]		

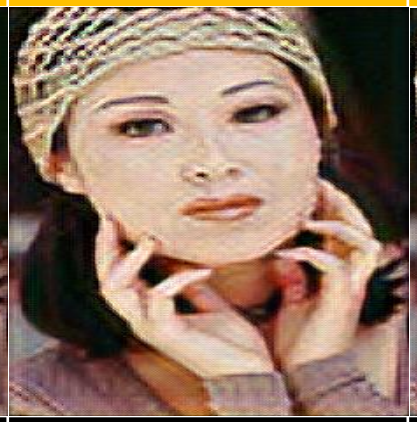
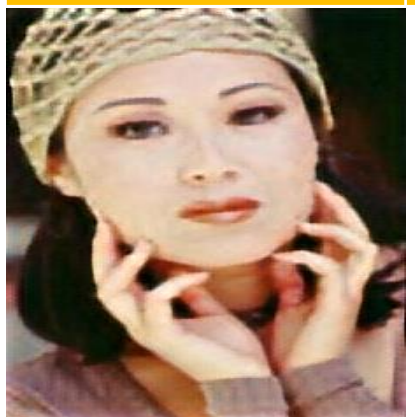
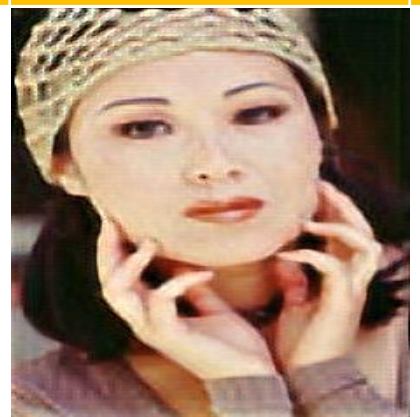
Experimental Results (2) SRGAN

- Visual Results on Set5 - Woman

GT	SRGAN-	(200 epochs)			
Set5/GTmod12/woman.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/woman.png	
[redacted]	[redacted]	[redacted]	[redacted]		

