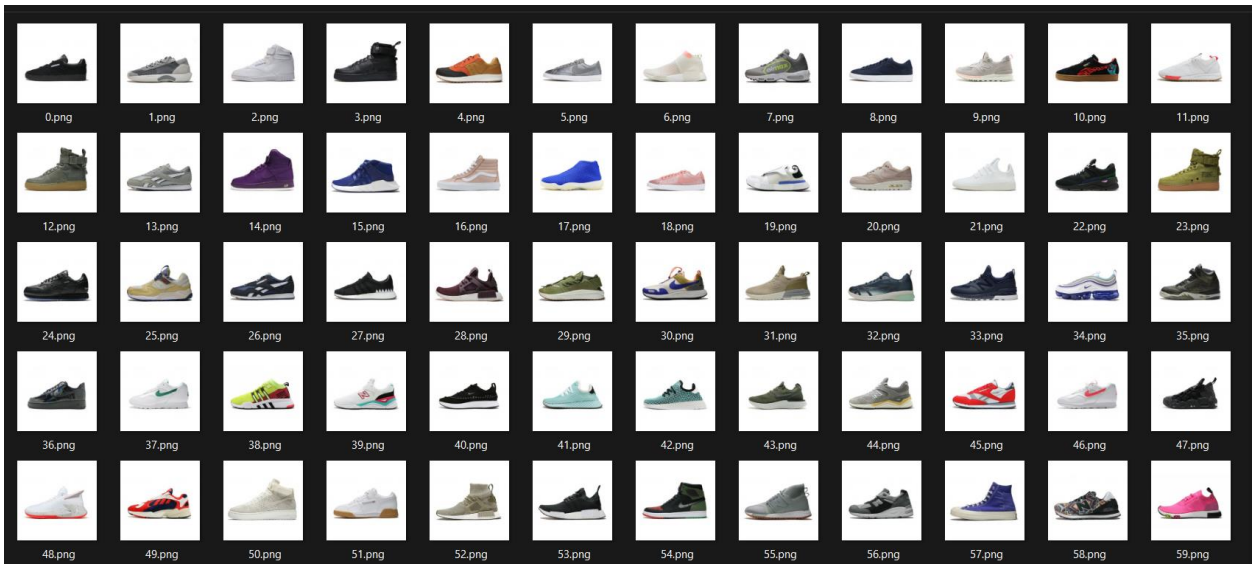
Sneaker Design Generation using Generative Adversarial Networks

1) Dataset

- Link for base dataset: https://github.com/micah5/sneaker-generator/tree/master/dataset/class1
- Part of dataset scraped from e-commerce websites, like amazon.
- Total images - 2718
- Sample –

