In [1]:
```
#Importing the required libraries
import torch
from torch.utils.data import *
import torch.nn as nn
import torchvision
import glob
import os
import torchvision.transforms as transforms
import torch.nn.functional as F
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from torchsummary import summary
import seaborn as sn
from torch.utils.data import Dataset
import glob
import os
from PIL import Image
from torchvision.transforms import ToTensor
from sklearn.metrics import confusion_matrix
import torch.optim as optim
from torchvision.models import vgg19
from sklearn.model_selection import KFold
from pathlib import Path
```

In [ ]:
```
# !rm -rf DIV2K_train_HR/
# !rm -rf DIV2K_train_LR_bicubic/
# !rm -rf DIV2K_valid_HR/
# !rm -rf DIV2K_valid_LR_bicubic/
# !rm -rf sr_img/
```

In [2]:
```
#Downloading the Dataset
!wget "http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip"
!wget "http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip"
!wget "http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_LR_bicubic_X2.zip"
!wget "http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_LR_bicubic_X2.zip"
```

```
--2022-05-12 00:31:18--  http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip
Resolving data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)... 129.132.52.178, 2001:67c:10
ec:36c2::178
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:80... conn
ected.
HTTP request sent, awaiting response... 302 Found
Location: https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip [following]
--2022-05-12 00:31:18--  https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 448993893 (428M) [application/zip]
Saving to: ‘DIV2K_valid_HR.zip’

DIV2K_valid_HR.zip  100%[===================>] 428.19M  17.2MB/s    in 31s

2022-05-12 00:31:49 (14.0 MB/s) - ‘DIV2K_valid_HR.zip’ saved [448993893/448993893]
```

```
--2022-05-12 00:31:50--  http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip
Resolving data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)... 129.132.52.178, 2001:67c:10
ec:36c2::178
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:80... conn
ected.
HTTP request sent, awaiting response... 302 Found
Location: https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip [following]
--2022-05-12 00:31:50--  https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 3530603713 (3.3G) [application/zip]
Saving to: ‘DIV2K_train_HR.zip’

DIV2K_train_HR.zip  100%[===================>]   3.29G  10.9MB/s    in 4m 6s

2022-05-12 00:35:57 (13.7 MB/s) - ‘DIV2K_train_HR.zip’ saved [3530603713/3530603713]

--2022-05-12 00:35:57--  http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_LR_bicubic_
X2.zip
Resolving data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)... 129.132.52.178, 2001:67c:10
ec:36c2::178
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:80... conn
ected.
HTTP request sent, awaiting response... 302 Found
Location: https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_LR_bicubic_X2.zip [follow
ing]
--2022-05-12 00:35:57--  https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_LR_bicubic
_X2.zip
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 925390592 (883M) [application/zip]
Saving to: ‘DIV2K_train_LR_bicubic_X2.zip’

DIV2K_train_LR_bicu 100%[===================>] 882.52M  19.1MB/s    in 48s

2022-05-12 00:36:45 (18.5 MB/s) - ‘DIV2K_train_LR_bicubic_X2.zip’ saved [925390592/92539
0592]

--2022-05-12 00:36:45--  http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_LR_bicubic_
X2.zip
Resolving data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)... 129.132.52.178, 2001:67c:10
ec:36c2::178
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:80... conn
ected.
HTTP request sent, awaiting response... 302 Found
Location: https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_LR_bicubic_X2.zip [follow
ing]
--2022-05-12 00:36:46--  https://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_LR_bicubic
_X2.zip
Connecting to data.vision.ee.ethz.ch (data.vision.ee.ethz.ch)|129.132.52.178|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 117763600 (112M) [application/zip]
Saving to: ‘DIV2K_valid_LR_bicubic_X2.zip’

DIV2K_valid_LR_bicu 100%[===================>] 112.31M  15.4MB/s    in 9.0s

2022-05-12 00:36:56 (12.5 MB/s) - ‘DIV2K_valid_LR_bicubic_X2.zip’ saved [117763600/11776
```

```
3600]
```

In [ ]:
```
#Unzipping the datafiles downloaded
!unzip DIV2K_train_LR_bicubic_X2.zip
!unzip DIV2K_valid_HR
!unzip DIV2K_valid_LR_bicubic_X2.zip
!unzip DIV2K_train_HR.zip
```

In [ ]:
```
TRAIN_HR_DIR = "./DIV2K_train_HR"
TRAIN_LR_DIR = "./DIV2K_train_LR_bicubic/X2"
TEST_HR_DIR = "./DIV2K_valid_HR"
TEST_LR_DIR = "./DIV2K_valid_LR_bicubic/X2"
```

In [ ]:
```
#Defining the model parameters
LR_CROPPED_SIZE = 100
UPSCALE = 2
HR_CROPPED_SIZE = UPSCALE * LR_CROPPED_SIZE

REAL_VALUE = 0.99
FAKE_VALUE = 0.0

BATCH_SIZE = 16
EPOCHS = 40

N_RESBLK_G = 20
LR = 0.0001
BETAS = (0.9,0.9999)

VGG_LOSS_COEF = 0.006
ADVERSARIAL_LOSS_COEF = 0.001
```

In [ ]:
```
#Defining a class
class DIV2K(Dataset):
        def __init__(self, data_dir, transform=transforms.ToTensor()):
                # Get all paths of images inside `data_dir` into a list
                pattern = os.path.join(data_dir, "**/*.png")
                self.file_paths = sorted(glob.glob(pattern, recursive=True))
                self.transform = transform

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

        def __getitem__(self, index):
                file_name = self.file_paths[index].split('/')[-1]
                img = Image.open(self.file_paths[index])
                img = self.transform(img)
                return img, file_name
```

In [ ]:
```
#Defining a Generator Class
class Generator(nn.Module):
        def __init__(self, n_res_blks, upscale_factor=2):
                super(Generator, self).__init__()
```

```
                #Convolutional layer
                self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=9, p
                self.prelu1 = nn.PReLU()

                #Residual Block
                self.res_blocks = nn.Sequential()
                for i in range(n_res_blks):
                        self.res_blocks.add_module(f"res_blk_{i}",
                                                                        Residual_Blo
                self.conv2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
                self.bn = nn.BatchNorm2d(64)

                #Pixel Shuffler
                self.pixel_shufflers = nn.Sequential()
                for i in range(1):
                        self.pixel_shufflers.add_module(f"pixel_shuffle_blk_{i}",

                self.conv3 = nn.Conv2d(in_channels=64, out_channels=3, kernel_size=9, p

        #Defining how the generator model will be run
        def forward(self, X):
                X = self.prelu1(self.conv1(X))
                X_before_resblks = X.clone()

                X = self.res_blocks(X)
                X = self.bn(self.conv2(X))
                #Skip Connections
                X = F.relu(X + X_before_resblks)
                X = self.pixel_shufflers(X)

                X = self.conv3(X)

                return F.tanh(X)

#Residual Block Class
class Residual_Block(nn.Module):
        def __init__(self, in_channels, out_channels, strides, use_1x1_conv=True):
                super(Residual_Block, self).__init__()

                self.use_1x1_conv = use_1x1_conv
                self.conv1x1 = nn.Conv2d(in_channels, out_channels, kernel_size=1, stri
                self.blk = nn.Sequential(
                        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stri
                        nn.BatchNorm2d(out_channels),
                        nn.PReLU(),
                        nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
                        nn.BatchNorm2d(out_channels),
                )

        #Defninig how the Residual block will be run
        def forward(self, X):
                """
                :param X: tensor with shape (N, C, H, W)
                """
                X_original = X.clone()
                X = self.blk(X)
                if self.use_1x1_conv:
                        X_original = self.conv1x1(X_original)

                return F.relu(X + X_original)
```

```
#Pixel Shuffler class
class PixelShufflerBlock(nn.Module):
        def __init__(self, in_channels, upscale_factor=2):
                super(PixelShufflerBlock, self).__init__()

                self.blk = nn.Sequential(
                        nn.Conv2d(in_channels=in_channels, out_channels=256, kernel_siz
                        nn.PixelShuffle(upscale_factor=upscale_factor),
                        nn.PReLU()
                )

        def forward(self, X):
                return self.blk(X)
```

In [ ]:

```
#Defining a Discriminator Class
class Discriminator(nn.Module):
        def __init__(self):
                super(Discriminator, self).__init__()

                self.conv1 = nn.Sequential(
                        nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, paddin
                        nn.LeakyReLU(negative_slope=0.2)
                )
                #Convolutional Blocks
                self.conv_blks = nn.Sequential(
                        ConvBlock(64, 64, 2),
                        ConvBlock(64, 128, 1),
                        ConvBlock(128, 128, 2),
                        ConvBlock(128, 256, 1),
                        ConvBlock(256, 256, 2),
                        ConvBlock(256, 512, 1),
                        ConvBlock(512, 512, 2)
                )
                self.global_pooling = nn.AdaptiveAvgPool2d(output_size=1)
                self.conv2 = nn.Sequential(
                        nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=1),
                        nn.LeakyReLU(negative_slope=0.2)
                )
                self.conv3 = nn.Conv2d(in_channels=1024, out_channels=1, kernel_size=1)

        def forward(self, X):
                X = self.conv1(X)
                X = self.conv_blks(X)
                X = self.global_pooling(X)
                X = self.conv2(X)
                X = self.conv3(X)
                X = X.flatten(start_dim=1)
                return F.sigmoid(X)

class ConvBlock(nn.Module):
        def __init__(self, in_channels, out_channels, strides=1):
                super(ConvBlock, self).__init__()
                self.blk = nn.Sequential(
                        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stri
                        nn.BatchNorm2d(out_channels),
                        nn.LeakyReLU(negative_slope=0.2)
                )
```

```
        def forward(self, X):
                return self.blk(X)
```