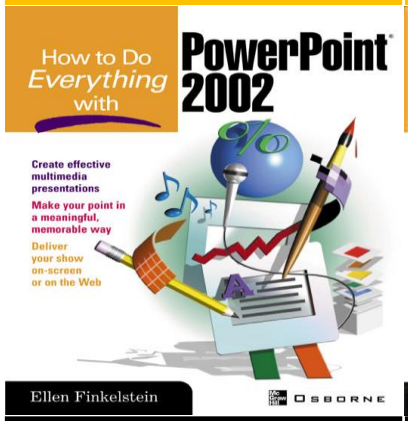
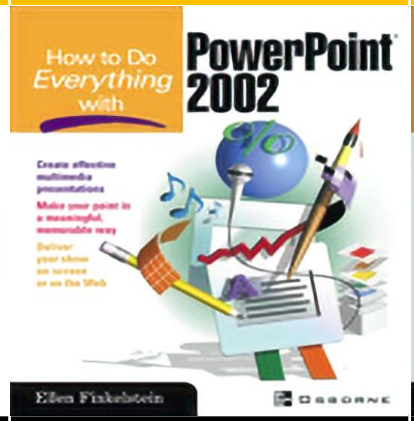
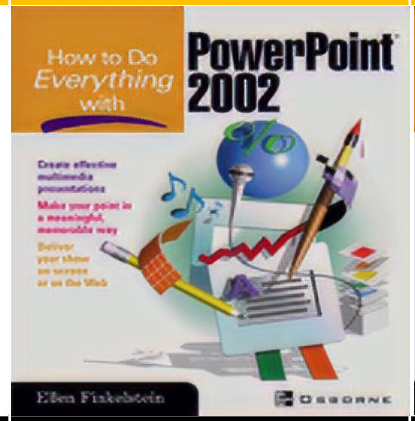
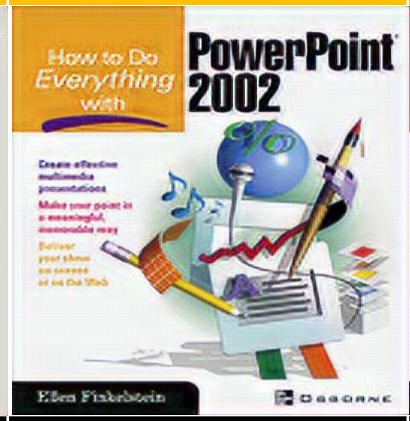
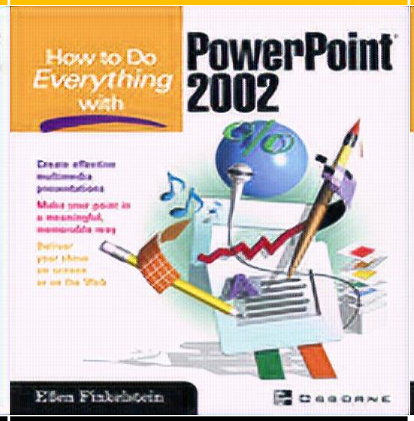
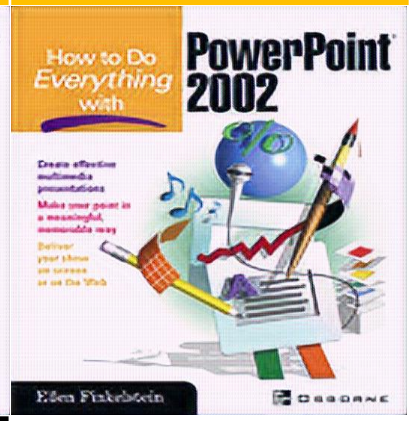
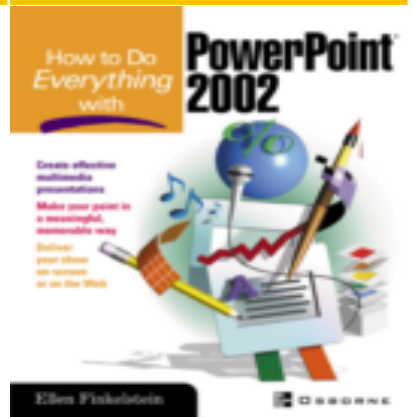
Experimental Results (2) SRGAN