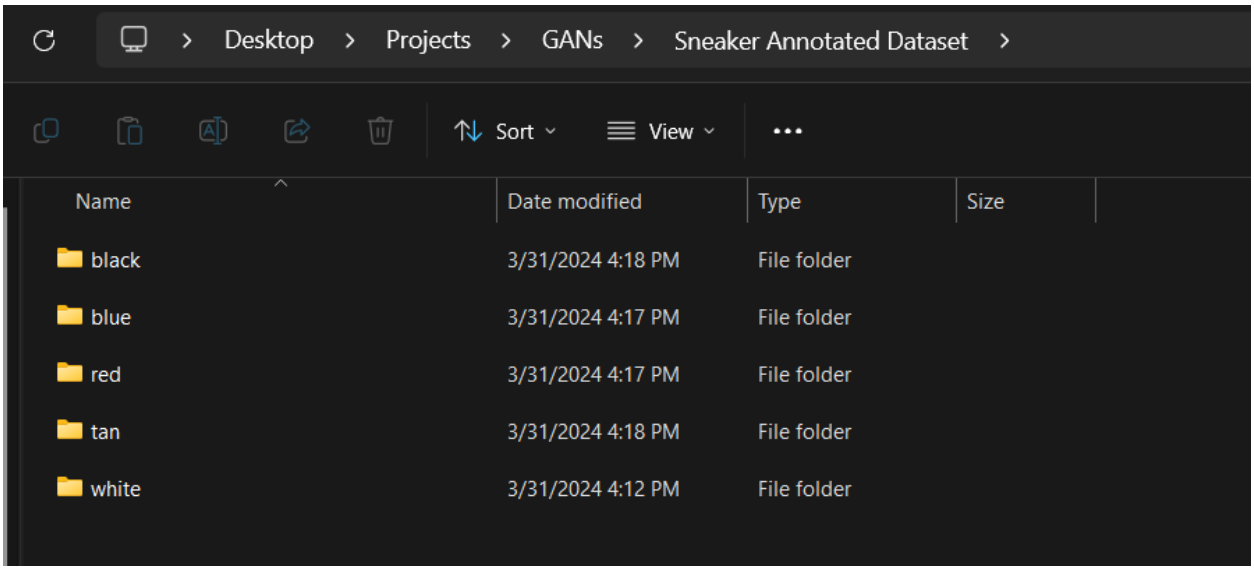


- Dataset annotated into black, white, blue, red and tan colors using following script:

```
from google.colab import drive
drive.mount('/content/drive')

!unzip "/content/drive/My Drive/CV Project/Sneaker Dataset.zip"
!mkdir "/content/dataset"
!mkdir "/content/dataset/black"
!mkdir "/content/dataset/white"
!mkdir "/content/dataset/brown"
```

```
!mkdir "/content/dataset/red"
!mkdir "/content/dataset/blue"
!mkdir "/content/dataset/yellow"


from transformers import pipeline
from PIL import Image
import os

pipe = pipeline("zero-shot-image-classification", model="openai/clip-vit-large-patch14")

for file in os.listdir("/content/Sneaker Dataset"):
    image = Image.open(os.path.join("/content/Sneaker Dataset", file))
    label = pipe(image, candidate_labels=["black shoe", "white shoe", "brown shoe", "red shoe", "blue shoe", "yellow shoe"])[0]['label']
    if label == "black shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/black", file))
    elif label == "white shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/white", file))
    elif label == "brown shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/brown", file))
    elif label == "red shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/red", file))
    elif label == "blue shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/blue", file))
    elif label == "yellow shoe":
        os.rename(os.path.join("/content/Sneaker Dataset", file), os.path.join("/content/dataset/yellow", file))


!mv "/content/dataset" "/content/drive/My Drive/dataset"
!zip -r "/content/drive/My Drive/dataset.zip" "/content/drive/My Drive/dataset"
```

- Annotated dataset file structure:



2) Training scripts

Deep Convolutional Generative Adversarial Network (DCGAN)

```
from google.colab import drive
drive.mount("/content/drive")

!unzip "/content/drive/My Drive/CV Project/Sneaker Dataset.zip"

import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Dense, Dropout,
Flatten, BatchNormalization, Input, Reshape, LeakyReLU, ReLU
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import img_to_array, load_img


def discriminator(input_shape=(128, 128, 3)):
    model = Sequential([
```

```
        Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=input_shape),
        LeakyReLU(alpha=0.2),

        Conv2D(128, (5, 5), strides=(2, 2), padding='same', use_bias=False),
        BatchNormalization(momentum=0.5),
        LeakyReLU(alpha=0.2),

        Conv2D(256, (5, 5), strides=(2, 2), padding='same', use_bias=False),
        BatchNormalization(momentum=0.5),
        LeakyReLU(alpha=0.2),

        Conv2D(512, (5, 5), strides=(2, 2), padding='same', use_bias=False),
        BatchNormalization(momentum=0.5),
        LeakyReLU(alpha=0.2),

        # Output => 8 * 8 * 512
        Flatten(),
        Dense(1, activation='sigmoid')
    ])

    opt = Adam(learning_rate=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=None)

    return model


def generator():
    model = Sequential([
        Dense(4*4*512, input_shape=(120,)),
        Reshape((4, 4, 512)),
        BatchNormalization(momentum=0.5),
        ReLU(),

        Conv2DTranspose(256, (5, 5), strides=(2, 2), padding='same',
use_bias=False),
        BatchNormalization(momentum=0.5),
        ReLU(),

        Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same',
use_bias=False),
        BatchNormalization(momentum=0.5),
        ReLU(),
```

```
        Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False),
        BatchNormalization(momentum=0.5),
        ReLU(),

        Conv2DTranspose(32, (5, 5), strides=(2, 2), padding='same',
use_bias=False),
        BatchNormalization(momentum=0.5),
        ReLU(),

        Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same',
activation='tanh', use_bias=False),
    ])

    opt = Adam(learning_rate=0.00015, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=None)

    return model


def gan(gen_model, disc_model):
    disc_model.trainable = False

    model = Sequential([
        gen_model,
        disc_model
    ])

    opt = Adam(learning_rate=0.001, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)

    return model


def load_dataset(directory="/content/Sneaker Dataset", target_size=(128, 128)):
    images = []
    for filename in os.listdir(directory):
        img = load_img(os.path.join(directory, filename),
target_size=target_size)
        images.append(img_to_array(img))

    dataset = np.array(images)
    dataset = dataset.astype('float32')
    dataset /= 127.5
    dataset -= 1.0
```

```
    return dataset

def generate_real_samples(dataset, num_samples):
    ix = np.random.randint(0, dataset.shape[0], num_samples)

    X = dataset[ix]
    y = np.zeros((num_samples, 1)) + 0.9  # One sided Label smoothing


    return X, y

def generate_latent_points(num_samples): # gen model input
    x_input = np.random.normal(0, 1, 120 * num_samples)
    x_input = x_input.reshape(num_samples, 120)

    return x_input

def generate_fake_samples(gen_model, num_samples): # gen model output
    x_input = generate_latent_points(num_samples)

    X = gen_model.predict(x_input)
    y = np.zeros((num_samples, 1))


    return X, y

def save_plot(examples, n):

    for i in range(n * n):
        plt.subplot(n, n, 1 + i)
        plt.axis('off')

        examples[i] += 1
        examples[i] *= 127.5
        plt.imshow(examples[i].astype(np.uint8), interpolation='nearest')

    plt.savefig("results.png")
    plt.show()


def train(gen_model, disc_model, gan_model, dataset, epochs=500, batch_size=64):
    num_batches_per_epoch = int(dataset.shape[0] / batch_size)
    d_loss_hist = []
    gan_loss_hist = []
```

```
    for i in range(epochs):
        for j in range(num_batches_per_epoch):
            X_real, y_real = generate_real_samples(dataset, batch_size // 2)
            X_fake, y_fake = generate_fake_samples(gen_model, batch_size // 2)

            disc_model.trainable = True
            X, y = np.vstack((X_real, X_fake)), np.vstack((y_real, y_fake))
            d_loss = disc_model.train_on_batch(X, y)
            disc_model.trainable = False

            X_gan = generate_latent_points(batch_size)
            y_gan = np.ones((batch_size, 1))
            gan_loss = gan_model.train_on_batch(X_gan, y_gan)

            print('>%d, %d/%d, d=%.3f, g=%.3f' % (i+1, j+1,
num_batches_per_epoch, d_loss, gan_loss))

        d_loss_hist.append(d_loss)
        gan_loss_hist.append(gan_loss)
        if (i+1) % 50 == 0:
            filename = 'generator_model_%03d.keras' % (i + 1)
            gen_model.save(filename)

            latent_points = generate_latent_points(25)
            y = gen_model.predict(latent_points)
            save_plot(y, 5)

            plt.plot(d_loss_hist, label='disc_loss')
            plt.plot(gan_loss_hist, label='gan_loss')
            plt.legend()
            plt.savefig("Loss Plot.png")

disc_model = discriminator()
gen_model = generator()
gan_model = gan(gen_model, disc_model)
dataset = load_dataset()

train(gen_model, disc_model, gan_model, dataset)


!mkdir -p "/content/drive/My Drive/CV Project/2nd train"

!mv "/content/generator_model_300.keras" "/content/drive/My Drive/CV Project/2nd
train/generator_model_300.keras"
```

```
!mv "/content/generator_model_450.keras" "/content/drive/My Drive/CV Project/2nd train/generator_model_450.keras"
!mv "/content/generator_model_500.keras" "/content/drive/My Drive/CV Project/2nd train/generator_model_500.keras"

!mv "/content/results.png" "/content/drive/My Drive/CV Project/2nd train/results.png"
!mv "/content/Loss Plot.png" "/content/drive/My Drive/CV Project/2nd train/Loss Plot.png"
```

