Amardeep Singh

E23CSEU2189

```
!wget http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/edges2shoe
!tar -xvf edges2shoes.tar.gz
```

```
edges2shoes/train/47281_AB.jpg
edges2shoes/train/47282_AB.jpg
edges2shoes/train/47283_AB.jpg
edges2shoes/train/47284_AB.jpg
edges2shoes/train/47285_AB.jpg
edges2shoes/train/47286_AB.jpg
edges2shoes/train/47287_AB.jpg
edges2shoes/train/47288_AB.jpg
edges2shoes/train/47289_AB.jpg
edges2shoes/train/4728_AB.jpg
edges2shoes/train/47290_AB.jpg
edges2shoes/train/47291_AB.jpg
edges2shoes/train/47292_AB.jpg
edges2shoes/train/47293_AB.jpg
edges2shoes/train/47294_AB.jpg
edges2shoes/train/48843_AB.jpg
edges2shoes/train/48844_AB.jpg
edges2shoes/train/48845_AB.jpg
edges2shoes/train/48846_AB.jpg
edges2shoes/train/48847_AB.jpg
edges2shoes/train/48848_AB.jpg
edges2shoes/train/48849_AB.jpg
edges2shoes/train/4884_AB.jpg
edges2shoes/train/48850_AB.jpg
edges2shoes/train/48851_AB.jpg
edges2shoes/train/48852_AB.jpg
edges2shoes/train/48853_AB.jpg
edges2shoes/train/48854_AB.jpg
edges2shoes/train/48855_AB.jpg
edges2shoes/train/48856_AB.jpg
edges2shoes/train/48857_AB.jpg
edges2shoes/train/48858_AB.jpg
edges2shoes/train/48859_AB.jpg
edges2shoes/train/480_AB.jpg
edges2shoes/train/48100_AB.jpg
edges2shoes/train/48101_AB.jpg
edges2shoes/train/48102_AB.jpg
edges2shoes/train/48103_AB.jpg
edges2shoes/train/48104_AB.jpg
edges2shoes/train/48105_AB.jpg
edges2shoes/train/48106_AB.jpg
edges2shoes/train/48107_AB.jpg
```

```
edges2shoes/train/48108_AB.jpg
edges2shoes/train/48109_AB.jpg
edges2shoes/train/4810_AB.jpg
edges2shoes/train/48110_AB.jpg
edges2shoes/train/48111_AB.jpg
edges2shoes/train/48112_AB.jpg
edges2shoes/train/48113_AB.jpg
edges2shoes/train/48114_AB.jpg
edges2shoes/train/48115_AB.jpg
edges2shoes/train/48116_AB.jpg
edges2shoes/train/48117_AB.jpg
edges2shoes/train/48118_AB.jpg
edges2shoes/train/48119_AB.jpg
edges2shoes/train/49521_AB.jpg
edges2shoes/train/49522_AB.jpg
edges2shoes/train/49523_AB.jpg
edges2shoes/train/49524_AB.jpg
```

```
import os
from PIL import Image

os.makedirs("dataset/train/input", exist_ok=True)
os.makedirs("dataset/train/target", exist_ok=True)

source_folder = "edges2shoes/train"

for i, img_name in enumerate(os.listdir(source_folder)):
    if i > 1000:  # LIMIT for faster training (remove this line for
        break

    img = Image.open(os.path.join(source_folder, img_name))
    w, h = img.size

    input_img = img.crop((0, 0, w//2, h))
    target_img = img.crop((w//2, 0, w, h))

    input_img.save(f"dataset/train/input/{img_name}")
    target_img.save(f"dataset/train/target/{img_name}")

print("Dataset Prepared Successfully!")
```

```
Dataset Prepared Successfully!
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from PIL import Image
```

```
import os
import matplotlib.pyplot as plt

# ================================
# CONFIG
# ================================
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu
BATCH_SIZE = 4
LR = 0.0002
EPOCHS = 50
LAMBDA_L1 = 100
IMAGE_SIZE = 256

# ================================
# DATASET
# ================================
class PairedDataset(Dataset):
    def __init__(self, input_dir, target_dir, transform=None):
        self.input_dir = input_dir
        self.target_dir = target_dir
        self.transform = transform
        self.images = os.listdir(input_dir)

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

    def __getitem__(self, idx):
        img_name = self.images[idx]
        input_img = Image.open(os.path.join(self.input_dir, img_nam
        target_img = Image.open(os.path.join(self.target_dir, img_n

        if self.transform:
            input_img = self.transform(input_img)
            target_img = self.transform(target_img)

        return input_img, target_img

transform = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = PairedDataset(
    "dataset/train/input",
    "dataset/train/target",
    transform
)
```