- Visual Results on Set5 - Woman

GT	SRGAN-	(100 epochs)			
Set5/GTmod12/woman.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/woman.png	
	-		-		

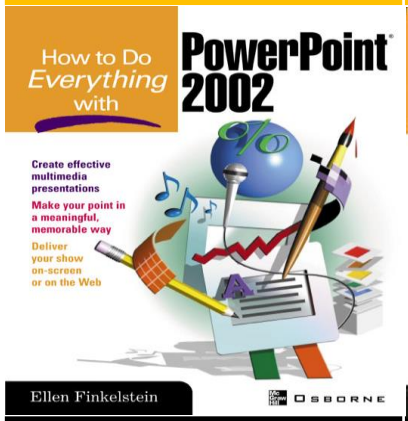
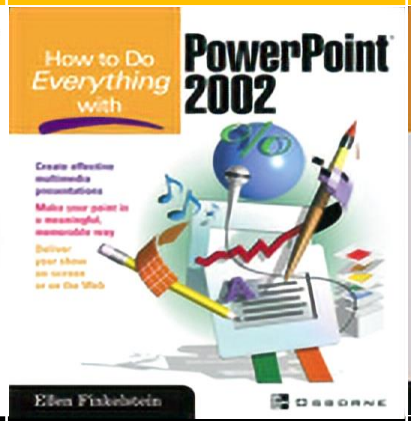
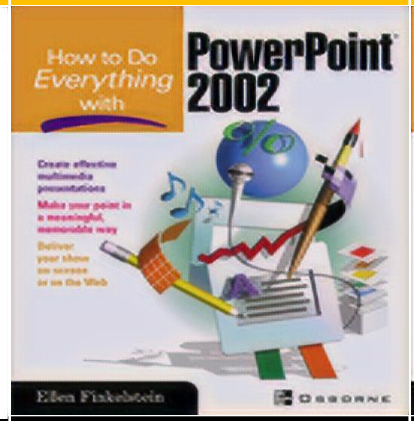
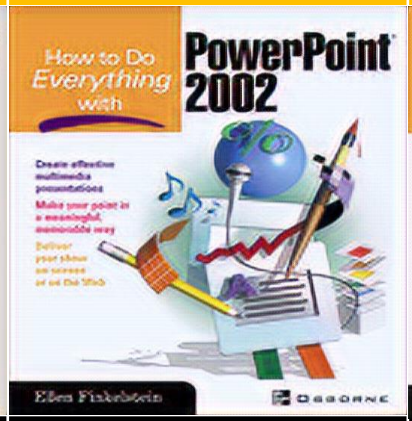
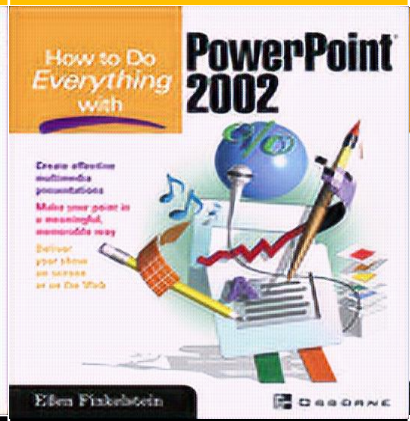
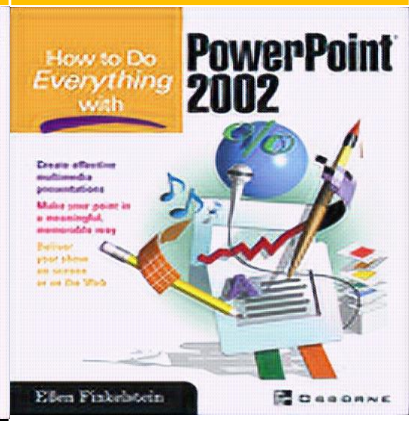
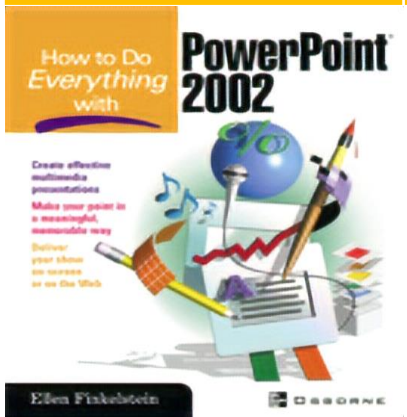
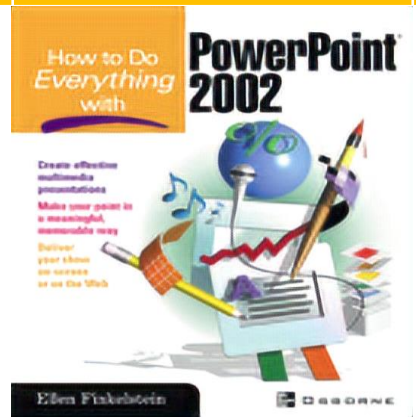
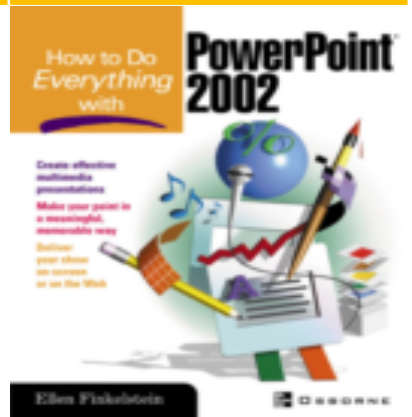
Experimental Results (2) SRGAN

- Visual Results on Set14 - Ppt3

GT	SRGAN-	(200 epochs)			
Set14/GTmod12/ppt3.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/ppt3.png	
[redacted]	[redacted]	[redacted]	[redacted]		