In [ ]:

```
#Transforming the images by cropping them
transform_hr = transforms.Compose([
        transforms.CenterCrop(HR_CROPPED_SIZE),
        transforms.ToTensor()
])
transform_lr = transforms.Compose([
        transforms.CenterCrop(LR_CROPPED_SIZE),
        transforms.ToTensor()
])
```

In [ ]:

```
#Defining a class to calculate perceptual loss, adverserial loss and vgg loss
class PerceptualLoss(nn.Module):
        def __init__(self, vgg_coef, adversarial_coef):
                super(PerceptualLoss, self).__init__()
                _vgg19 = vgg19(pretrained=True)
                self.vgg19 = nn.Sequential(*_vgg19.features).eval()
                for p in self.vgg19.parameters():
                        p.requires_grad = False
                self.euclidean_distance = nn.MSELoss()
                self.vgg_coef = vgg_coef
                self.adversarial_coef = adversarial_coef

        def forward(self, sr_img, hr_img, output_labels):
                adversarial_loss = torch.mean(1-output_labels)
                vgg_loss = self.euclidean_distance(self.vgg19(sr_img), self.vgg19(hr_im
                pixel_loss = self.euclidean_distance(sr_img, hr_img)
                return pixel_loss, self.adversarial_coef*adversarial_loss, self.vgg_coe
```

In [ ]:

```
#Creating paths to store checkpoints for the Generator and Discriminator so that the tr
PATH_G = Path("./model/G.pt")
PATH_D = Path("./model/D.pt")
```