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shu

# ================================
# U-NET GENERATOR
# ================================
class Down(nn.Module):
    def __init__(self, in_c, out_c, normalize=True):
        super().__init__()
        layers = [nn.Conv2d(in_c, out_c, 4, 2, 1, bias=False)]
        if normalize:
            layers.append(nn.BatchNorm2d(out_c))
        layers.append(nn.LeakyReLU(0.2))
        self.model = nn.Sequential(*layers)

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

class Up(nn.Module):
    def __init__(self, in_c, out_c):
        super().__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(in_c, out_c, 4, 2, 1, bias=False),
            nn.BatchNorm2d(out_c),
            nn.ReLU(True)
        )

    def forward(self, x, skip):
        x = self.model(x)
        return torch.cat([x, skip], 1)

class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.d1 = Down(3, 64, False)
        self.d2 = Down(64, 128)
        self.d3 = Down(128, 256)
        self.d4 = Down(256, 512)

        self.u1 = Up(512, 256)
        self.u2 = Up(512, 128)
        self.u3 = Up(256, 64)

        self.final = nn.Sequential(
            nn.ConvTranspose2d(128, 3, 4, 2, 1),
            nn.Tanh()
        )
```

```
        def forward(self, x):
            d1 = self.d1(x)
            d2 = self.d2(d1)
            d3 = self.d3(d2)
            d4 = self.d4(d3)

            u1 = self.u1(d4, d3)
            u2 = self.u2(u1, d2)
            u3 = self.u3(u2, d1)

            return self.final(u3)

    # ================================
    # PATCHGAN DISCRIMINATOR
    # ================================
    class Discriminator(nn.Module):
        def __init__(self):
            super().__init__()
            self.model = nn.Sequential(
                nn.Conv2d(6, 64, 4, 2, 1),
                nn.LeakyReLU(0.2),

                nn.Conv2d(64, 128, 4, 2, 1),
                nn.BatchNorm2d(128),
                nn.LeakyReLU(0.2),

                nn.Conv2d(128, 256, 4, 2, 1),
                nn.BatchNorm2d(256),
                nn.LeakyReLU(0.2),

                nn.Conv2d(256, 1, 4, 1, 1)
            )

        def forward(self, input_img, target_img):
            x = torch.cat([input_img, target_img], dim=1)
            return self.model(x)

    # ================================
    # BASELINE CNN
    # ================================
    class BaselineCNN(nn.Module):
        def __init__(self):
            super().__init__()
            self.encoder = nn.Sequential(
                nn.Conv2d(3, 64, 4, 2, 1),
                nn.ReLU(),
                nn.Conv2d(64, 128, 4, 2, 1),
                nn.ReLU()
```

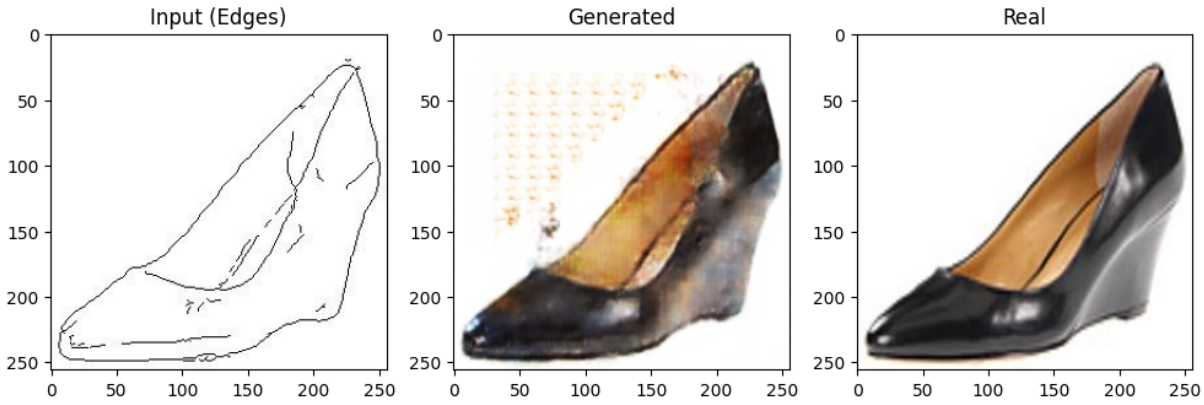
```
            )

            self.decoder = nn.Sequential(
                nn.ConvTranspose2d(128, 64, 4, 2, 1),
                nn.ReLU(),
                nn.ConvTranspose2d(64, 3, 4, 2, 1),
                nn.Tanh()
            )

        def forward(self, x):
            return self.decoder(self.encoder(x))