Experimental Results (2) SRGAN

- Visual Results on Set14 - Ppt3

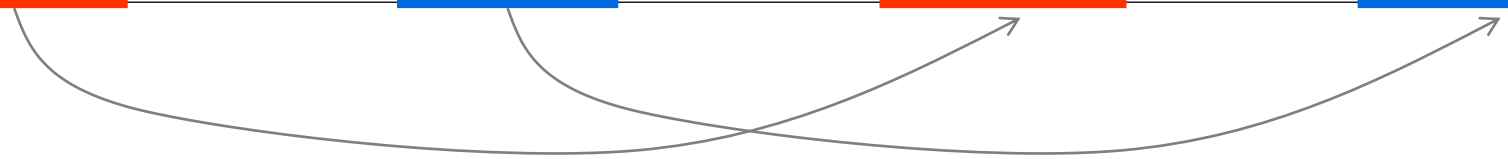
GT	SRGAN-	(100 epochs)			
Set14/GTmod12/ppt3.png	MSE	VGG19_5_4	Res18_4	Res34_3	Res34_5
					
				Bicubic	
0.6 * MSE + 0.4 * VGG19_5_4	0.6 * MSE + 0.4 * Res18_4	0.6 * MSE + 0.4 * Res34_3	0.6 * MSE + 0.4 * Res34_5	Set5/LRbicx4/ppt3.png	
					

Experimental Results (2) SRGAN

- PSNR / SSIM on Set5 and Set14
  - Mean PSNR and Mean SSIM

	SRGAN-	200 epochs							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	24.17	20.12	20.21	18.88	18.75	-	-	-	-
SSIM	0.9470	0.8926	0.8917	0.8754	0.8706	-	-	-	-

	SRGAN-	100 epochs							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	23.07	20.96	20.44	19.62	19.24	23.48	-	23.29	-
SSIM	0.9364	0.9073	0.8717	0.8891	0.8769	0.9426	-	0.9375	-

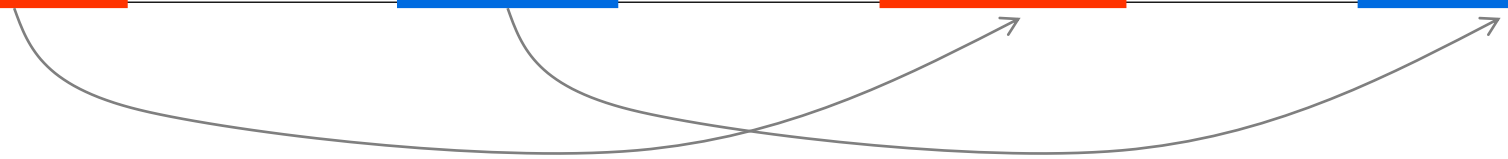


Experimental Results (2) SRGAN

- PSNR / SSIM on Set5 and Set14
  - Mean PSNR and Mean SSIM

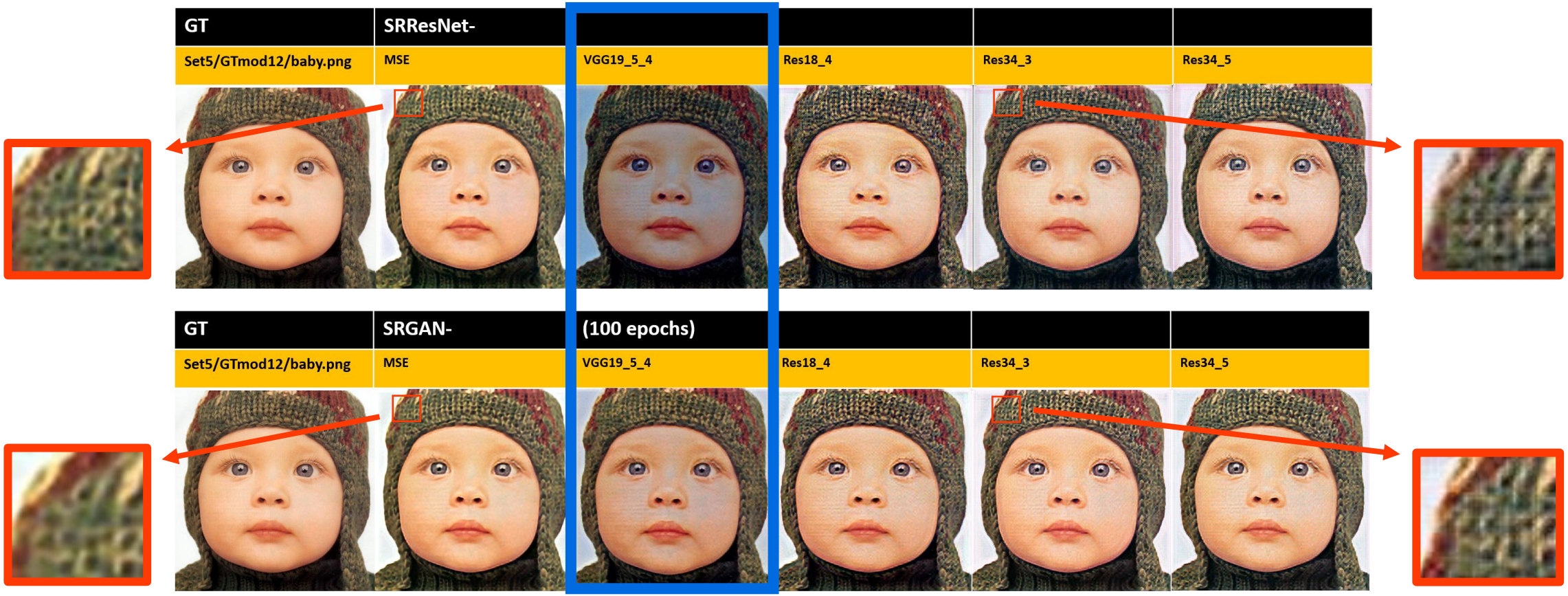
	SRGAN-	200 epochs							
Set14	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	22.50	19.40	19.29	18.03	18.02	-	-	-	-
SSIM	0.9203	0.8783	0.8462	0.8465	0.8449	-	-	-	-

