DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC452 Machine Learning

Generative Adversarial Networks (GANs)

SUBMITTED TO:

Prof Dr Asad Mansoor khan

SUBMITTED BY:

Uzair Waheed

Reg # 371828

DE- 43 CE

Submission Date:13/05/24

Tasks:

1. A detailed report outlining what you have done and what approaches you have selected along with your results. It should also include any other information that you deem necessary.

2. A working demo of the GAN that you have implemented in the form of a notebook.

3. Fréchet Inception Distance (FID) score of images from the test_photos folder. You can read more about this metric on the Web and you can also use an already available implementation.

Solution:

The assignment is about image-to-image translation task which is convert the real-world photos to Monet style paintings. Defined Generator architecture based on cycle Gan which comprises of encoder(down sampling), residual blocks and decoder(up sampling). In the Cycle GAN training, two generators and two discriminators are used: one generator translates images from the photo domain to the Monet painting domain, while the second reconstructs the original image to enforce cycle-consistency; discriminators distinguish real from generated images, optimizing generators via adversarial and cycle-consistency losses. Here, we used weights of pretrained model. Input images are fetched from google drive. Preprocessing, implemented with the albumentations library, involves resizing images to 256x256 pixels, normalizing pixel values to [-1, 1] (mean=0.5, std=0.5 per RGB channel), and converting to PyTorch tensors in [C, H, W] format, with a batch dimension added (shape: [1, 3, 256, 256]). Images are loaded with OpenCV and converted from BGR to RGB. During inference, the Generator processes each input tensor to produce a stylized output, which is post-processed by permuting dimensions to [H, W, C], denormalizing to [0, 255] via scaling and shifting, and converting to uint8. Outputs are converted to BGR and saved to an output directory with original filenames, using tqdm for progress visualization. The FID score, printed as output, quantifies similarity.

Code:

```
import os
import torch
import torch.nn as nn
import numpy as np
import cv2
from tqdm import tqdm
import albumentations as A
from albumentations.pytorch import ToTensorV2
from torch_fidelity import calculate_metrics
```

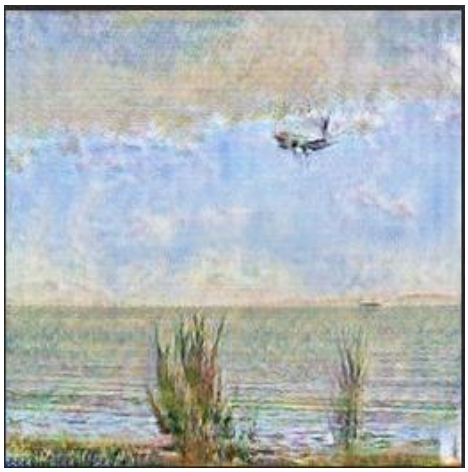
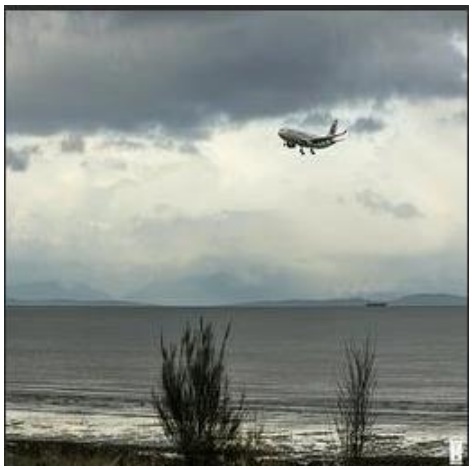
```
import os
from google.colab import drive

drive.mount('/content/drive')


class DownConv(nn.Module):
    def __init__(self, in_filters, out_filters):
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_filters, out_filters, kernel_size=3, stride=2,
padding=1, padding_mode='reflect'),
            nn.InstanceNorm2d(out_filters),
            nn.ReLU(inplace=True)
        )

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

class UpConv(nn.Module):
    def __init__(self, in_filters, out_filters):
        super().__init__()
        self.block = nn.Sequential(
            nn.ConvTranspose2d(in_filters, out_filters, kernel_size=3,
stride=2, padding=1, output_padding=1),
            nn.InstanceNorm2d(out_filters),
            nn.ReLU(inplace=True)
        )

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

class ResBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.block = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(channels, channels, 3),
            nn.InstanceNorm2d(channels),
            nn.ReLU(inplace=True),
            nn.ReflectionPad2d(1),
            nn.Conv2d(channels, channels, 3),
            nn.InstanceNorm2d(channels)
        )

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

class Generator(nn.Module):
```

```
    def __init__(self, img_channels, num_res=9):
        super().__init__()
        self.conv_1 = nn.Sequential(
            nn.Conv2d(img_channels, out_channels=64, kernel_size=7, padding=3,
padding_mode='reflect'),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True))

        self.down = nn.Sequential(
            DownConv(64, 128),
            DownConv(128, 256))

        self.bottleneck = nn.Sequential(*[ResBlock(256) for _ in
range(num_res)])

        self.up = nn.Sequential(
            UpConv(256, 128),
            UpConv(128, 64))

        self.conv_2 = nn.Conv2d(64, img_channels, kernel_size=7, stride=1,
padding=3, padding_mode='reflect')

    def forward(self, x):
        x = self.conv_1(x)
        x = self.down(x)
        x = self.bottleneck(x)
        x = self.up(x)
        return torch.tanh(self.conv_2(x))

state_dict_path = '/content/drive/MyDrive/gen_monet_dict_1.pth'
model = Generator(3)
model.load_state_dict(torch.load(state_dict_path,
map_location=torch.device('cpu')))
model.eval()

transform = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=[0.5]*3, std=[0.5]*3, max_pixel_value=255.0),
    ToTensorV2()
])

input_dir = '/content/drive/MyDrive/test_photos'
output_dir = '/content/drive/MyDrive/monet_output'

os.makedirs(output_dir, exist_ok=True)

for filename in tqdm(os.listdir(input_dir)):
    if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
```

```
        path = os.path.join(input_dir, filename)
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        transformed = transform(image=image)['image'].unsqueeze(0)

        with torch.no_grad():
            output = model(transformed)
            output = output.squeeze().permute(1, 2, 0).numpy()
            output = ((output + 1) * 127.5).astype(np.uint8)

        out_path = os.path.join(output_dir, filename)
        cv2.imwrite(out_path, cv2.cvtColor(output, cv2.COLOR_RGB2BGR))


metrics = calculate_metrics(
    input1=input_dir,
    input2=output_dir,
    cuda=True,
    isc=False,
    fid=True
)

print( metrics['frechet_inception_distance'])
```

Output:

Monet: *Real:*

Test_photos:
https://drive.google.com/drive/folders/1G3KHmGTHnFEZELBsk7OTWMui6BvrPja?usp=drive_link

Monet_output:

https://drive.google.com/drive/folders/16rHuTybLIm3yLLSkIFyK8IJyIPUS4NV?usp=drive_link

FID_Score :

```
Creating feature extractor "inception-v3-compat" with features ['2048']
Extracting statistics from input 1
Looking for samples non-recursivelty in "/content/drive/MyDrive/test_photos" with extensions png,jpg,jpeg
Found 400 samples, some are lossy-compressed - this may affect metrics
Processing samples
Extracting statistics from input 2
Looking for samples non-recursivelty in "/content/drive/MyDrive/monet_output" with extensions png,jpg,jpeg
Found 400 samples, some are lossy-compressed - this may affect metrics
Processing samples
135.4182421594357
Frechet Inception Distance: 135.4182421594357
```