    # ================================
    # INITIALIZE MODELS
    # ================================
    generator = Generator().to(DEVICE)
    discriminator = Discriminator().to(DEVICE)
    baseline = BaselineCNN().to(DEVICE)

    opt_G = optim.Adam(generator.parameters(), lr=LR, betas=(0.5, 0.999
    opt_D = optim.Adam(discriminator.parameters(), lr=LR, betas=(0.5, 0
    opt_base = optim.Adam(baseline.parameters(), lr=LR)

    criterion_GAN = nn.MSELoss()
    criterion_L1 = nn.L1Loss()

    # ================================
    # TRAINING LOOP
    # ================================
    for epoch in range(EPOCHS):
        for input_img, target_img in train_loader:
            input_img = input_img.to(DEVICE)
            target_img = target_img.to(DEVICE)

            # ---------------------
            # Train Discriminator
            # ---------------------
            opt_D.zero_grad()
            fake_img = generator(input_img)

            real_pred = discriminator(input_img, target_img)
            fake_pred = discriminator(input_img, fake_img.detach())

            loss_real = criterion_GAN(real_pred, torch.ones_like(real_p
            loss_fake = criterion_GAN(fake_pred, torch.zeros_like(fake_
            loss_D = (loss_real + loss_fake) * 0.5

            loss_D.backward()
```

```
            opt_D.step()

            # ---------------------
            # Train Generator
            # ---------------------
            opt_G.zero_grad()
            fake_pred = discriminator(input_img, fake_img)

            loss_GAN = criterion_GAN(fake_pred, torch.ones_like(fake_pr
            loss_L1 = criterion_L1(fake_img, target_img) * LAMBDA_L1
            loss_G = loss_GAN + loss_L1

            loss_G.backward()
            opt_G.step()

            # ---------------------
            # Train Baseline CNN
            # ---------------------
            opt_base.zero_grad()
            base_out = baseline(input_img)
            loss_base = criterion_L1(base_out, target_img)
            loss_base.backward()
            opt_base.step()

        print(f"Epoch [{epoch+1}/{EPOCHS}] | D: {loss_D.item():.4f} | G

    # ================================
    # SAVE MODELS
    # ================================
    torch.save(generator.state_dict(), "generator.pth")
    torch.save(discriminator.state_dict(), "discriminator.pth")
    torch.save(baseline.state_dict(), "baseline.pth")

    print("Training Complete!")
```

```
Epoch [1/50] | D: 0.1326 | G: 23.8385 | Base: 0.3046
Epoch [2/50] | D: 0.1740 | G: 23.6484 | Base: 0.4318
Epoch [3/50] | D: 0.2696 | G: 20.5333 | Base: 0.4286
Epoch [4/50] | D: 0.2144 | G: 16.2506 | Base: 0.5296
Epoch [5/50] | D: 0.0709 | G: 12.5423 | Base: 0.2945
Epoch [6/50] | D: 0.0793 | G: 23.4040 | Base: 0.2501
Epoch [7/50] | D: 0.0875 | G: 28.0215 | Base: 0.2757
Epoch [8/50] | D: 0.1936 | G: 12.7036 | Base: 0.3238
Epoch [9/50] | D: 0.3700 | G: 17.4188 | Base: 0.4683
Epoch [10/50] | D: 0.0661 | G: 29.0178 | Base: 0.3511
Epoch [11/50] | D: 0.1447 | G: 12.4998 | Base: 0.5132
Epoch [12/50] | D: 0.0819 | G: 13.6663 | Base: 0.3779
Epoch [13/50] | D: 0.0919 | G: 14.3000 | Base: 0.3940
Epoch [14/50] | D: 0.0555 | G: 12.9992 | Base: 0.4653
```