	SRGAN-	100 epochs							
Set14	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	21.73	20.33	19.59	18.88	18.42	22.21	-	21.96	-
SSIM	0.9106	0.8920	0.8485	0.8561	0.8494	0.9197	-	0.9120	-



Conclusion

A. **Finer texture details** can be recovered with **VGG-based and ResNet-based content loss function**.



[Note] 1) Content loss function has more impact on the super resolution task than GAN!

2) Of course, GAN also boosts the performance of super resolution (See the 'blue' column ☐.)!

Conclusion

B. **Generative Adversarial Network** boosts the super resolution performance in terms of **PSNR & SSIM** in this work.

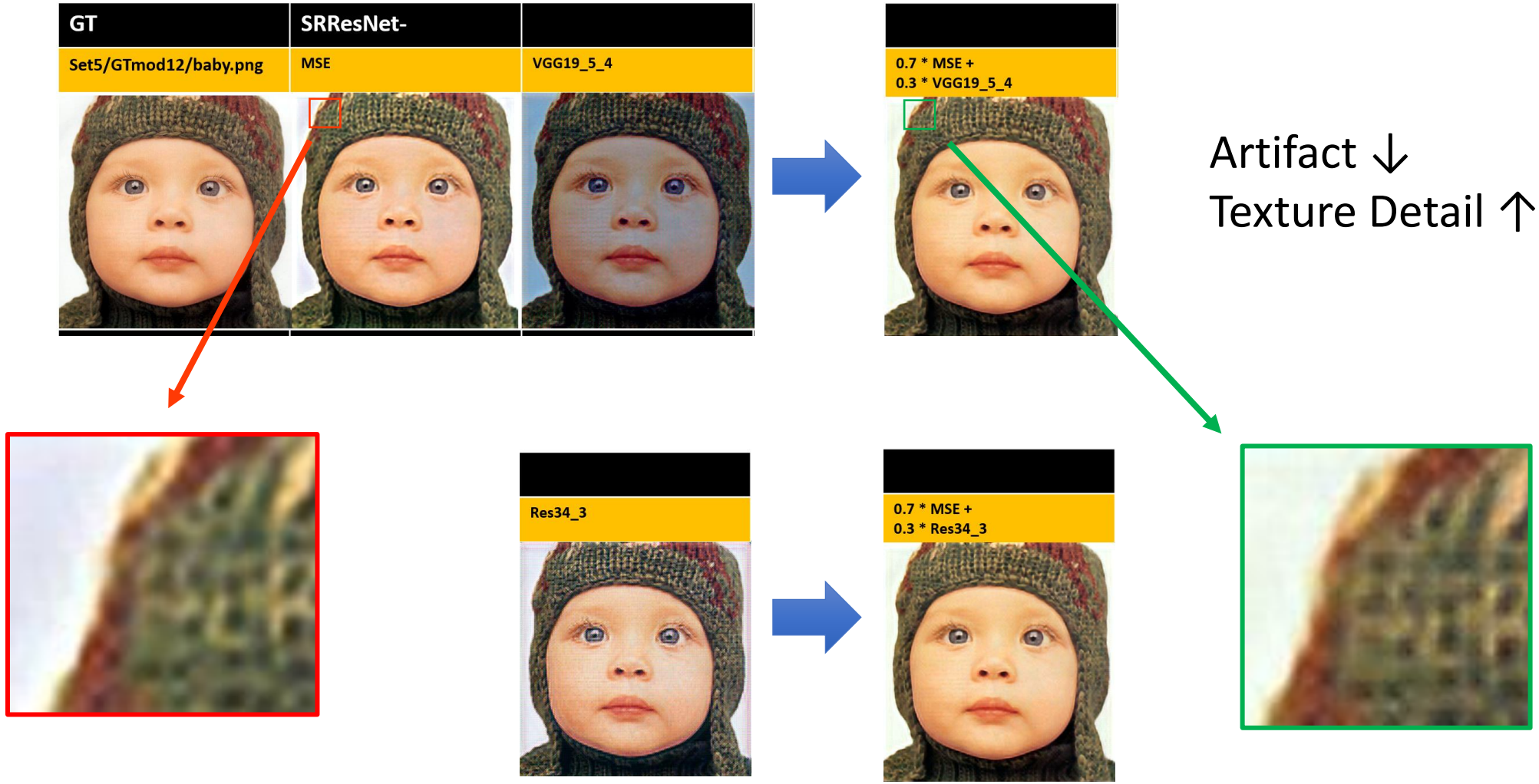
	SRResNet-								