In [ ]:

```
#Training
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"device: {str(device).upper()}")

def train(resume_training=True):
        #Loading the data
        data_train_hr, data_train_lr = load_training_data()
        hr_train_loader = DataLoader(dataset=data_train_hr, shuffle=False, batch_size=B
        lr_train_loader = DataLoader(dataset=data_train_lr, shuffle=False, batch_size=B
        assert len(hr_train_loader) == len(lr_train_loader)

        #Loading the models
        G = Generator(n_res_blks=N_RESBLK_G, upscale_factor=UPSCALE).to(device)
        D = Discriminator().to(device)
        optimizer_G = optim.Adam(G.parameters(), lr=LR, betas=BETAS)
        optimizer_D = optim.Adam(D.parameters(), lr=LR, betas=BETAS)

        #Save Checkpoints
```

```
if resume_training and PATH_G.exists() and PATH_D.exists():
                G, D, optimizer_G, optimizer_D, prev_epochs = load_checkpoints(
                print("Continue training from previous checkpoints ...")
                warmup = False
else:
        G.apply(xavier_init_weights)
        D.apply(xavier_init_weights)
        prev_epochs = 0
        summary(G, input_size=(3, LR_CROPPED_SIZE, LR_CROPPED_SIZE), batch_size
        summary(D, input_size=(3, HR_CROPPED_SIZE, HR_CROPPED_SIZE), batch_size
        print("Training from start ...")
        warmup = True

#Training
G.train()
D.train()

#Loss
criterion_G = PerceptualLoss(vgg_coef=VGG_LOSS_COEF, adversarial_coef=ADVERSARI
warmup_loss = torch.nn.L1Loss()
criterion_D = torch.nn.BCELoss()
epoch_pixel_loss = []
epoch_adversial_loss =[]
epoch_vgg_loss =[]
epoch_error_G =[]
#Warming up the Generator
if warmup:
        for w in range(10):
                print(f"\nWarmup: {w+1}")
                for (batch, hr_batch), lr_batch in zip(enumerate(hr_train_loade
                        hr_img, lr_img = hr_batch[0].to(device), lr_batch[0].to
                        optimizer_G.zero_grad()

                        sr_img = G(lr_img)
                        err_G = warmup_loss(sr_img, hr_img)
                        err_G.backward()
                        optimizer_G.step()
                        if batch % 10 == 0:
                                print(f"\tBatch: {batch + 1}/{len(data_train_hr
                                print(f"\tMAE G: {err_G.item():.4f}")

#Epochs
for e in range(EPOCHS):
        batch_adversarial_loss=0
        batch_pixel_loss = 0
        batch_vgg_loss =0
        batch_error_G = 0
        print(f"\nEpoch: {e+prev_epochs+1}")
        n_batches = 0
        for (batch, hr_batch), lr_batch in zip(enumerate(hr_train_loader), lr_t
                n_batches+=1
                # Transfer data to GPU if available
                hr_img, lr_img = hr_batch[0].to(device), lr_batch[0].to(device)

                #Training Discriminator to maximize log(D(x)) + log(1-D(G(z)))
                optimizer_D.zero_grad()

                # Classify all-real HR images
                real_labels = torch.full(size=(len(hr_img),), fill_value=REAL_V
                output_real = D(hr_img).view(-1)
```

```
                        err_D_real = criterion_D(output_real, real_labels)
                        err_D_real.backward()

                        # Classify all-fake HR images (or SR images)
                        fake_labels = torch.full(size=(len(hr_img),), fill_value=FAKE_V
                        sr_img = G(lr_img)

                        output_fake = D(sr_img.detach()).view(-1)
                        err_D_fake = criterion_D(output_fake, fake_labels)
                        err_D_fake.backward()

                        optimizer_D.step()
                        D_Gz1 = output_fake.mean().item()

                        #Training Generator to minimize log(D(G(z))
                        optimizer_G.zero_grad()

                        output_fake = D(sr_img).view(-1)
                        pixel_loss, adversarial_loss, vgg_loss = criterion_G(sr_img, hr
                        err_G = pixel_loss + adversarial_loss + vgg_loss
                        err_G.backward()

                        optimizer_G.step()
                        batch_adversarial_loss += adversarial_loss.item()
                        batch_pixel_loss += pixel_loss.item()
                        batch_vgg_loss +=vgg_loss.item()
                        batch_error_G +=err_G.item()
                        # Print stats
                        if batch%10==0:
                                print(f"\tBatch: {batch + 1}/{len(data_train_hr) // BAT
                                D_x = output_real.mean().item()
                                D_Gz2 = output_fake.mean().item()
                                print(f"\terr_D_real: {err_D_real.item():.4f}; err_D_fa
                                        f" err_G: {err_G.item():.4f}; D_x: {D_x:.4f}; D
                                print(f"\t adversarial_loss: {adversarial_loss:.4f}, vg
                                        f"pixel_loss: {pixel_loss:.4f}")
                        ## Free up GPU memory
                        del hr_img, lr_img, err_D_fake, err_D_real, err_G, real_labels,
                                output_real, output_fake, sr_img, pixel_loss, adversari
                        torch.cuda.empty_cache()
                epoch_adversial_loss.append(batch_adversarial_loss/n_batches)
                epoch_pixel_loss.append(batch_pixel_loss/n_batches)
                epoch_error_G.append(batch_error_G/n_batches)
                epoch_vgg_loss.append(batch_vgg_loss/n_batches)
                n_batches = 0
                #Save checkpoints
                save_checkpoints(G, D, optimizer_G, optimizer_D, epoch=prev_epochs+e+1)

        #Plotting errors
        plt.subplot(2,2,1)
        plt.plot(epoch_adversial_loss)
        plt.subplot(2,2,2)
        plt.plot(epoch_pixel_loss)
        plt.subplot(2,2,3)
        plt.plot(epoch_error_G)
        plt.subplot(2,2,4)
        plt.plot(epoch_vgg_loss)
        plt.show()

def save_checkpoints(G, D, optimizer_G, optimizer_D, epoch):
```

```
        checkpoint_G = {
                'model': G,
                'state_dict': G.state_dict(),
                'optimizer': optimizer_G.state_dict(),
                'epoch': epoch
        }
        checkpoint_D = {
                'model': D,
                'state_dict': D.state_dict(),
                'optimizer': optimizer_D.state_dict(),
        }
        torch.save(checkpoint_G, PATH_G)
        torch.save(checkpoint_D, PATH_D)

def load_checkpoints(G, D, optimizerG, optimizerD):
        print("Loading checkpoints ...")
        checkpoint_G = torch.load(PATH_G)
        checkpoint_D = torch.load(PATH_D)
        G.load_state_dict(checkpoint_G['state_dict'])
        optimizerG.load_state_dict(checkpoint_G['optimizer'])
        D.load_state_dict(checkpoint_D['state_dict'])
        optimizerD.load_state_dict(checkpoint_D['optimizer'])
        prev_epochs = checkpoint_G['epoch']
        print("Loaded checkpoints successfully!")
        return G, D, optimizerG, optimizerD, prev_epochs

def load_training_data():
        data_train_hr = DIV2K(data_dir=os.path.join("./", TRAIN_HR_DIR), transform=tran
        data_train_lr = DIV2K(data_dir=os.path.join("./", TRAIN_LR_DIR), transform=tran
        return data_train_hr, data_train_lr

def xavier_init_weights(model):
        if isinstance(model, torch.nn.Linear) or isinstance(model, torch.nn.Conv2d):
                torch.nn.init.xavier_uniform_(model.weight)

if __name__ == "__main__":
        train()
```

```
device: CUDA
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [16, 64, 100, 100]          15,616
             PReLU-2         [16, 64, 100, 100]               1
            Conv2d-3         [16, 64, 100, 100]          36,928
       BatchNorm2d-4         [16, 64, 100, 100]             128
             PReLU-5         [16, 64, 100, 100]               1
            Conv2d-6         [16, 64, 100, 100]          36,928
       BatchNorm2d-7         [16, 64, 100, 100]             128
    Residual_Block-8         [16, 64, 100, 100]               0
            Conv2d-9         [16, 64, 100, 100]          36,928
      BatchNorm2d-10         [16, 64, 100, 100]             128
             PReLU-11        [16, 64, 100, 100]               1
           Conv2d-12         [16, 64, 100, 100]          36,928
      BatchNorm2d-13         [16, 64, 100, 100]             128
   Residual_Block-14         [16, 64, 100, 100]               0
           Conv2d-15         [16, 64, 100, 100]          36,928
      BatchNorm2d-16         [16, 64, 100, 100]             128
             PReLU-17        [16, 64, 100, 100]               1
           Conv2d-18         [16, 64, 100, 100]          36,928
```

```
       BatchNorm2d-19         [16, 64, 100, 100]             128
     Residual_Block-20         [16, 64, 100, 100]               0
            Conv2d-21         [16, 64, 100, 100]          36,928
       BatchNorm2d-22         [16, 64, 100, 100]             128
              PReLU-23         [16, 64, 100, 100]               1
            Conv2d-24         [16, 64, 100, 100]          36,928
       BatchNorm2d-25         [16, 64, 100, 100]             128
     Residual_Block-26         [16, 64, 100, 100]               0
            Conv2d-27         [16, 64, 100, 100]          36,928
       BatchNorm2d-28         [16, 64, 100, 100]             128
              PReLU-29         [16, 64, 100, 100]               1
            Conv2d-30         [16, 64, 100, 100]          36,928
       BatchNorm2d-31         [16, 64, 100, 100]             128
     Residual_Block-32         [16, 64, 100, 100]               0
            Conv2d-33         [16, 64, 100, 100]          36,928
       BatchNorm2d-34         [16, 64, 100, 100]             128
              PReLU-35         [16, 64, 100, 100]               1
            Conv2d-36         [16, 64, 100, 100]          36,928
       BatchNorm2d-37         [16, 64, 100, 100]             128
     Residual_Block-38         [16, 64, 100, 100]               0
            Conv2d-39         [16, 64, 100, 100]          36,928
       BatchNorm2d-40         [16, 64, 100, 100]             128
              PReLU-41         [16, 64, 100, 100]               1
            Conv2d-42         [16, 64, 100, 100]          36,928
       BatchNorm2d-43         [16, 64, 100, 100]             128
     Residual_Block-44         [16, 64, 100, 100]               0
            Conv2d-45         [16, 64, 100, 100]          36,928
       BatchNorm2d-46         [16, 64, 100, 100]             128
              PReLU-47         [16, 64, 100, 100]               1
            Conv2d-48         [16, 64, 100, 100]          36,928
       BatchNorm2d-49         [16, 64, 100, 100]             128
     Residual_Block-50         [16, 64, 100, 100]               0
            Conv2d-51         [16, 64, 100, 100]          36,928
       BatchNorm2d-52         [16, 64, 100, 100]             128
              PReLU-53         [16, 64, 100, 100]               1
            Conv2d-54         [16, 64, 100, 100]          36,928
       BatchNorm2d-55         [16, 64, 100, 100]             128
     Residual_Block-56         [16, 64, 100, 100]               0
            Conv2d-57         [16, 64, 100, 100]          36,928
       BatchNorm2d-58         [16, 64, 100, 100]             128
              PReLU-59         [16, 64, 100, 100]               1
            Conv2d-60         [16, 64, 100, 100]          36,928
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1933: UserWarning: nn.func
tional.tanh is deprecated. Use torch.tanh instead.
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1944: UserWarning: nn.func
tional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
       BatchNorm2d-61         [16, 64, 100, 100]             128
     Residual_Block-62         [16, 64, 100, 100]               0