Wassersteins Generative Adversarial Network with Gradient Penalty (WGAN-GP)

```
import torch
from torch import nn
import torchvision
import torchvision.transforms as tfs
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import matplotlib.pyplot as plt
from numpy import transpose, array
import torchvision.transforms.functional as TF


device = "cuda" if torch.cuda.is_available() else "cpu"
device


class Critic(nn.Module):
    def __init__(self):
        super().__init__()

        self.crit = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2),

            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=4, stride=2, padding=1, bias=False),
```

```
            nn.InstanceNorm2d(128, affine=True),
            nn.LeakyReLU(0.2),

            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.InstanceNorm2d(256, affine=True), # affine=True makes params
learnable
            nn.LeakyReLU(0.2),

            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.InstanceNorm2d(512, affine=True),
            nn.LeakyReLU(0.2),

            # Output here - 16*16*512

            nn.Conv2d(in_channels=512, out_channels=1, kernel_size=16, stride=1,
padding=0)
        )


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


class Generator(nn.Module):
    def __init__(self):
        super().__init__()

        self.gen = nn.Sequential(
           nn.ConvTranspose2d(in_channels=64, out_channels=512, kernel_size=16,
stride=1, padding=0),
           nn.BatchNorm2d(512),
           nn.ReLU(),

           nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=4,
stride=2, padding=1, bias=False),
           nn.BatchNorm2d(256),
           nn.ReLU(),

           nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=4,
stride=2, padding=1, bias=False),
           nn.BatchNorm2d(128),
           nn.ReLU(),
```

```
            nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.ConvTranspose2d(in_channels=64, out_channels=3, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.Tanh()
        )

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




def initialize_weights(model):
    # Initializes weights according to the DCGAN paper
    for m in model.modules():
        if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)):
            nn.init.normal_(m.weight.data, 0.0, 0.02) # normal distribution with
mean 0 and std dev 0.02


def gradient_penalty(critic, real, fake, device="cpu"):
    BATCH_SIZE, C, H, W = real.shape
    alpha = torch.rand((BATCH_SIZE, 1, 1, 1)).repeat(1, C, H, W).to(device)
    interpolated_images = real * alpha + fake * (1 - alpha)

    # Calculate critic scores
    mixed_scores = critic(interpolated_images)

    # Take the gradient of the scores with respect to the images
    gradient = torch.autograd.grad(
        inputs=interpolated_images,
        outputs=mixed_scores,
        grad_outputs=torch.ones_like(mixed_scores),
        create_graph=True,
        retain_graph=True,
    )[0]
    gradient = gradient.view(gradient.shape[0], -1)
    gradient_norm = gradient.norm(2, dim=1) # L2 norm
    gradient_penalty = torch.mean((gradient_norm - 1) ** 2)

    return gradient_penalty
```

```
def plot_generated_images():
    noise = torch.randn(25, 64, 1, 1).to(device)
    fake = gen(noise)

    fake_images = [(TF.to_pil_image((img * 0.5) + 0.5)) for img in fake]

    print(array(fake_images[5]))
    # Plot the images
    for i in range(25):
      plt.subplot(5, 5, 1 + i)
      plt.axis('off')

      plt.imshow(fake_images[i])

    plt.savefig(f"/kaggle/working/results_{epoch+1}.png")
    plt.show()



# gen = torch.load("../input/model-100-pt/model_100.pt").to(device) # resume
training

transform = tfs.Compose(
    [
        tfs.Resize(256),
        tfs.ToTensor(),
        tfs.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
        # mean and std of 0.5 normalizes between [-1, 1]. applied to all 3
channels
    ]
)

# Load dataset
dataset = ImageFolder(root="../input/sneaker-dataset", transform=transform)
loader = DataLoader(dataset, batch_size=64, shuffle=True)
print("Folder loaded")

# Initialize gen and crit
gen = Generator().to(device)
crit = Critic().to(device)
initialize_weights(gen)
initialize_weights(crit)

# Initialize optimizer
```

```
gen_opt = torch.optim.Adam(gen.parameters(), lr=0.0001, betas=(0, 0.9))
crit_opt = torch.optim.Adam(crit.parameters(), lr=0.0001, betas=(0, 0.9))

gen.train()
crit.train()

print("Training started")
for epoch in range(0, 300):
    for i, (images, _) in enumerate(loader):
        real = images.to(device)
        batch_size = real.size(0)

        # Train Critic: max E[critic(real)] - E[critic(fake)]
        for _ in range(5):
            noise = torch.randn(batch_size, 64, 1, 1).to(device)
            fake = gen(noise)

            critic_real = crit(real).reshape(-1)
            critic_fake = crit(fake).reshape(-1) # reshape(-1) will flatten the
1x1 conv2d output
            gp = gradient_penalty(crit, real, fake, device)

            critic_loss = (
                -(torch.mean(critic_real) - torch.mean(critic_fake)) + 10 * gp
            )
            crit.zero_grad()
            critic_loss.backward(retain_graph=True)
            crit_opt.step()

        # Train Generator: max E[critic(gen_fake)] <-> min -E[critic(gen_fake)]
        gen_fake = crit(fake).reshape(-1)
        gen_loss = -torch.mean(gen_fake)

        gen.zero_grad()
        gen_loss.backward()
        gen_opt.step()

    print(f"Epoch [{epoch}/{300}] Loss D: {critic_loss:.4f}, loss G:
{gen_loss:.4f}")

    if (epoch + 1) % 25 == 0:
      plot_generated_images()
      torch.save(gen, f"/kaggle/working/model_{epoch+1}.pt")
```