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.7 * MSE + 0.3 * VGG19_5_4**	**0.7 * MSE + 0.3 * Res18_4**	**0.7 * MSE + 0.3 * Res34_3**	**0.7 * MSE + 0.3 * Res34_5**
PSNR	23.13	14.68	16.79	14.90	14.58	23.42	-	23.34	-
SSIM	0.9348	0.8186	0.8017	0.8287	0.8184	0.9379	-	0.9413	-

	SRGAN-	200 epochs							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	24.17	20.12	20.21	18.88	18.75	-	-	-	-
SSIM	0.9470	0.8926	0.8917	0.8754	0.8706	-	-	-	-

	SRGAN-	100 epochs							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	23.07	20.96	20.44	19.62	19.24	23.48	-	23.29	-
SSIM	0.9364	0.9073	0.8717	0.8891	0.8769	0.9426	-	0.9375	-

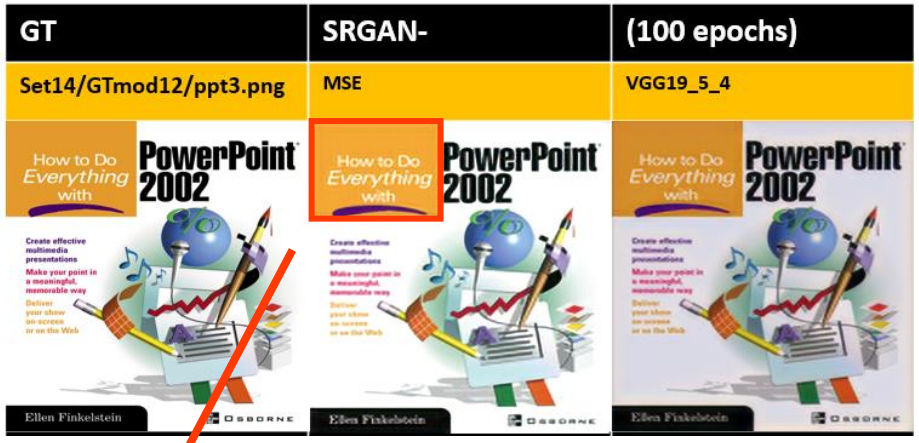
Conclusion

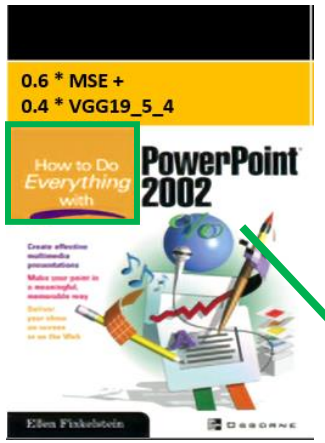
C. **Hybrid of 'Mean Squared Error Loss' and 'Content Loss'** improves the quality of super resolution results.