            Conv2d-63         [16, 64, 100, 100]          36,928
       BatchNorm2d-64         [16, 64, 100, 100]             128
              PReLU-65         [16, 64, 100, 100]               1
            Conv2d-66         [16, 64, 100, 100]          36,928
       BatchNorm2d-67         [16, 64, 100, 100]             128
     Residual_Block-68         [16, 64, 100, 100]               0
            Conv2d-69         [16, 64, 100, 100]          36,928
       BatchNorm2d-70         [16, 64, 100, 100]             128
              PReLU-71         [16, 64, 100, 100]               1
            Conv2d-72         [16, 64, 100, 100]          36,928
```

```
       BatchNorm2d-73         [16, 64, 100, 100]             128
  Residual_Block-74         [16, 64, 100, 100]               0
         Conv2d-75         [16, 64, 100, 100]          36,928
    BatchNorm2d-76         [16, 64, 100, 100]             128
          PReLU-77         [16, 64, 100, 100]               1
         Conv2d-78         [16, 64, 100, 100]          36,928
    BatchNorm2d-79         [16, 64, 100, 100]             128
  Residual_Block-80         [16, 64, 100, 100]               0
         Conv2d-81         [16, 64, 100, 100]          36,928
    BatchNorm2d-82         [16, 64, 100, 100]             128
          PReLU-83         [16, 64, 100, 100]               1
         Conv2d-84         [16, 64, 100, 100]          36,928
    BatchNorm2d-85         [16, 64, 100, 100]             128
  Residual_Block-86         [16, 64, 100, 100]               0
         Conv2d-87         [16, 64, 100, 100]          36,928
    BatchNorm2d-88         [16, 64, 100, 100]             128
          PReLU-89         [16, 64, 100, 100]               1
         Conv2d-90         [16, 64, 100, 100]          36,928
    BatchNorm2d-91         [16, 64, 100, 100]             128
  Residual_Block-92         [16, 64, 100, 100]               0
         Conv2d-93         [16, 64, 100, 100]          36,928
    BatchNorm2d-94         [16, 64, 100, 100]             128
          PReLU-95         [16, 64, 100, 100]               1
         Conv2d-96         [16, 64, 100, 100]          36,928
    BatchNorm2d-97         [16, 64, 100, 100]             128
  Residual_Block-98         [16, 64, 100, 100]               0
         Conv2d-99         [16, 64, 100, 100]          36,928
   BatchNorm2d-100         [16, 64, 100, 100]             128
         PReLU-101         [16, 64, 100, 100]               1
        Conv2d-102         [16, 64, 100, 100]          36,928
   BatchNorm2d-103         [16, 64, 100, 100]             128
 Residual_Block-104         [16, 64, 100, 100]               0
        Conv2d-105         [16, 64, 100, 100]          36,928
   BatchNorm2d-106         [16, 64, 100, 100]             128
         PReLU-107         [16, 64, 100, 100]               1
        Conv2d-108         [16, 64, 100, 100]          36,928
   BatchNorm2d-109         [16, 64, 100, 100]             128
 Residual_Block-110         [16, 64, 100, 100]               0
        Conv2d-111         [16, 64, 100, 100]          36,928
   BatchNorm2d-112         [16, 64, 100, 100]             128
         PReLU-113         [16, 64, 100, 100]               1
        Conv2d-114         [16, 64, 100, 100]          36,928
   BatchNorm2d-115         [16, 64, 100, 100]             128
 Residual_Block-116         [16, 64, 100, 100]               0
        Conv2d-117         [16, 64, 100, 100]          36,928
   BatchNorm2d-118         [16, 64, 100, 100]             128
         PReLU-119         [16, 64, 100, 100]               1
        Conv2d-120         [16, 64, 100, 100]          36,928
   BatchNorm2d-121         [16, 64, 100, 100]             128
 Residual_Block-122         [16, 64, 100, 100]               0
        Conv2d-123         [16, 64, 100, 100]          36,928
   BatchNorm2d-124         [16, 64, 100, 100]             128
        Conv2d-125        [16, 256, 100, 100]         147,712
   PixelShuffle-126         [16, 64, 200, 200]               0
         PReLU-127         [16, 64, 200, 200]               1
PixelShufflerBlock-128           [16, 64, 200, 200]               0
        Conv2d-129          [16, 3, 200, 200]          15,555
================================================================
Total params: 1,698,201
Trainable params: 1,698,201
```

```
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.83
Forward/backward pass size (MB): 10952.15
Params size (MB): 6.48
Estimated Total Size (MB): 10960.46
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [16, 64, 200, 200]           1,792
         LeakyReLU-2         [16, 64, 200, 200]               0
            Conv2d-3         [16, 64, 100, 100]          36,928
       BatchNorm2d-4         [16, 64, 100, 100]             128
         LeakyReLU-5         [16, 64, 100, 100]               0
         ConvBlock-6         [16, 64, 100, 100]               0
            Conv2d-7        [16, 128, 100, 100]          73,856
       BatchNorm2d-8        [16, 128, 100, 100]             256
         LeakyReLU-9        [16, 128, 100, 100]               0
        ConvBlock-10        [16, 128, 100, 100]               0
           Conv2d-11          [16, 128, 50, 50]         147,584
      BatchNorm2d-12          [16, 128, 50, 50]             256
        LeakyReLU-13          [16, 128, 50, 50]               0
        ConvBlock-14          [16, 128, 50, 50]               0
           Conv2d-15          [16, 256, 50, 50]         295,168
      BatchNorm2d-16          [16, 256, 50, 50]             512
        LeakyReLU-17          [16, 256, 50, 50]               0
        ConvBlock-18          [16, 256, 50, 50]               0
           Conv2d-19          [16, 256, 25, 25]         590,080
      BatchNorm2d-20          [16, 256, 25, 25]             512
        LeakyReLU-21          [16, 256, 25, 25]               0
        ConvBlock-22          [16, 256, 25, 25]               0
           Conv2d-23          [16, 512, 25, 25]       1,180,160
      BatchNorm2d-24          [16, 512, 25, 25]           1,024
        LeakyReLU-25          [16, 512, 25, 25]               0
        ConvBlock-26          [16, 512, 25, 25]               0
           Conv2d-27          [16, 512, 13, 13]       2,359,808
      BatchNorm2d-28          [16, 512, 13, 13]           1,024
        LeakyReLU-29          [16, 512, 13, 13]               0
        ConvBlock-30          [16, 512, 13, 13]               0
AdaptiveAvgPool2d-31            [16, 512, 1, 1]               0
           Conv2d-32           [16, 1024, 1, 1]         525,312
        LeakyReLU-33           [16, 1024, 1, 1]               0
           Conv2d-34              [16, 1, 1, 1]           1,025
================================================================
Total params: 5,215,425
Trainable params: 5,215,425
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 7.32
Forward/backward pass size (MB): 2308.19
Params size (MB): 19.90
Estimated Total Size (MB): 2335.41
----------------------------------------------------------------
Training from start ...

Warmup: 1
        Batch: 1/50
        MAE G: 0.4451
        Batch: 11/50
```

```
                MAE G: 0.1614
                Batch: 21/50
                MAE G: 0.1199
                Batch: 31/50
                MAE G: 0.1057
                Batch: 41/50
                MAE G: 0.1110

        Warmup: 2
                Batch: 1/50
                MAE G: 0.0905
                Batch: 11/50
                MAE G: 0.0899
                Batch: 21/50
                MAE G: 0.0836
                Batch: 31/50
                MAE G: 0.0864
                Batch: 41/50
                MAE G: 0.0955

        Warmup: 3
                Batch: 1/50
                MAE G: 0.0782
                Batch: 11/50
                MAE G: 0.0766
                Batch: 21/50
                MAE G: 0.0757
                Batch: 31/50
                MAE G: 0.0807
                Batch: 41/50
                MAE G: 0.0894

        Warmup: 4
                Batch: 1/50
                MAE G: 0.0694
                Batch: 11/50
                MAE G: 0.0701
                Batch: 21/50
                MAE G: 0.0694
                Batch: 31/50
                MAE G: 0.0713
                Batch: 41/50
                MAE G: 0.0807

        Warmup: 5
                Batch: 1/50
                MAE G: 0.0656
                Batch: 11/50
                MAE G: 0.0646
                Batch: 21/50
                MAE G: 0.0645
                Batch: 31/50
                MAE G: 0.0649
                Batch: 41/50
                MAE G: 0.0757

        Warmup: 6
                Batch: 1/50
                MAE G: 0.0630
                Batch: 11/50
```

```
                MAE G: 0.0620
                Batch: 21/50
                MAE G: 0.0612
                Batch: 31/50
                MAE G: 0.0603
                Batch: 41/50
                MAE G: 0.0725

        Warmup: 7
                Batch: 1/50
                MAE G: 0.0597
                Batch: 11/50
                MAE G: 0.0586
                Batch: 21/50
                MAE G: 0.0590
                Batch: 31/50
                MAE G: 0.0576
                Batch: 41/50
                MAE G: 0.0701

        Warmup: 8
                Batch: 1/50
                MAE G: 0.0571
                Batch: 11/50
                MAE G: 0.0556
                Batch: 21/50
                MAE G: 0.0569
                Batch: 31/50
                MAE G: 0.0548
                Batch: 41/50
                MAE G: 0.0669

        Warmup: 9
                Batch: 1/50
                MAE G: 0.0548
                Batch: 11/50
                MAE G: 0.0539
                Batch: 21/50
                MAE G: 0.0562
                Batch: 31/50
                MAE G: 0.0525
                Batch: 41/50
                MAE G: 0.0626

        Warmup: 10
                Batch: 1/50
                MAE G: 0.0531
                Batch: 11/50
                MAE G: 0.0515
                Batch: 21/50
                MAE G: 0.0544
                Batch: 31/50
                MAE G: 0.0504
                Batch: 41/50
                MAE G: 0.0594

        Epoch: 1
                Batch: 1/50
                err_D_real: 0.5710; err_D_fake: 0.8399;  err_G: 0.0071; D_x: 0.5674; D_Gz1: 0.56
        70; D_Gz2: 0.4136
```

```
         adversarial_loss: 0.0006, vgg_loss: 0.0009, pixel_loss: 0.0056
        Batch: 11/50
        err_D_real: 0.7306; err_D_fake: 0.5697;  err_G: 0.0076; D_x: 0.4850; D_Gz1: 0.43
05; D_Gz2: 0.4307
         adversarial_loss: 0.0006, vgg_loss: 0.0013, pixel_loss: 0.0057
        Batch: 21/50
        err_D_real: 0.5398; err_D_fake: 0.5057;  err_G: 0.0076; D_x: 0.5903; D_Gz1: 0.39
49; D_Gz2: 0.3732
         adversarial_loss: 0.0006, vgg_loss: 0.0014, pixel_loss: 0.0056
        Batch: 31/50
        err_D_real: 0.3422; err_D_fake: 0.6076;  err_G: 0.0067; D_x: 0.7239; D_Gz1: 0.44
65; D_Gz2: 0.3491
         adversarial_loss: 0.0007, vgg_loss: 0.0010, pixel_loss: 0.0050
        Batch: 41/50
        err_D_real: 0.3062; err_D_fake: 0.3213;  err_G: 0.0093; D_x: 0.7511; D_Gz1: 0.26
43; D_Gz2: 0.1691
         adversarial_loss: 0.0008, vgg_loss: 0.0016, pixel_loss: 0.0069

Epoch: 2
        Batch: 1/50
        err_D_real: 0.8643; err_D_fake: 0.1271;  err_G: 0.0067; D_x: 0.4427; D_Gz1: 0.11
56; D_Gz2: 0.1353
         adversarial_loss: 0.0009, vgg_loss: 0.0009, pixel_loss: 0.0049
        Batch: 11/50
        err_D_real: 0.2061; err_D_fake: 0.2917;  err_G: 0.0070; D_x: 0.8396; D_Gz1: 0.24
32; D_Gz2: 0.2603
         adversarial_loss: 0.0007, vgg_loss: 0.0013, pixel_loss: 0.0050
        Batch: 21/50
        err_D_real: 0.1796; err_D_fake: 0.1062;  err_G: 0.0076; D_x: 0.8556; D_Gz1: 0.09
91; D_Gz2: 0.0885
         adversarial_loss: 0.0009, vgg_loss: 0.0014, pixel_loss: 0.0053
        Batch: 31/50