Conditional Generative Adversarial Network (cGAN)

```
from google.colab import drive
drive.mount('/content/drive')

!unzip "/content/drive/My Drive/Sneaker Annotated Dataset.zip"

import torch
from torch import nn
import torchvision
import torchvision.transforms as tfs
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import matplotlib.pyplot as plt
from numpy import transpose, array
import torchvision.transforms.functional as TF

z_dim = 100
img_dim = 256
num_classes = 5
lr = 0.0002
beta1 = 0.5
beta2 = 0.999
weight_decay = 1e-5
num_epochs = 300

device = "cuda" if torch.cuda.is_available() else "cpu"
device

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_embedding = nn.Embedding(num_classes, img_dim*img_dim)

        self.disc = nn.Sequential(
            nn.Conv2d(in_channels=3+1, out_channels=64, kernel_size=4, stride=2,
padding=1),
            nn.LeakyReLU(0.2),

            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
```

```
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.InstanceNorm2d(256),
            nn.LeakyReLU(0.2),

            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.InstanceNorm2d(512),
            nn.LeakyReLU(0.2),

            # Shape here - 16*16*512

            nn.Conv2d(in_channels=512, out_channels=1, kernel_size=16, stride=1,
padding=0),
            # nn.Sigmoid()
        )

    def forward(self, x, label):
        label_embed = self.label_embedding(label)
        label_embed = label_embed.view(-1, 1, img_dim, img_dim)
        x = torch.cat((x, label_embed), dim=1)

        return self.disc(x)

class Generator(nn.Module):
    def __init__(self, label_embed_size=10):
        super().__init__()

        self.label_embedding = nn.Embedding(num_classes, label_embed_size)

        self.gen = nn.Sequential(
            nn.ConvTranspose2d(in_channels=z_dim+label_embed_size,
out_channels=512, kernel_size=16, stride=1, padding=0),
            nn.BatchNorm2d(512),
            nn.ReLU(),

            nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(),

            nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(128),
```

```
            nn.ReLU(),

            nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.ConvTranspose2d(in_channels=64, out_channels=3, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.Tanh()
        )

    def forward(self, x, label):
        label_embed = self.label_embedding(label)
        label_embed= label_embed.view(label_embed.shape[0], -1, 1, 1) #
(batch_size, z_dim, 1, 1), concat along dim=1
        x = torch.cat((x, label_embed), dim=1)

        return self.gen(x)

def initialize_weights(model):
    # Initializes weights according to the DCGAN paper
    for m in model.modules():
        if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)):
            nn.init.normal_(m.weight.data, 0.0, 0.02) # normal distribution with
mean 0 and std dev 0.02

def plot_generated_images():
    noise = torch.randn(25, z_dim, 1, 1).to(device)
    labels = torch.randint(0, num_classes, (25,), device=device)
    print(labels)
    fake = gen(noise, labels)

    # Convert tensor to PIL images and renormalize
    fake_images = [(TF.to_pil_image((img * 0.5) + 0.5)) for img in fake]

    # Plot the images
    for i in range(25):
      plt.subplot(5, 5, 1 + i)
      plt.axis('off')

      plt.imshow(fake_images[i])

    plt.savefig(f"/content/drive/My Drive/Sneaker cGAN/results_{epoch+1}.png")
    plt.show()
```

```
transform = tfs.Compose(
    [
        tfs.Resize(img_dim),
        tfs.ToTensor(),
        tfs.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
        # mean and std of 0.5 normalizes between [-1, 1]. applied to all 3
channels
    ]
)

# Load dataset
dataset = ImageFolder(root="/content/Sneaker Annotated Dataset",
transform=transform)
loader = DataLoader(dataset, batch_size=256, shuffle=True)
print("Folder loaded")

# Initialize gen and disc
gen = Generator(label_embed_size=10).to(device)
disc = Discriminator().to(device)
initialize_weights(gen)
initialize_weights(disc)

# Initialize optimizer
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr, betas=(beta1, beta2),
weight_decay=weight_decay)
disc_opt = torch.optim.Adam(disc.parameters(), lr=lr, betas=(beta1, beta2),
weight_decay=weight_decay)

# Loss functions
loss_fn = nn.BCEWithLogitsLoss() # combination of sigmoid and BCE loss, more
numerically stable

gen.train()
disc.train()

print("Training started")

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(loader):
        real = images.to(device)
        batch_size = real.size(0)

        noise = torch.randn(batch_size, z_dim, 1, 1).to(device)
        fake = gen(noise, labels.to(device))
```

```
        disc_real = disc(real.detach(), labels.to(device))
        disc_fake = disc(fake.detach(), labels.to(device))
        disc_loss = (loss_fn(disc_real, torch.ones_like(disc_real)) +
loss_fn(disc_fake, torch.zeros_like(disc_fake))) / 2

        disc_opt.zero_grad()
        disc_loss.backward()
        disc_opt.step()

        disc_fake = disc(fake, labels.to(device))
        gen_loss = loss_fn(disc_fake, torch.ones_like(disc_fake))

        gen_opt.zero_grad()
        gen_loss.backward()
        gen_opt.step()

    print(f"Epoch [{epoch+1}/{num_epochs}] Loss D: {disc_loss:.4f} Loss G:
{gen_loss:.4f}")

    if (epoch + 1) % 50 == 0:
        plot_generated_images()
        torch.save(gen, f"/content/drive/My Drive/Sneaker
cGAN/model_{epoch+1}.pt")
```