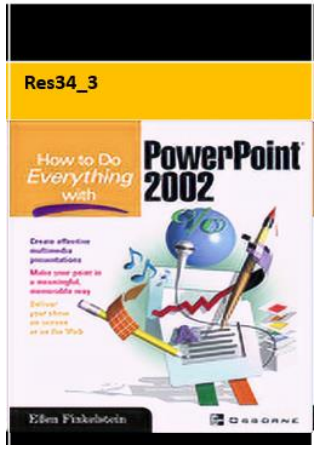
Conclusion

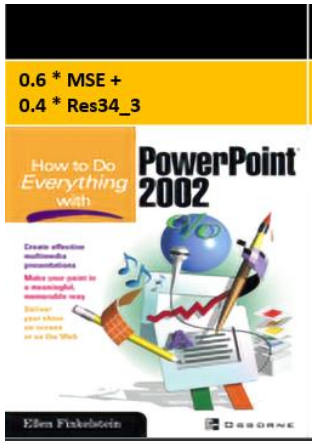
C. **Hybrid of 'Mean Squared Error Loss' and 'Content Loss'** improves the quality of super resolution results.





Noise ↓





Conclusion

C. **Hybrid of 'Mean Squared Error Loss' and 'Content Loss'** improves the quality of super resolution results.

	SRResNet-							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.7 * MSE + 0.3 * VGG19_5_4**	**0.7 * MSE + 0.3 * Res18_4**	**0.7 * MSE + 0.3 * Res34_3**	**0.7 * MSE + 0.3 * Res34_5**
PSNR	23.13	14.68	16.79	14.90	14.58	23.42	-	23.34	-
SSIM	0.9348	0.8186	0.8017	0.8287	0.8184	0.9379	-	0.9413	-

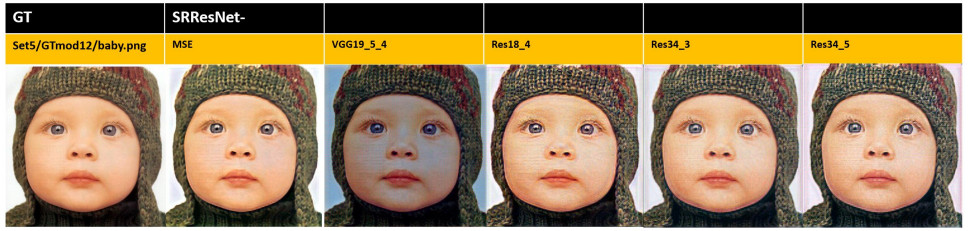
	SRResNet-							
Set14	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.7 * MSE + 0.3 * VGG19_5_4**	**0.7 * MSE + 0.3 * Res18_4**	**0.7 * MSE + 0.3 * Res34_3**	**0.7 * MSE + 0.3 * Res34_5**
PSNR	21.73	14.00	16.29	14.67	14.37	22.16	-	21.99	-
SSIM	0.9090	0.8064	0.7812	0.8104	0.7989	0.9112	-	0.9164	-

	SRGAN-	100 epochs							
Set5	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	23.07	20.96	20.44	19.62	19.24	23.48	-	23.29	-
SSIM	0.9364	0.9073	0.8717	0.8891	0.8769	0.9426	-	0.9375	-