        err_D_real: 0.1292; err_D_fake: 0.1175;  err_G: 0.0069; D_x: 0.9016; D_Gz1: 0.10
94; D_Gz2: 0.0817
         adversarial_loss: 0.0009, vgg_loss: 0.0010, pixel_loss: 0.0049
        Batch: 41/50
        err_D_real: 0.1169; err_D_fake: 0.0516;  err_G: 0.0093; D_x: 0.9155; D_Gz1: 0.04
92; D_Gz2: 0.0715
         adversarial_loss: 0.0009, vgg_loss: 0.0016, pixel_loss: 0.0068

Epoch: 3
        Batch: 1/50
        err_D_real: 0.0775; err_D_fake: 0.0535;  err_G: 0.0066; D_x: 0.9587; D_Gz1: 0.05
08; D_Gz2: 0.0496
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0048
        Batch: 11/50
        err_D_real: 0.1118; err_D_fake: 0.3827;  err_G: 0.0070; D_x: 0.9271; D_Gz1: 0.29
79; D_Gz2: 0.1861
         adversarial_loss: 0.0008, vgg_loss: 0.0012, pixel_loss: 0.0049
        Batch: 21/50
        err_D_real: 0.1207; err_D_fake: 0.1184;  err_G: 0.0075; D_x: 0.9099; D_Gz1: 0.10
61; D_Gz2: 0.1253
         adversarial_loss: 0.0009, vgg_loss: 0.0013, pixel_loss: 0.0052
        Batch: 31/50
        err_D_real: 0.1582; err_D_fake: 0.0402;  err_G: 0.0068; D_x: 0.8786; D_Gz1: 0.03
90; D_Gz2: 0.0401
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0048
        Batch: 41/50
        err_D_real: 0.0976; err_D_fake: 0.0308;  err_G: 0.0092; D_x: 0.9351; D_Gz1: 0.03
01; D_Gz2: 0.0292
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0066

Epoch: 4
        Batch: 1/50
        err_D_real: 0.0883; err_D_fake: 0.0218;  err_G: 0.0066; D_x: 0.9503; D_Gz1: 0.02
14; D_Gz2: 0.0204
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0047
        Batch: 11/50
        err_D_real: 0.0946; err_D_fake: 0.0267;  err_G: 0.0070; D_x: 0.9495; D_Gz1: 0.02
63; D_Gz2: 0.0289
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0048
        Batch: 21/50
        err_D_real: 0.0703; err_D_fake: 0.0214;  err_G: 0.0074; D_x: 0.9700; D_Gz1: 0.02
11; D_Gz2: 0.0197
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0051
        Batch: 31/50
        err_D_real: 0.0650; err_D_fake: 0.0213;  err_G: 0.0067; D_x: 0.9776; D_Gz1: 0.02
10; D_Gz2: 0.0236
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0048
        Batch: 41/50
        err_D_real: 0.0678; err_D_fake: 0.0142;  err_G: 0.0090; D_x: 0.9702; D_Gz1: 0.01
40; D_Gz2: 0.0127
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0065

Epoch: 5
        Batch: 1/50
        err_D_real: 0.0623; err_D_fake: 0.0155;  err_G: 0.0065; D_x: 0.9839; D_Gz1: 0.01
53; D_Gz2: 0.0124
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0047
        Batch: 11/50
        err_D_real: 0.1020; err_D_fake: 0.0269;  err_G: 0.0069; D_x: 0.9485; D_Gz1: 0.02
64; D_Gz2: 0.0473
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0048
        Batch: 21/50
        err_D_real: 0.2725; err_D_fake: 0.0192;  err_G: 0.0073; D_x: 0.7948; D_Gz1: 0.01
83; D_Gz2: 0.0175
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0050
        Batch: 31/50
        err_D_real: 0.0666; err_D_fake: 0.0211;  err_G: 0.0066; D_x: 0.9818; D_Gz1: 0.02
07; D_Gz2: 0.0227
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0047
        Batch: 41/50
        err_D_real: 0.0629; err_D_fake: 0.0065;  err_G: 0.0089; D_x: 0.9788; D_Gz1: 0.00
64; D_Gz2: 0.0060
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0064

Epoch: 6
        Batch: 1/50
        err_D_real: 0.0627; err_D_fake: 0.0063;  err_G: 0.0064; D_x: 0.9865; D_Gz1: 0.00
63; D_Gz2: 0.0065
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0046
        Batch: 11/50
        err_D_real: 0.0812; err_D_fake: 0.0079;  err_G: 0.0068; D_x: 0.9690; D_Gz1: 0.00
78; D_Gz2: 0.0078
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0047
        Batch: 21/50
        err_D_real: 0.0597; err_D_fake: 0.0197;  err_G: 0.0072; D_x: 0.9891; D_Gz1: 0.01
94; D_Gz2: 0.0151
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0049
        Batch: 31/50
```

```
        err_D_real: 0.0654; err_D_fake: 0.0084;  err_G: 0.0066; D_x: 0.9829; D_Gz1: 0.00
84; D_Gz2: 0.0083
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0046
        Batch: 41/50
        err_D_real: 0.0590; err_D_fake: 0.0044;  err_G: 0.0088; D_x: 0.9892; D_Gz1: 0.00
44; D_Gz2: 0.0043
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0063

Epoch: 7
        Batch: 1/50
        err_D_real: 0.0609; err_D_fake: 0.0048;  err_G: 0.0063; D_x: 0.9912; D_Gz1: 0.00
47; D_Gz2: 0.0046
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0045
        Batch: 11/50
        err_D_real: 0.0842; err_D_fake: 0.0054;  err_G: 0.0068; D_x: 0.9660; D_Gz1: 0.00
54; D_Gz2: 0.0052
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0046
        Batch: 21/50
        err_D_real: 0.0651; err_D_fake: 0.0159;  err_G: 0.0071; D_x: 0.9828; D_Gz1: 0.01
57; D_Gz2: 0.0094
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0049
        Batch: 31/50
        err_D_real: 0.0887; err_D_fake: 0.0058;  err_G: 0.0065; D_x: 0.9528; D_Gz1: 0.00
58; D_Gz2: 0.0058
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0046
        Batch: 41/50
        err_D_real: 0.0632; err_D_fake: 0.0034;  err_G: 0.0086; D_x: 0.9866; D_Gz1: 0.00
34; D_Gz2: 0.0031
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0062

Epoch: 8
        Batch: 1/50
        err_D_real: 0.0607; err_D_fake: 0.0035;  err_G: 0.0062; D_x: 0.9957; D_Gz1: 0.00
35; D_Gz2: 0.0034
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0044
        Batch: 11/50
        err_D_real: 0.0717; err_D_fake: 0.0031;  err_G: 0.0067; D_x: 0.9790; D_Gz1: 0.00
31; D_Gz2: 0.0030
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0045
        Batch: 21/50
        err_D_real: 0.0594; err_D_fake: 0.0316;  err_G: 0.0070; D_x: 0.9904; D_Gz1: 0.02
98; D_Gz2: 0.0088
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0048
        Batch: 31/50
        err_D_real: 0.0994; err_D_fake: 0.0038;  err_G: 0.0064; D_x: 0.9432; D_Gz1: 0.00
38; D_Gz2: 0.0032
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0045
        Batch: 41/50
        err_D_real: 0.0664; err_D_fake: 0.0028;  err_G: 0.0085; D_x: 0.9830; D_Gz1: 0.00
28; D_Gz2: 0.0029
         adversarial_loss: 0.0010, vgg_loss: 0.0015, pixel_loss: 0.0060

Epoch: 9
        Batch: 1/50
        err_D_real: 0.0614; err_D_fake: 0.0030;  err_G: 0.0061; D_x: 0.9963; D_Gz1: 0.00
30; D_Gz2: 0.0028
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0044
        Batch: 11/50
        err_D_real: 0.0666; err_D_fake: 0.0035;  err_G: 0.0066; D_x: 0.9848; D_Gz1: 0.00
35; D_Gz2: 0.0034
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0045
        Batch: 21/50
        err_D_real: 0.0600; err_D_fake: 0.0953;  err_G: 0.0069; D_x: 0.9887; D_Gz1: 0.08
31; D_Gz2: 0.0123
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0047
        Batch: 31/50
        err_D_real: 0.0745; err_D_fake: 0.0020;  err_G: 0.0063; D_x: 0.9672; D_Gz1: 0.00
20; D_Gz2: 0.0020
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0044
        Batch: 41/50
        err_D_real: 0.0682; err_D_fake: 0.0019;  err_G: 0.0084; D_x: 0.9799; D_Gz1: 0.00
18; D_Gz2: 0.0020
         adversarial_loss: 0.0010, vgg_loss: 0.0014, pixel_loss: 0.0059

Epoch: 10
        Batch: 1/50
        err_D_real: 0.0610; err_D_fake: 0.0029;  err_G: 0.0061; D_x: 0.9964; D_Gz1: 0.00
29; D_Gz2: 0.0027
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0043
        Batch: 11/50
        err_D_real: 0.0665; err_D_fake: 0.0024;  err_G: 0.0065; D_x: 0.9858; D_Gz1: 0.00
24; D_Gz2: 0.0024
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044
        Batch: 21/50
        err_D_real: 0.0572; err_D_fake: 0.0163;  err_G: 0.0068; D_x: 0.9920; D_Gz1: 0.01
59; D_Gz2: 0.0100
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0046
        Batch: 31/50
        err_D_real: 0.0612; err_D_fake: 0.0044;  err_G: 0.0062; D_x: 0.9883; D_Gz1: 0.00
44; D_Gz2: 0.0039
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0044
        Batch: 41/50
        err_D_real: 0.0615; err_D_fake: 0.0008;  err_G: 0.0082; D_x: 0.9857; D_Gz1: 0.00
08; D_Gz2: 0.0008
         adversarial_loss: 0.0010, vgg_loss: 0.0014, pixel_loss: 0.0058

Epoch: 11
        Batch: 1/50
        err_D_real: 0.0606; err_D_fake: 0.0011;  err_G: 0.0060; D_x: 0.9904; D_Gz1: 0.00
11; D_Gz2: 0.0012
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0042
        Batch: 11/50
        err_D_real: 0.0676; err_D_fake: 0.0018;  err_G: 0.0064; D_x: 0.9844; D_Gz1: 0.00
18; D_Gz2: 0.0018
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044
        Batch: 21/50
        err_D_real: 0.0578; err_D_fake: 0.0038;  err_G: 0.0067; D_x: 0.9904; D_Gz1: 0.00
38; D_Gz2: 0.0036
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0045
        Batch: 31/50
        err_D_real: 0.0601; err_D_fake: 0.0052;  err_G: 0.0062; D_x: 0.9883; D_Gz1: 0.00
52; D_Gz2: 0.0045
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0043
        Batch: 41/50
        err_D_real: 0.0611; err_D_fake: 0.0010;  err_G: 0.0081; D_x: 0.9882; D_Gz1: 0.00
10; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0014, pixel_loss: 0.0057

Epoch: 12
        Batch: 1/50
```

```
        err_D_real: 0.0615; err_D_fake: 0.0009;  err_G: 0.0059; D_x: 0.9919; D_Gz1: 0.00
09; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0042
        Batch: 11/50
        err_D_real: 0.0676; err_D_fake: 0.0013;  err_G: 0.0064; D_x: 0.9826; D_Gz1: 0.00
13; D_Gz2: 0.0013
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043
        Batch: 21/50
        err_D_real: 0.0690; err_D_fake: 0.0218;  err_G: 0.0066; D_x: 0.9730; D_Gz1: 0.02
03; D_Gz2: 0.0109
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0045
        Batch: 31/50
        err_D_real: 0.0738; err_D_fake: 0.0070;  err_G: 0.0061; D_x: 0.9672; D_Gz1: 0.00
69; D_Gz2: 0.0055
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0042
        Batch: 41/50
        err_D_real: 0.0629; err_D_fake: 0.0033;  err_G: 0.0080; D_x: 0.9844; D_Gz1: 0.00
33; D_Gz2: 0.0052
         adversarial_loss: 0.0010, vgg_loss: 0.0014, pixel_loss: 0.0056

Epoch: 13
        Batch: 1/50
        err_D_real: 0.0652; err_D_fake: 0.0147;  err_G: 0.0058; D_x: 0.9936; D_Gz1: 0.01
43; D_Gz2: 0.0097
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0041
        Batch: 11/50
        err_D_real: 0.0720; err_D_fake: 0.0021;  err_G: 0.0063; D_x: 0.9821; D_Gz1: 0.00
21; D_Gz2: 0.0022
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0042
        Batch: 21/50
        err_D_real: 0.0694; err_D_fake: 0.0096;  err_G: 0.0065; D_x: 0.9860; D_Gz1: 0.00
95; D_Gz2: 0.0062
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044
        Batch: 31/50
        err_D_real: 0.0649; err_D_fake: 0.0018;  err_G: 0.0060; D_x: 0.9810; D_Gz1: 0.00
18; D_Gz2: 0.0018
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0042
        Batch: 41/50
        err_D_real: 0.0640; err_D_fake: 0.0033;  err_G: 0.0078; D_x: 0.9900; D_Gz1: 0.00
33; D_Gz2: 0.0028
         adversarial_loss: 0.0010, vgg_loss: 0.0014, pixel_loss: 0.0055

Epoch: 14
        Batch: 1/50
        err_D_real: 0.0621; err_D_fake: 0.0012;  err_G: 0.0058; D_x: 0.9955; D_Gz1: 0.00
12; D_Gz2: 0.0012
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0040
        Batch: 11/50
        err_D_real: 0.0713; err_D_fake: 0.0009;  err_G: 0.0062; D_x: 0.9794; D_Gz1: 0.00
09; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0042
        Batch: 21/50
        err_D_real: 0.0615; err_D_fake: 0.2863;  err_G: 0.0064; D_x: 0.9899; D_Gz1: 0.17
81; D_Gz2: 0.0100
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043
        Batch: 31/50
        err_D_real: 0.0678; err_D_fake: 0.0018;  err_G: 0.0060; D_x: 0.9797; D_Gz1: 0.00
18; D_Gz2: 0.0020
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0041
        Batch: 41/50
```

```
        err_D_real: 0.0755; err_D_fake: 0.0032;  err_G: 0.0077; D_x: 0.9726; D_Gz1: 0.00
32; D_Gz2: 0.0042
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0054

Epoch: 15
        Batch: 1/50
        err_D_real: 0.0687; err_D_fake: 0.0036;  err_G: 0.0057; D_x: 0.9878; D_Gz1: 0.00