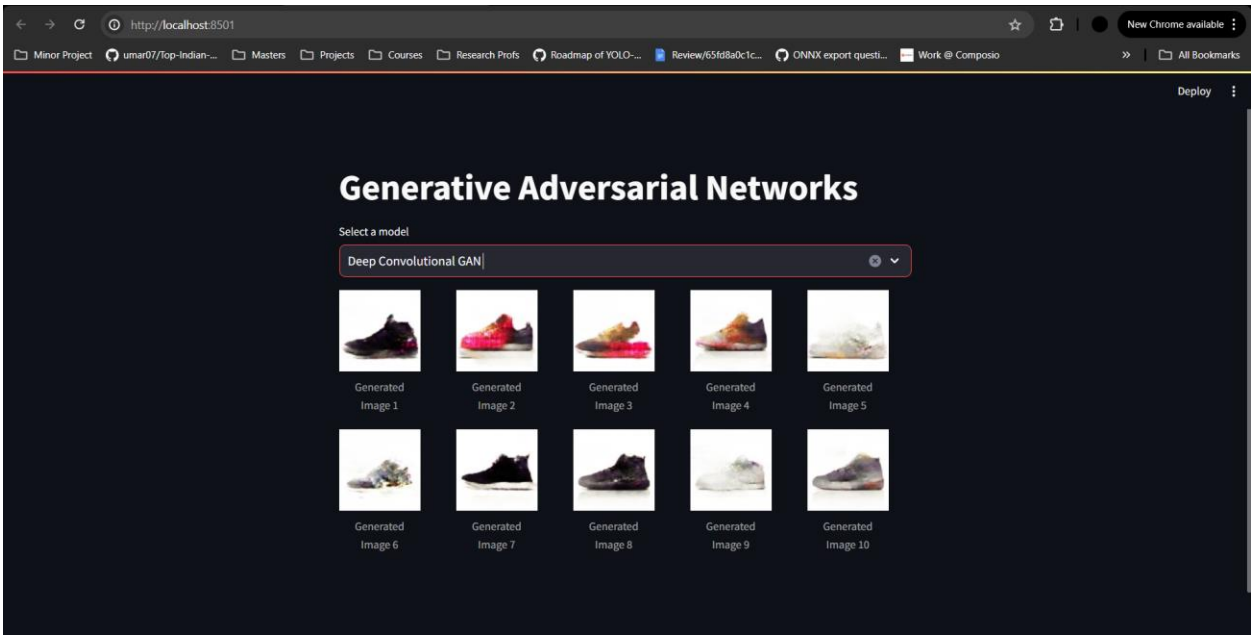
Streamlit Deployment Script

```
import streamlit as st

st.title("Generative Adversarial Networks")

model_option = st.selectbox(
    label="Select a model",
    options=["Deep Convolutional GAN", "Wasserstein GAN with GP", "Conditional
GAN"],
    index=None,
    placeholder="Select a model..."
)

if model_option == "Conditional GAN":
    from test.cgan import Generator
```