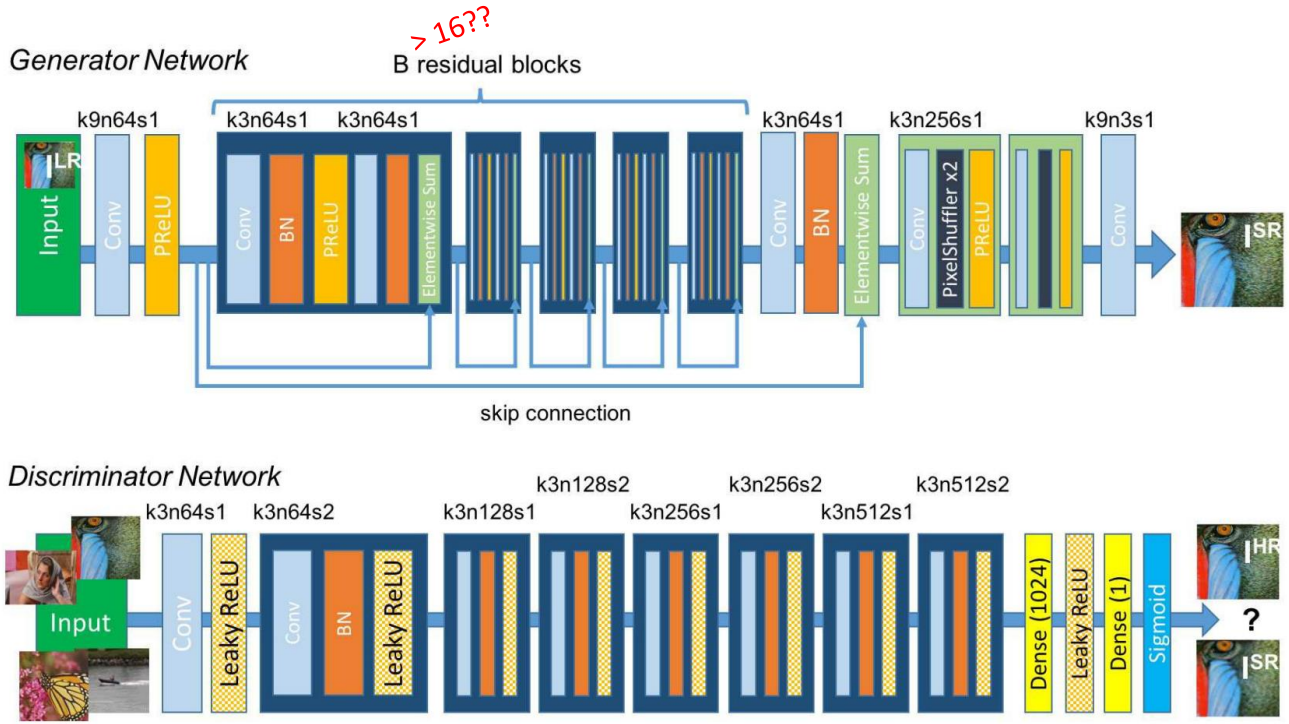
	SRGAN-	100 epochs							
Set14	**MSE**	**VGG19_5_4**	**Res18_4**	**Res34_3**	**Res34_5**	**0.6 * MSE + 0.4 * VGG19_5_4**	**0.6 * MSE + 0.4 * Res18_4**	**0.6 * MSE + 0.4 * Res34_3**	**0.6 * MSE + 0.4 * Res34_5**
PSNR	21.73	20.33	19.59	18.88	18.42	22.21	-	21.96	-
SSIM	0.9106	0.8920	0.8485	0.8561	0.8494	0.9197	-	0.9120	-

Both PSNR and SSIM are also improved!!!

Discussion



- In the future work,
  - We will investigate why VGG content loss results in color-desaturation of super-resolution results, compared to ResNet-based loss.
  - We will design more various content loss with other CNNs.
  - We will increase the number (B) of residual blocks in the generator in SRGAN.
    - B > 16



References

- [1] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
- [2] https://kh-kim.gitbook.io/natural-language-processing-with-pytorch/00-cover-11/01-intro
- [3] https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network
- [4] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- [5] https://paperswithcode.com/dataset/div2k
- [6] https://paperswithcode.com/dataset/set5
- [7] https://paperswithcode.com/dataset/set11
- [8] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
- [9] https://cvnote.ddlee.cc/2019/09/12/psnr-ssim-python
- [10] https://en.wikipedia.org/wiki/Structural_similarity