36; D_Gz2: 0.0040
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0040
        Batch: 11/50
        err_D_real: 0.0689; err_D_fake: 0.0021;  err_G: 0.0061; D_x: 0.9874; D_Gz1: 0.00
21; D_Gz2: 0.0022
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0041
        Batch: 21/50
        err_D_real: 0.0661; err_D_fake: 0.0054;  err_G: 0.0064; D_x: 0.9894; D_Gz1: 0.00
54; D_Gz2: 0.0052
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043
        Batch: 31/50
        err_D_real: 0.0639; err_D_fake: 0.0151;  err_G: 0.0059; D_x: 0.9935; D_Gz1: 0.01
50; D_Gz2: 0.0109
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0040
        Batch: 41/50
        err_D_real: 0.0669; err_D_fake: 0.0008;  err_G: 0.0076; D_x: 0.9825; D_Gz1: 0.00
08; D_Gz2: 0.0008
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0053

Epoch: 16
        Batch: 1/50
        err_D_real: 0.0617; err_D_fake: 0.0008;  err_G: 0.0056; D_x: 0.9949; D_Gz1: 0.00
08; D_Gz2: 0.0008
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0039
        Batch: 11/50
        err_D_real: 0.0644; err_D_fake: 0.0008;  err_G: 0.0061; D_x: 0.9874; D_Gz1: 0.00
08; D_Gz2: 0.0008
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0041
        Batch: 21/50
        err_D_real: 0.0611; err_D_fake: 0.0103;  err_G: 0.0063; D_x: 0.9896; D_Gz1: 0.01
02; D_Gz2: 0.0091
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0042
        Batch: 31/50
        err_D_real: 0.0813; err_D_fake: 0.0935;  err_G: 0.0058; D_x: 0.9680; D_Gz1: 0.05
96; D_Gz2: 0.0694
         adversarial_loss: 0.0009, vgg_loss: 0.0008, pixel_loss: 0.0040
        Batch: 41/50
        err_D_real: 0.0634; err_D_fake: 0.0017;  err_G: 0.0075; D_x: 0.9820; D_Gz1: 0.00
17; D_Gz2: 0.0024
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0052

Epoch: 17
        Batch: 1/50
        err_D_real: 0.0673; err_D_fake: 0.0036;  err_G: 0.0056; D_x: 0.9895; D_Gz1: 0.00
36; D_Gz2: 0.0033
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0038
        Batch: 11/50
        err_D_real: 0.0734; err_D_fake: 0.0071;  err_G: 0.0060; D_x: 0.9894; D_Gz1: 0.00
70; D_Gz2: 0.0179
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0040
        Batch: 21/50
        err_D_real: 0.0883; err_D_fake: 0.0323;  err_G: 0.0062; D_x: 0.9479; D_Gz1: 0.03
13; D_Gz2: 0.0377
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0041
        Batch: 31/50
        err_D_real: 0.1264; err_D_fake: 0.0412;  err_G: 0.0057; D_x: 0.9203; D_Gz1: 0.03
96; D_Gz2: 0.0410
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0039
        Batch: 41/50
        err_D_real: 0.1343; err_D_fake: 0.0037;  err_G: 0.0074; D_x: 0.9045; D_Gz1: 0.00
37; D_Gz2: 0.0032
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0051

Epoch: 18
        Batch: 1/50
        err_D_real: 0.0659; err_D_fake: 0.0073;  err_G: 0.0055; D_x: 0.9907; D_Gz1: 0.00
72; D_Gz2: 0.0046
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0038
        Batch: 11/50
        err_D_real: 0.0684; err_D_fake: 0.0154;  err_G: 0.0059; D_x: 0.9910; D_Gz1: 0.01
51; D_Gz2: 0.0394
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0039
        Batch: 21/50
        err_D_real: 0.0647; err_D_fake: 0.0208;  err_G: 0.0061; D_x: 0.9778; D_Gz1: 0.02
04; D_Gz2: 0.0318
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0041
        Batch: 31/50
        err_D_real: 0.0669; err_D_fake: 0.0063;  err_G: 0.0057; D_x: 0.9744; D_Gz1: 0.00
62; D_Gz2: 0.0051
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0039
        Batch: 41/50
        err_D_real: 0.2104; err_D_fake: 0.0032;  err_G: 0.0073; D_x: 0.8569; D_Gz1: 0.00
31; D_Gz2: 0.0032
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0051

Epoch: 19
        Batch: 1/50
        err_D_real: 0.0692; err_D_fake: 0.0067;  err_G: 0.0054; D_x: 0.9958; D_Gz1: 0.00
66; D_Gz2: 0.0064
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0037
        Batch: 11/50
        err_D_real: 0.0663; err_D_fake: 0.0023;  err_G: 0.0059; D_x: 0.9914; D_Gz1: 0.00
23; D_Gz2: 0.0020
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0039
        Batch: 21/50
        err_D_real: 0.0597; err_D_fake: 0.2090;  err_G: 0.0060; D_x: 0.9918; D_Gz1: 0.17
93; D_Gz2: 0.0128
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0040
        Batch: 31/50
        err_D_real: 0.1046; err_D_fake: 0.0023;  err_G: 0.0056; D_x: 0.9302; D_Gz1: 0.00
23; D_Gz2: 0.0028
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0038
        Batch: 41/50
        err_D_real: 0.0648; err_D_fake: 0.0159;  err_G: 0.0073; D_x: 0.9884; D_Gz1: 0.01
48; D_Gz2: 0.0135
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0050

Epoch: 20
        Batch: 1/50
        err_D_real: 0.0618; err_D_fake: 0.0016;  err_G: 0.0054; D_x: 0.9954; D_Gz1: 0.00
16; D_Gz2: 0.0015
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0037
        Batch: 11/50
```

```
        err_D_real: 0.0714; err_D_fake: 0.0007;  err_G: 0.0058; D_x: 0.9776; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0038
        Batch: 21/50
        err_D_real: 0.0606; err_D_fake: 0.1164;  err_G: 0.0060; D_x: 0.9912; D_Gz1: 0.10
50; D_Gz2: 0.0275
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0040
        Batch: 31/50
        err_D_real: 0.0944; err_D_fake: 0.0056;  err_G: 0.0056; D_x: 0.9388; D_Gz1: 0.00
56; D_Gz2: 0.0063
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0038
        Batch: 41/50
        err_D_real: 0.0619; err_D_fake: 0.0075;  err_G: 0.0072; D_x: 0.9822; D_Gz1: 0.00
74; D_Gz2: 0.0061
         adversarial_loss: 0.0010, vgg_loss: 0.0013, pixel_loss: 0.0049

Epoch: 21
        Batch: 1/50
        err_D_real: 0.0611; err_D_fake: 0.0032;  err_G: 0.0053; D_x: 0.9947; D_Gz1: 0.00
32; D_Gz2: 0.0030
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0036
        Batch: 11/50
        err_D_real: 0.0670; err_D_fake: 0.0012;  err_G: 0.0058; D_x: 0.9860; D_Gz1: 0.00
12; D_Gz2: 0.0013
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0038
        Batch: 21/50
        err_D_real: 0.0652; err_D_fake: 0.0287;  err_G: 0.0059; D_x: 0.9940; D_Gz1: 0.02
82; D_Gz2: 0.0164
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0039
        Batch: 31/50
        err_D_real: 0.0628; err_D_fake: 0.0020;  err_G: 0.0055; D_x: 0.9937; D_Gz1: 0.00
20; D_Gz2: 0.0018
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0037
        Batch: 41/50
        err_D_real: 0.0771; err_D_fake: 0.0010;  err_G: 0.0071; D_x: 0.9699; D_Gz1: 0.00
10; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0049

Epoch: 22
        Batch: 1/50
        err_D_real: 0.0621; err_D_fake: 0.0022;  err_G: 0.0053; D_x: 0.9935; D_Gz1: 0.00
21; D_Gz2: 0.0021
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0036
        Batch: 11/50
        err_D_real: 0.0652; err_D_fake: 0.0009;  err_G: 0.0057; D_x: 0.9859; D_Gz1: 0.00
09; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0038
        Batch: 21/50
        err_D_real: 0.0618; err_D_fake: 0.0011;  err_G: 0.0058; D_x: 0.9924; D_Gz1: 0.00
11; D_Gz2: 0.0012
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0038
        Batch: 31/50
        err_D_real: 0.0628; err_D_fake: 0.0019;  err_G: 0.0055; D_x: 0.9957; D_Gz1: 0.00
19; D_Gz2: 0.0018
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0037
        Batch: 41/50
        err_D_real: 0.0610; err_D_fake: 0.0012;  err_G: 0.0070; D_x: 0.9897; D_Gz1: 0.00
12; D_Gz2: 0.0012
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0048
```

```
Epoch: 23
        Batch: 1/50
        err_D_real: 0.0597; err_D_fake: 0.0012;  err_G: 0.0052; D_x: 0.9951; D_Gz1: 0.00
12; D_Gz2: 0.0011
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0035
        Batch: 11/50
        err_D_real: 0.0679; err_D_fake: 0.0006;  err_G: 0.0056; D_x: 0.9806; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0037
        Batch: 21/50
        err_D_real: 0.0587; err_D_fake: 0.0005;  err_G: 0.0058; D_x: 0.9895; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0038
        Batch: 31/50
        err_D_real: 0.0591; err_D_fake: 0.0007;  err_G: 0.0054; D_x: 0.9939; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0036
        Batch: 41/50
        err_D_real: 0.0599; err_D_fake: 0.0007;  err_G: 0.0070; D_x: 0.9879; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0048

Epoch: 24
        Batch: 1/50
        err_D_real: 0.0597; err_D_fake: 0.0011;  err_G: 0.0051; D_x: 0.9948; D_Gz1: 0.00
11; D_Gz2: 0.0011
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0035
        Batch: 11/50
        err_D_real: 0.0648; err_D_fake: 0.0007;  err_G: 0.0056; D_x: 0.9843; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0037
        Batch: 21/50
        err_D_real: 0.0581; err_D_fake: 0.0005;  err_G: 0.0057; D_x: 0.9905; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0037
        Batch: 31/50
        err_D_real: 0.0587; err_D_fake: 0.0007;  err_G: 0.0054; D_x: 0.9936; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0036
        Batch: 41/50
        err_D_real: 0.0589; err_D_fake: 0.0006;  err_G: 0.0069; D_x: 0.9881; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0047

Epoch: 25
        Batch: 1/50
        err_D_real: 0.0589; err_D_fake: 0.0010;  err_G: 0.0051; D_x: 0.9942; D_Gz1: 0.00
10; D_Gz2: 0.0010
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0034
        Batch: 11/50
        err_D_real: 0.0643; err_D_fake: 0.0006;  err_G: 0.0055; D_x: 0.9838; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0036
        Batch: 21/50
        err_D_real: 0.0578; err_D_fake: 0.0005;  err_G: 0.0056; D_x: 0.9903; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0037
        Batch: 31/50
        err_D_real: 0.0583; err_D_fake: 0.0007;  err_G: 0.0053; D_x: 0.9933; D_Gz1: 0.00
07; D_Gz2: 0.0007
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0035
        Batch: 41/50
        err_D_real: 0.0583; err_D_fake: 0.0006;  err_G: 0.0068; D_x: 0.9885; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0047

Epoch: 26
        Batch: 1/50
        err_D_real: 0.0584; err_D_fake: 0.0009;  err_G: 0.0050; D_x: 0.9939; D_Gz1: 0.00
09; D_Gz2: 0.0009
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0034
        Batch: 11/50
        err_D_real: 0.0636; err_D_fake: 0.0006;  err_G: 0.0055; D_x: 0.9840; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0036
        Batch: 21/50
        err_D_real: 0.0576; err_D_fake: 0.0005;  err_G: 0.0056; D_x: 0.9901; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0036
        Batch: 31/50
        err_D_real: 0.0581; err_D_fake: 0.0007;  err_G: 0.0052; D_x: 0.9931; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0035
        Batch: 41/50
        err_D_real: 0.0580; err_D_fake: 0.0006;  err_G: 0.0068; D_x: 0.9886; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0046

Epoch: 27
        Batch: 1/50
        err_D_real: 0.0581; err_D_fake: 0.0008;  err_G: 0.0050; D_x: 0.9937; D_Gz1: 0.00
08; D_Gz2: 0.0008
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0034
        Batch: 11/50
        err_D_real: 0.0629; err_D_fake: 0.0006;  err_G: 0.0054; D_x: 0.9844; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0036
        Batch: 21/50
        err_D_real: 0.0575; err_D_fake: 0.0005;  err_G: 0.0055; D_x: 0.9901; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0036
        Batch: 31/50
        err_D_real: 0.0579; err_D_fake: 0.0007;  err_G: 0.0052; D_x: 0.9929; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0035
        Batch: 41/50
        err_D_real: 0.0577; err_D_fake: 0.0005;  err_G: 0.0067; D_x: 0.9887; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0046

Epoch: 28
        Batch: 1/50
        err_D_real: 0.0578; err_D_fake: 0.0008;  err_G: 0.0050; D_x: 0.9935; D_Gz1: 0.00
08; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0033
        Batch: 11/50
        err_D_real: 0.0623; err_D_fake: 0.0005;  err_G: 0.0054; D_x: 0.9847; D_Gz1: 0.00
05; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0035
        Batch: 21/50
```