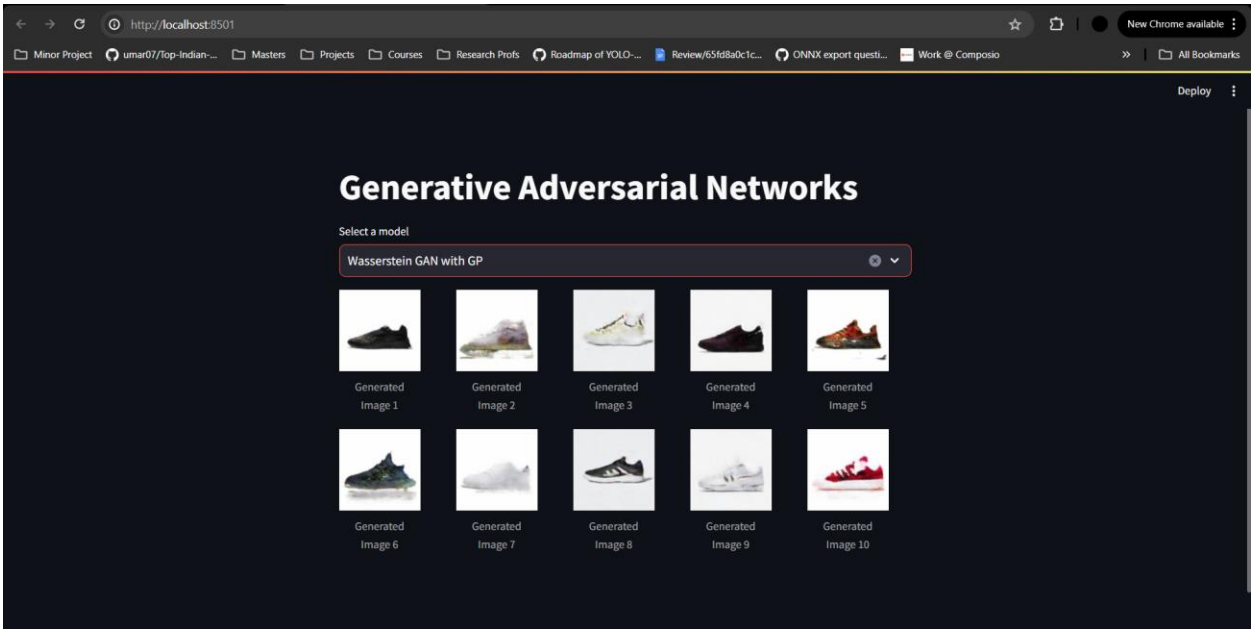
```
    from test.cgan import plot_generated_images

    label_option = st.selectbox(
        label="Select a color",
        options=["Black", "Blue", "Red", "Yellow", "White"],
        index=None,
        placeholder="Select a color.."
    )

    label_mapping = {
        "Black": 0,
        "Blue": 1,
        "Red": 2,
        "Yellow": 3,
        "White": 4
    }

    container = st.container()
    cols = container.columns(5)

    if label_option:
        for i in range(10):
            img = plot_generated_images(label_mapping[label_option])
            cols[i % 5].image(img, caption=f"Generated Image {i + 1}", width=100)

elif model_option == "Wasserstein GAN with GP":
    from test.wgangp import Generator
    from test.wgangp import plot_generated_images

    container = st.container()
    cols = container.columns(5)

    for i in range(10):
        img = plot_generated_images()
        cols[i % 5].image(img, caption=f"Generated Image {i + 1}", width=100)

elif model_option == "Deep Convolutional GAN":
    from test.dcgan import generate_single_image

    container = st.container()
    cols = container.columns(5)

    for i in range(10):
        img = generate_single_image()
        cols[i % 5].image(img, caption=f"Generated Image {i + 1}", width=100)
```