```
Epoch [15/50] | D: 0.3715 | G: 13.9545 | Base: 0.4313
Epoch [16/50] | D: 0.2921 | G: 8.2361 | Base: 0.1598
Epoch [17/50] | D: 0.1919 | G: 13.8081 | Base: 0.3025
Epoch [18/50] | D: 0.1326 | G: 13.0462 | Base: 0.6527
Epoch [19/50] | D: 0.2411 | G: 5.2493 | Base: 0.2288
Epoch [20/50] | D: 0.0493 | G: 9.2747 | Base: 0.3478
Epoch [21/50] | D: 0.2254 | G: 11.3301 | Base: 0.6343
Epoch [22/50] | D: 0.2876 | G: 15.3689 | Base: 0.3124
Epoch [23/50] | D: 0.1354 | G: 6.4848 | Base: 0.2591
Epoch [24/50] | D: 0.0393 | G: 5.6221 | Base: 0.4948
Epoch [25/50] | D: 0.1374 | G: 13.6044 | Base: 0.2861
Epoch [26/50] | D: 0.1897 | G: 3.9318 | Base: 0.2597
Epoch [27/50] | D: 0.1872 | G: 9.1262 | Base: 0.5169
Epoch [28/50] | D: 0.1604 | G: 9.2063 | Base: 0.4226
Epoch [29/50] | D: 0.2314 | G: 7.1753 | Base: 0.2388
Epoch [30/50] | D: 0.0972 | G: 8.4770 | Base: 0.4747
Epoch [31/50] | D: 0.1922 | G: 6.6966 | Base: 0.0946
Epoch [32/50] | D: 0.2732 | G: 6.1737 | Base: 0.2416
Epoch [33/50] | D: 0.1113 | G: 6.8787 | Base: 0.1766
Epoch [34/50] | D: 0.2740 | G: 3.8584 | Base: 0.5602
Epoch [35/50] | D: 0.1239 | G: 16.1348 | Base: 0.4111
Epoch [36/50] | D: 0.2689 | G: 9.2613 | Base: 0.4196
Epoch [37/50] | D: 0.1940 | G: 6.5232 | Base: 0.6022
Epoch [38/50] | D: 0.0922 | G: 5.6428 | Base: 0.3318
Epoch [39/50] | D: 0.0779 | G: 7.7947 | Base: 0.3940
Epoch [40/50] | D: 0.1708 | G: 5.6364 | Base: 0.5595
Epoch [41/50] | D: 0.1839 | G: 7.6923 | Base: 0.3897
Epoch [42/50] | D: 0.2386 | G: 12.7532 | Base: 0.3503
Epoch [43/50] | D: 0.1116 | G: 4.4683 | Base: 0.3012
Epoch [44/50] | D: 0.3022 | G: 4.1060 | Base: 0.1722
Epoch [45/50] | D: 0.3117 | G: 4.3732 | Base: 0.2126
Epoch [46/50] | D: 0.2686 | G: 7.9298 | Base: 0.3121
Epoch [47/50] | D: 0.1358 | G: 7.9810 | Base: 0.3704
Epoch [48/50] | D: 0.1213 | G: 9.1755 | Base: 0.2516
Epoch [49/50] | D: 0.2128 | G: 3.6445 | Base: 0.3876
Epoch [50/50] | D: 0.2295 | G: 5.8088 | Base: 0.3301
Training Complete!
```

```
import matplotlib.pyplot as plt

generator.eval()

input_img, target_img = next(iter(train_loader))
input_img = input_img.to(DEVICE)

with torch.no_grad():
    fake_img = generator(input_img)

plt.figure(figsize=(12,4))

plt.subplot(1,3,1)
plt.title("Input (Edges)")
plt.imshow(input_img[0].permute(1,2,0).cpu()*0.5+0.5)

plt.subplot(1,3,2)
plt.title("Generated")
plt.imshow(fake_img[0].permute(1,2,0).cpu()*0.5+0.5)

plt.subplot(1,3,3)
plt.title("Real")
plt.imshow(target_img[0].permute(1,2,0)*0.5+0.5)

plt.show()
```



```
import os
os.makedirs("generated_outputs", exist_ok=True)
print("Folder created:", os.path.exists("generated_outputs"))
```

```
Folder created: True
```

```
import torch
import matplotlib.pyplot as plt
import os

generator.eval()

count = 0

for input_img, target_img in train_loader:
    input_img = input_img.to(DEVICE)

    with torch.no_grad():
        fake_img = generator(input_img)

    for i in range(fake_img.size(0)):
        fake = fake_img[i].permute(1,2,0).cpu()*0.5+0.5
        fake = fake.clamp(0,1)

        plt.imsave(f"generated_outputs/output_{count}.png", fake.nu
        count += 1

        if count >= 10:
            break
    if count >= 10:
        break

print("Saved files:", os.listdir("generated_outputs"))
```

```
Saved files: ['output_5.png', 'output_3.png', 'output_7.png', 'outpu
```

```
!zip -r outputs.zip generated_outputs
```

```
  adding: generated_outputs/ (stored 0%)
  adding: generated_outputs/output_5.png (deflated 0%)
  adding: generated_outputs/output_3.png (deflated 0%)
  adding: generated_outputs/output_7.png (deflated 0%)
  adding: generated_outputs/output_1.png (deflated 0%)
  adding: generated_outputs/output_8.png (deflated 0%)
  adding: generated_outputs/output_0.png (deflated 0%)
  adding: generated_outputs/output_2.png (deflated 0%)
  adding: generated_outputs/output_6.png (deflated 0%)
  adding: generated_outputs/output_9.png (deflated 0%)
  adding: generated_outputs/output_4.png (deflated 0%)
```

```
from google.colab import files
files.download("outputs.zip")
```