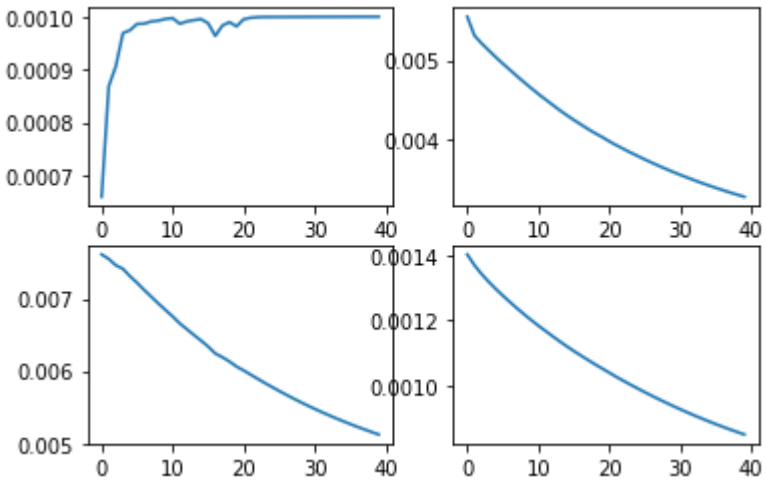
```
        err_D_real: 0.0574; err_D_fake: 0.0004;  err_G: 0.0055; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0035
        Batch: 31/50
        err_D_real: 0.0577; err_D_fake: 0.0006;  err_G: 0.0051; D_x: 0.9928; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0034
        Batch: 41/50
        err_D_real: 0.0575; err_D_fake: 0.0005;  err_G: 0.0067; D_x: 0.9888; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0012, pixel_loss: 0.0045

Epoch: 29
        Batch: 1/50
        err_D_real: 0.0576; err_D_fake: 0.0007;  err_G: 0.0049; D_x: 0.9934; D_Gz1: 0.00
07; D_Gz2: 0.0007
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0033
        Batch: 11/50
        err_D_real: 0.0618; err_D_fake: 0.0005;  err_G: 0.0053; D_x: 0.9851; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0035
        Batch: 21/50
        err_D_real: 0.0573; err_D_fake: 0.0004;  err_G: 0.0054; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0035
        Batch: 31/50
        err_D_real: 0.0576; err_D_fake: 0.0006;  err_G: 0.0051; D_x: 0.9927; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0034
        Batch: 41/50
        err_D_real: 0.0574; err_D_fake: 0.0005;  err_G: 0.0066; D_x: 0.9888; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0045

Epoch: 30
        Batch: 1/50
        err_D_real: 0.0575; err_D_fake: 0.0007;  err_G: 0.0049; D_x: 0.9933; D_Gz1: 0.00
07; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0032
        Batch: 11/50
        err_D_real: 0.0613; err_D_fake: 0.0005;  err_G: 0.0053; D_x: 0.9854; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0035
        Batch: 21/50
        err_D_real: 0.0572; err_D_fake: 0.0004;  err_G: 0.0054; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0034
        Batch: 31/50
        err_D_real: 0.0575; err_D_fake: 0.0006;  err_G: 0.0050; D_x: 0.9926; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0033
        Batch: 41/50
        err_D_real: 0.0573; err_D_fake: 0.0005;  err_G: 0.0066; D_x: 0.9889; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044

Epoch: 31
        Batch: 1/50
        err_D_real: 0.0573; err_D_fake: 0.0006;  err_G: 0.0048; D_x: 0.9932; D_Gz1: 0.00
06; D_Gz2: 0.0006
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0032
        Batch: 11/50
        err_D_real: 0.0609; err_D_fake: 0.0005;  err_G: 0.0053; D_x: 0.9857; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0034
        Batch: 21/50
        err_D_real: 0.0572; err_D_fake: 0.0004;  err_G: 0.0053; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0034
        Batch: 31/50
        err_D_real: 0.0575; err_D_fake: 0.0006;  err_G: 0.0050; D_x: 0.9925; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0033
        Batch: 41/50
        err_D_real: 0.0572; err_D_fake: 0.0004;  err_G: 0.0065; D_x: 0.9889; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044

Epoch: 32
        Batch: 1/50
        err_D_real: 0.0572; err_D_fake: 0.0006;  err_G: 0.0048; D_x: 0.9931; D_Gz1: 0.00
06; D_Gz2: 0.0006
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0032
        Batch: 11/50
        err_D_real: 0.0605; err_D_fake: 0.0004;  err_G: 0.0052; D_x: 0.9860; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0034
        Batch: 21/50
        err_D_real: 0.0571; err_D_fake: 0.0004;  err_G: 0.0052; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0034
        Batch: 31/50
        err_D_real: 0.0574; err_D_fake: 0.0006;  err_G: 0.0049; D_x: 0.9924; D_Gz1: 0.00
06; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0033
        Batch: 41/50
        err_D_real: 0.0571; err_D_fake: 0.0004;  err_G: 0.0065; D_x: 0.9890; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0044

Epoch: 33
        Batch: 1/50
        err_D_real: 0.0571; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9931; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0031
        Batch: 11/50
        err_D_real: 0.0601; err_D_fake: 0.0004;  err_G: 0.0052; D_x: 0.9862; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0033
        Batch: 21/50
        err_D_real: 0.0571; err_D_fake: 0.0004;  err_G: 0.0052; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0033
        Batch: 31/50
        err_D_real: 0.0573; err_D_fake: 0.0005;  err_G: 0.0049; D_x: 0.9924; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0032
        Batch: 41/50
        err_D_real: 0.0570; err_D_fake: 0.0004;  err_G: 0.0064; D_x: 0.9890; D_Gz1: 0.00
04; D_Gz2: 0.0004
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043

Epoch: 34
        Batch: 1/50
        err_D_real: 0.0571; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9930; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0031
        Batch: 11/50
        err_D_real: 0.0598; err_D_fake: 0.0004;  err_G: 0.0051; D_x: 0.9864; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0033
        Batch: 21/50
        err_D_real: 0.0570; err_D_fake: 0.0004;  err_G: 0.0052; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0033
        Batch: 31/50
        err_D_real: 0.0573; err_D_fake: 0.0005;  err_G: 0.0049; D_x: 0.9923; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0032
        Batch: 41/50
        err_D_real: 0.0570; err_D_fake: 0.0004;  err_G: 0.0064; D_x: 0.9891; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043

Epoch: 35
        Batch: 1/50
        err_D_real: 0.0570; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9929; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0031
        Batch: 11/50
        err_D_real: 0.0596; err_D_fake: 0.0004;  err_G: 0.0051; D_x: 0.9866; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0033
        Batch: 21/50
        err_D_real: 0.0570; err_D_fake: 0.0004;  err_G: 0.0051; D_x: 0.9900; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0032
        Batch: 31/50
        err_D_real: 0.0572; err_D_fake: 0.0005;  err_G: 0.0048; D_x: 0.9923; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0031
        Batch: 41/50
        err_D_real: 0.0569; err_D_fake: 0.0004;  err_G: 0.0063; D_x: 0.9892; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0043

Epoch: 36
        Batch: 1/50
        err_D_real: 0.0570; err_D_fake: 0.0004;  err_G: 0.0046; D_x: 0.9928; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 11/50
        err_D_real: 0.0593; err_D_fake: 0.0004;  err_G: 0.0050; D_x: 0.9868; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0032
        Batch: 21/50
        err_D_real: 0.0569; err_D_fake: 0.0003;  err_G: 0.0051; D_x: 0.9900; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0009, pixel_loss: 0.0032
        Batch: 31/50
```

```
        err_D_real: 0.0572; err_D_fake: 0.0005;  err_G: 0.0048; D_x: 0.9922; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0031
        Batch: 41/50
        err_D_real: 0.0568; err_D_fake: 0.0004;  err_G: 0.0063; D_x: 0.9892; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0042

Epoch: 37
        Batch: 1/50
        err_D_real: 0.0569; err_D_fake: 0.0004;  err_G: 0.0046; D_x: 0.9928; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 11/50
        err_D_real: 0.0591; err_D_fake: 0.0004;  err_G: 0.0050; D_x: 0.9869; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0032
        Batch: 21/50
        err_D_real: 0.0569; err_D_fake: 0.0003;  err_G: 0.0050; D_x: 0.9901; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0032
        Batch: 31/50
        err_D_real: 0.0571; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9922; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0007, pixel_loss: 0.0031
        Batch: 41/50
        err_D_real: 0.0568; err_D_fake: 0.0003;  err_G: 0.0063; D_x: 0.9893; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0042

Epoch: 38
        Batch: 1/50
        err_D_real: 0.0569; err_D_fake: 0.0004;  err_G: 0.0046; D_x: 0.9927; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 11/50
        err_D_real: 0.0589; err_D_fake: 0.0004;  err_G: 0.0050; D_x: 0.9871; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0032
        Batch: 21/50
        err_D_real: 0.0569; err_D_fake: 0.0003;  err_G: 0.0050; D_x: 0.9901; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0031
        Batch: 31/50
        err_D_real: 0.0571; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9922; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0031
        Batch: 41/50
        err_D_real: 0.0567; err_D_fake: 0.0003;  err_G: 0.0062; D_x: 0.9893; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0011, pixel_loss: 0.0042

Epoch: 39
        Batch: 1/50
        err_D_real: 0.0568; err_D_fake: 0.0004;  err_G: 0.0045; D_x: 0.9927; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 11/50
        err_D_real: 0.0587; err_D_fake: 0.0003;  err_G: 0.0049; D_x: 0.9872; D_Gz1: 0.00
03; D_Gz2: 0.0003
```

```
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0032
        Batch: 21/50
        err_D_real: 0.0568; err_D_fake: 0.0003;  err_G: 0.0049; D_x: 0.9901; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0031
        Batch: 31/50
        err_D_real: 0.0570; err_D_fake: 0.0005;  err_G: 0.0047; D_x: 0.9922; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 41/50
        err_D_real: 0.0567; err_D_fake: 0.0003;  err_G: 0.0062; D_x: 0.9894; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0041

Epoch: 40
        Batch: 1/50
        err_D_real: 0.0568; err_D_fake: 0.0004;  err_G: 0.0045; D_x: 0.9926; D_Gz1: 0.00
04; D_Gz2: 0.0004
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0029
        Batch: 11/50
        err_D_real: 0.0586; err_D_fake: 0.0003;  err_G: 0.0049; D_x: 0.9873; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0031
        Batch: 21/50
        err_D_real: 0.0568; err_D_fake: 0.0003;  err_G: 0.0049; D_x: 0.9901; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0008, pixel_loss: 0.0031
        Batch: 31/50
        err_D_real: 0.0570; err_D_fake: 0.0005;  err_G: 0.0046; D_x: 0.9921; D_Gz1: 0.00
05; D_Gz2: 0.0005
         adversarial_loss: 0.0010, vgg_loss: 0.0006, pixel_loss: 0.0030
        Batch: 41/50
        err_D_real: 0.0567; err_D_fake: 0.0003;  err_G: 0.0061; D_x: 0.9894; D_Gz1: 0.00
03; D_Gz2: 0.0003
         adversarial_loss: 0.0010, vgg_loss: 0.0010, pixel_loss: 0.0041
```