3) Output

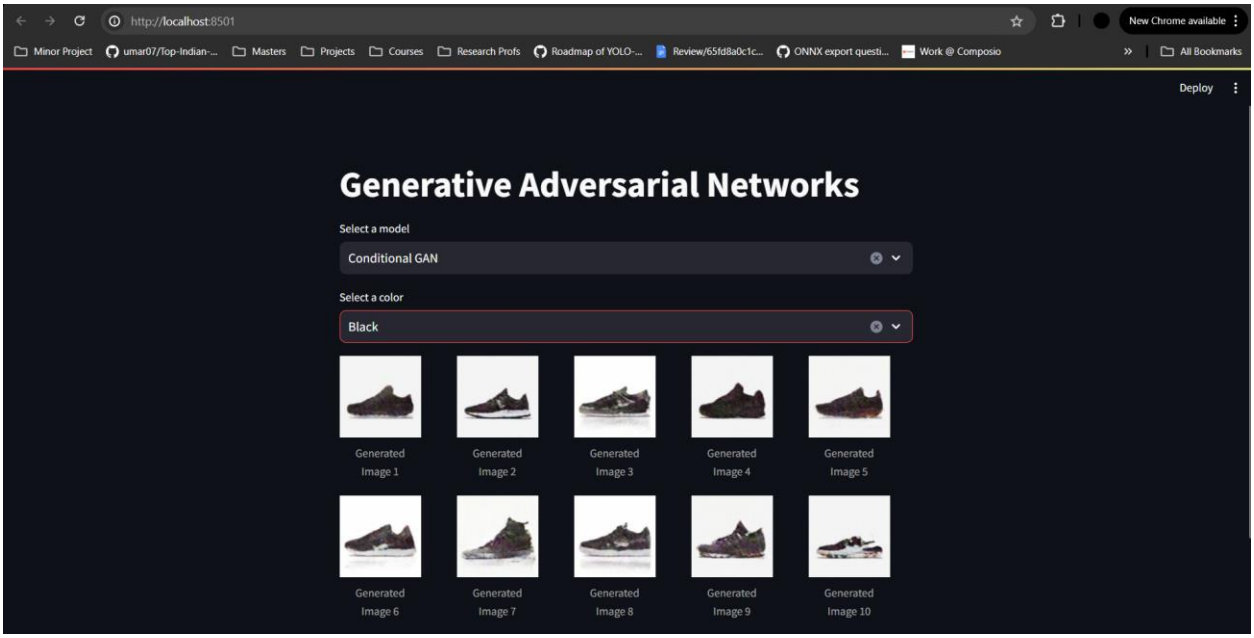
DCGAN

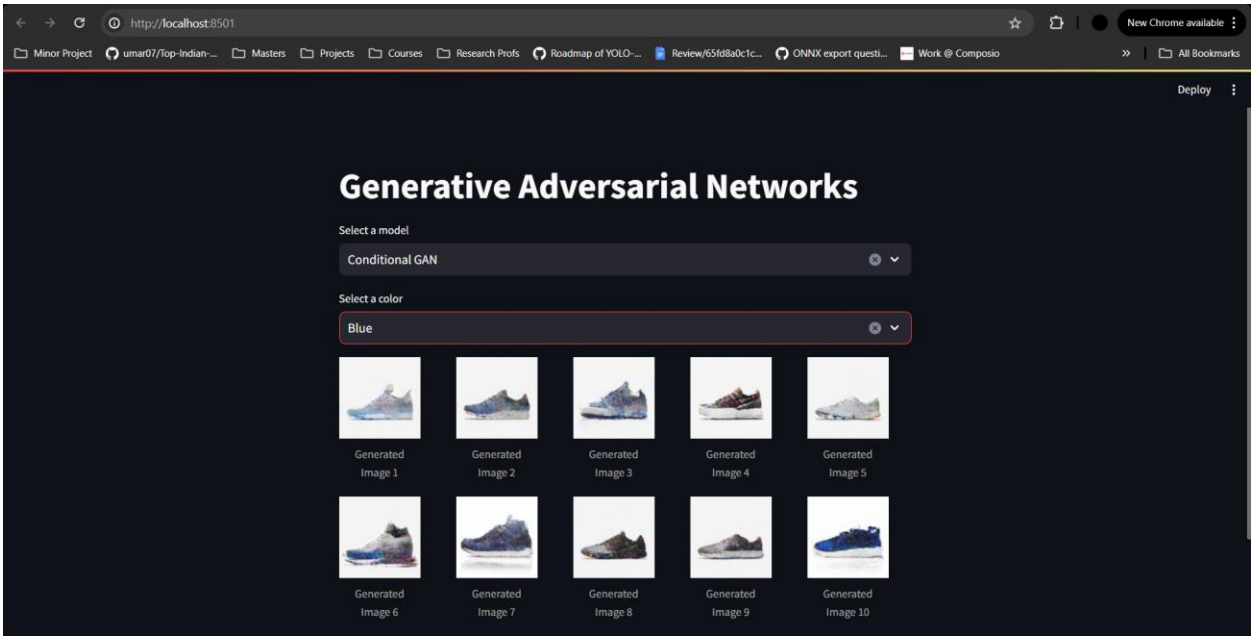


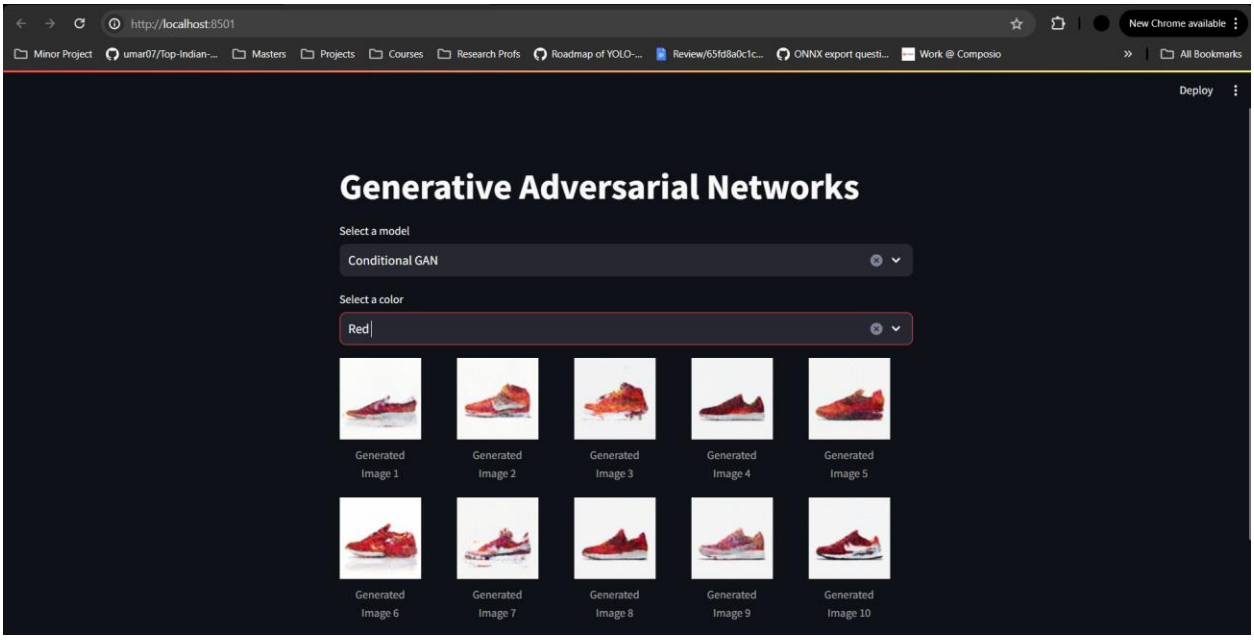
WGAN-GP

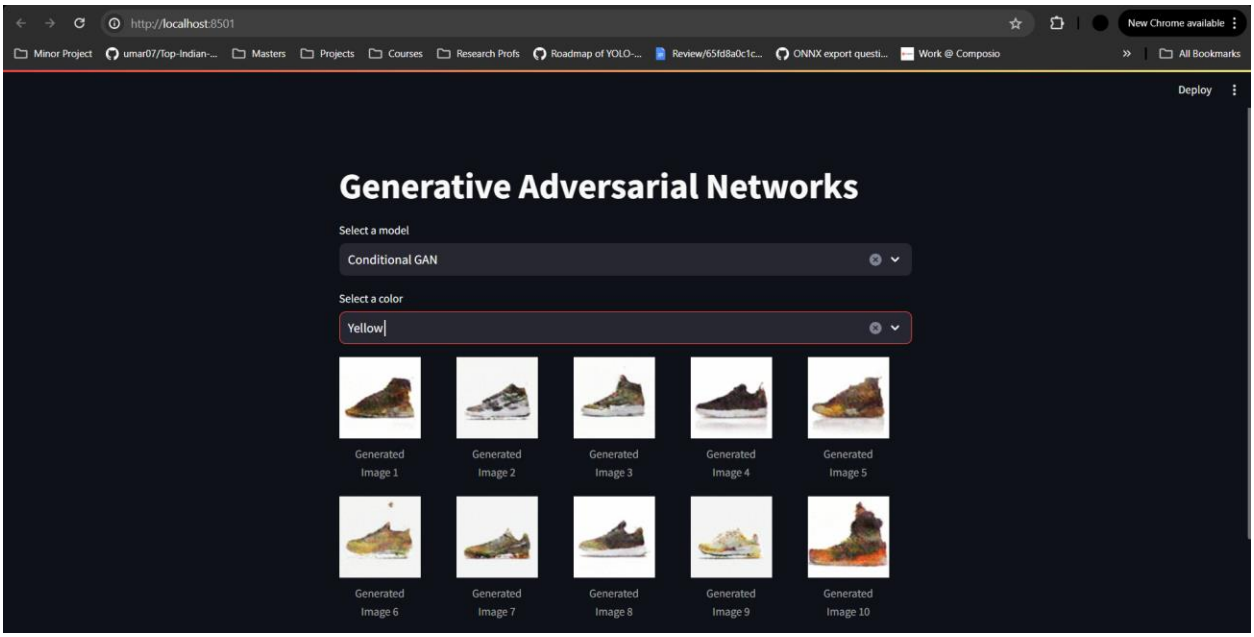

http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Wasserstein GAN with GP
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10

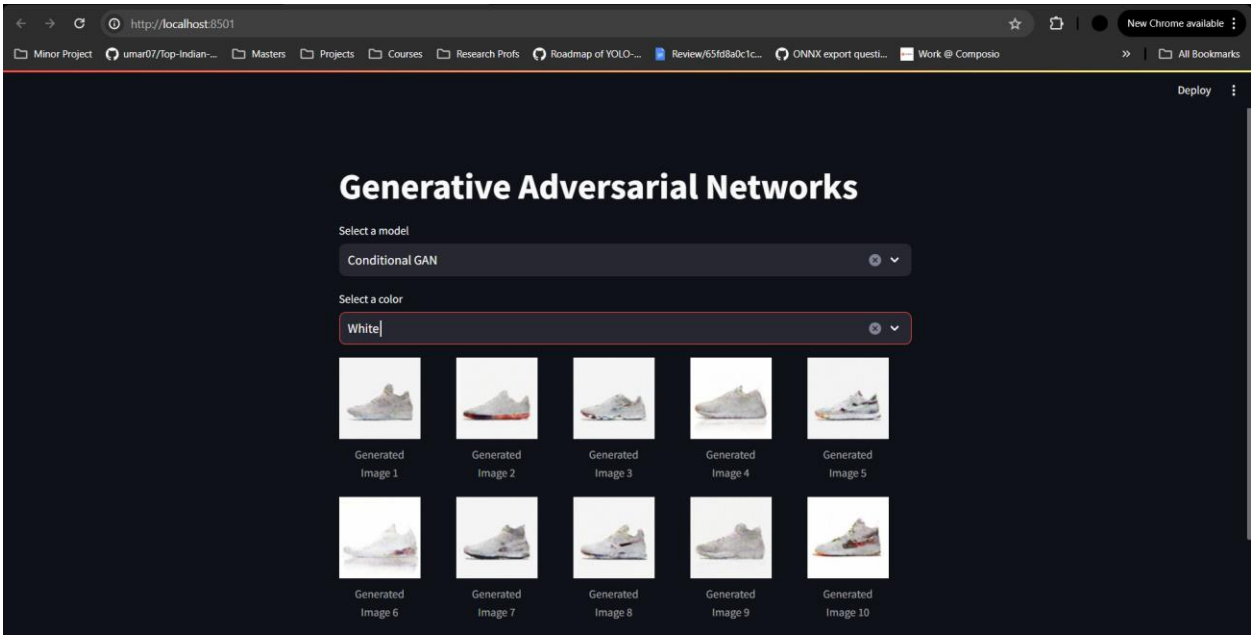
cGAN


http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Conditional GAN
Select a color
Black
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10


http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Conditional GAN
Select a color
Blue
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10


http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Conditional GAN
Select a color
Red
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10


http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Conditional GAN
Select a color
Yellow
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10


http://localhost:8501
New Chrome available
Minor Project
umar07/Top-Indian-...
Masters
Projects
Courses
Research Profs
Roadmap of YOLO-...
Review/65fd8a0c1c...
ONNX export questi...
Work @ Composio
All Bookmarks
Deploy
Generative Adversarial Networks
Select a model
Conditional GAN
Select a color
White
Generated Image 1
Generated Image 2
Generated Image 3
Generated Image 4
Generated Image 5
Generated Image 6
Generated Image 7
Generated Image 8
Generated Image 9
Generated Image 10