In [ ]:

```
#Testing on the Test Set -> The SR images generated are saved in the 'sr_img' folder
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
SR_DIR = "./sr_img"

def generate_sr(lr_img_path):
        with torch.no_grad():
                pil_img = Image.open(lr_img_path)
                img_tensor = transforms.ToTensor()(pil_img)
```

```
                img_tensor = torch.unsqueeze(img_tensor, 0) # add batch dimension
                img_tensor = img_tensor.to(device)
                sr_img = G(img_tensor)
                print(f"Upscaled from size [{img_tensor.shape[2]}, {img_tensor.shape[3]
        file_name = lr_img_path.split('/')[-1]
        sr_img_path = os.path.join(SR_DIR, f"sr_{file_name}")
        tensor_to_img(sr_img, sr_img_path)

def tensor_to_img(tensor, filepath):
        tensor = tensor.cpu()
        pil = transforms.ToPILImage()(tensor.squeeze_(0))
        pil.save(filepath)
        print(f"Saved to {filepath}")

# Load checkpoints
G = Generator(n_res_blks=N_RESBLK_G, upscale_factor=UPSCALE)
if PATH_G.exists():
        checkpoint_G = torch.load(PATH_G)
        G.load_state_dict(checkpoint_G['state_dict'])
        G.to(device)
else:
        print("Checkpoints not found, using Xavier initialization.")
        G.apply(xavier_init_weights).to(device)
G.eval()
generate_sr('./DIV2K_valid_LR_bicubic/X2/0860x2.png')
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1933: UserWarning: nn.func
tional.tanh is deprecated. Use torch.tanh instead.
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
Upscaled from size [768, 1020] to [1536, 2040]
Saved to ./sr_img/sr_0860x2.png
```

In [ ]: