```
!ls
```

```
drive  sample_data
```

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

```
Mounted at /content/drive
```

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, utils
import matplotlib.pyplot as plt
import pandas as pd
from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cuda
```

```
OUTPUT_DIR = "/content/drive/MyDrive/flower_results"

vae_img_dir = OUTPUT_DIR + "/vae_images"
vaegan_img_dir = OUTPUT_DIR + "/vaegan_images"
model_dir = OUTPUT_DIR + "/models"
plot_dir = OUTPUT_DIR + "/plots"

for d in [vae_img_dir, vaegan_img_dir, model_dir, plot_dir]:
    os.makedirs(d, exist_ok=True)

print("Folders created in Drive")
```

```
Folders created in Drive
```

```
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.CenterCrop(64),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

dataset = datasets.Flowers102(
    root="/content/data",
    split="train",
```

```
    download=True,
    transform=transform
)

loader = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=2

print("Total images:", len(dataset))
print("Batches per epoch:", len(loader))
```

```
100%|██████████| 345M/345M [00:12<00:00, 27.0MB/s]
100%|██████████| 502/502 [00:00<00:00, 2.05MB/s]
100%|██████████| 15.0k/15.0k [00:00<00:00, 34.7MB/s]Total images: 1020
Batches per epoch: 32
```

```
images, _ = next(iter(loader))
grid = utils.make_grid(images[:16], normalize=True)
plt.figure(figsize=(6,6))
plt.imshow(grid.permute(1,2,0))
plt.axis("off")
plt.show()
```

```
latent_dim = 128

class Encoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3,64,4,2,1), nn.ReLU(),
            nn.Conv2d(64,128,4,2,1), nn.ReLU(),
            nn.Conv2d(128,256,4,2,1), nn.ReLU(),
            nn.Conv2d(256,512,4,2,1), nn.ReLU()
        )
        self.fc_mu = nn.Linear(512*4*4, latent_dim)
        self.fc_logvar = nn.Linear(512*4*4, latent_dim)

    def forward(self,x):
        x = self.conv(x).view(x.size(0), -1)
        return self.fc_mu(x), self.fc_logvar(x)


class Decoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(latent_dim, 512*4*4)
```

```
            self.deconv = nn.Sequential(
                nn.ConvTranspose2d(512,256,4,2,1), nn.ReLU(),
                nn.ConvTranspose2d(256,128,4,2,1), nn.ReLU(),
                nn.ConvTranspose2d(128,64,4,2,1), nn.ReLU(),
                nn.ConvTranspose2d(64,3,4,2,1), nn.Tanh()
            )

        def forward(self,z):
            x = self.fc(z).view(-1,512,4,4)
            return self.deconv(x)

    encoder = Encoder().to(device)
    decoder = Decoder().to(device)
```

```
    import torch.nn.functional as F

    def vae_loss(recon, x, mu, logvar):
        recon_loss = F.mse_loss(recon, x)
        kl_loss = -0.5 * torch.mean(1 + logvar - mu.pow(2) - logvar.exp())
        return recon_loss, kl_loss
```

```
    optimizer = optim.Adam(list(encoder.parameters()) + list(decoder.param

    epochs = 50

    vae_recon_losses = []
    vae_kl_losses = []

    for epoch in range(epochs):
        recon_epoch = 0
        kl_epoch = 0

        for imgs,_ in tqdm(loader):
            imgs = imgs.to(device)

            mu, logvar = encoder(imgs)
            std = torch.exp(0.5*logvar)
            z = mu + std*torch.randn_like(std)

            recon = decoder(z)

            recon_loss, kl_loss = vae_loss(recon, imgs, mu, logvar)
            loss = recon_loss + kl_loss

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            recon_epoch += recon_loss.item()
            kl_epoch += kl_loss.item()

        recon_epoch /= len(loader)
        kl_epoch /= len(loader)
```

```
    vae_recon_losses.append(recon_epoch)
    vae_kl_losses.append(kl_epoch)

    utils.save_image(recon[:16], f"{vae_img_dir}/recon_epoch_{epoch}.pn
    utils.save_image(decoder(torch.randn(16,latent_dim).to(device)),
                     f"{vae_img_dir}/sample_epoch_{epoch}.png", normal

    torch.save({"encoder":encoder.state_dict(),"decoder":decoder.state_
                f"{model_dir}/vae_epoch_{epoch}.pt")

    print(f"Epoch {epoch}: Recon={recon_epoch:.4f}, KL={kl_epoch:.4f}"
```

```
100%|██████████| 32/32 [00:03<00:00,  9.50it/s]
Epoch 21: Recon=0.1948, KL=0.0259
100%|██████████| 32/32 [00:05<00:00,  6.25it/s]
Epoch 22: Recon=0.1936, KL=0.0260
```

```
100%|██████████| 32/32 [00:03<00:00,  8.93it/s]
Epoch 44: Recon=0.1849, KL=0.0273
100%|██████████| 32/32 [00:03<00:00,  9.45it/s]
Epoch 45: Recon=0.1839, KL=0.0280
100%|██████████| 32/32 [00:04<00:00,  7.49it/s]
Epoch 46: Recon=0.1834, KL=0.0282
100%|██████████| 32/32 [00:04<00:00,  7.23it/s]
Epoch 47: Recon=0.1830, KL=0.0276
100%|██████████| 32/32 [00:03<00:00,  9.20it/s]
Epoch 48: Recon=0.1819, KL=0.0269
100%|██████████| 32/32 [00:03<00:00,  9.41it/s]
Epoch 49: Recon=0.1815, KL=0.0287
```

```
pd.DataFrame({
    "Reconstruction Loss": vae_recon_losses,
    "KL Loss": vae_kl_losses
}).to_csv(OUTPUT_DIR+"/vae_losses.csv", index=False)
```

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3,64,4,2,1), nn.LeakyReLU(0.2),
            nn.Conv2d(64,128,4,2,1), nn.LeakyReLU(0.2),
            nn.Conv2d(128,256,4,2,1), nn.LeakyReLU(0.2),
            nn.Conv2d(256,1,4,1,0), nn.Sigmoid()
        )

    def forward(self,x):
        return self.net(x).view(-1)

disc = Discriminator().to(device)
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
import os

GAN_OUTPUT_DIR = "/content/drive/MyDrive/flower_gan_results"

gan_img_dir = os.path.join(GAN_OUTPUT_DIR, "images")
gan_model_dir = os.path.join(GAN_OUTPUT_DIR, "models")

os.makedirs(gan_img_dir, exist_ok=True)
os.makedirs(gan_model_dir, exist_ok=True)

print("New GAN folders created:")
print(gan_img_dir)
print(gan_model_dir)

# verify
```

```
os.listdir(GAN_OUTPUT_DIR)
```

```
New GAN folders created:
/content/drive/MyDrive/flower_gan_results/images
/content/drive/MyDrive/flower_gan_results/models
['images', 'models']
```

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
from torchvision import utils

# ===== NEW GAN OUTPUT FOLDER =====
GAN_OUTPUT_DIR = "/content/drive/MyDrive/flower_gan_results"
gan_img_dir = os.path.join(GAN_OUTPUT_DIR, "images")
gan_model_dir = os.path.join(GAN_OUTPUT_DIR, "models")

os.makedirs(gan_img_dir, exist_ok=True)
os.makedirs(gan_model_dir, exist_ok=True)

print("GAN folders ready:", gan_img_dir, gan_model_dir)

# ===== Hyperparameters =====
epochs = 100

lambda_recon = 10.0
lambda_kl = 0.1
lambda_adv = 1.0

# ===== Optimizers =====
opt_G = optim.Adam(list(encoder.parameters()) + list(decoder.parameter
opt_D = optim.Adam(disc.parameters(), lr=5e-5)

bce = nn.BCELoss()

gan_losses = []
recon_losses2 = []
kl_losses2 = []

# ===== Training Loop =====
for epoch in range(epochs):
    g_epoch = 0.0
    d_epoch = 0.0
    recon_epoch = 0.0
    kl_epoch = 0.0

    for imgs, _ in tqdm(loader):
        imgs = imgs.to(device)

        # ---- Encode & Decode ----
        mu, logvar = encoder(imgs)
        std = torch.exp(0.5 * logvar)
        z = mu + std * torch.randn_like(std)
```

```
            fake = decoder(z)

            # ---- Train Discriminator ----
            d_real = disc(imgs)
            d_fake = disc(fake.detach())

            # label smoothing
            real_labels = 0.9 * torch.ones_like(d_real)
            fake_labels = 0.1 * torch.zeros_like(d_fake)

            d_loss = bce(d_real, real_labels) + bce(d_fake, fake_labels)

            opt_D.zero_grad()
            d_loss.backward()
            opt_D.step()

            # ---- Train Generator (VAE-GAN) ----
            d_fake = disc(fake)
            adv_loss = bce(d_fake, torch.ones_like(d_fake))

            recon_loss, kl_loss = vae_loss(fake, imgs, mu, logvar)

            g_loss = (
                lambda_recon * recon_loss +
                lambda_kl * kl_loss +
                lambda_adv * adv_loss
            )

            opt_G.zero_grad()
            g_loss.backward()
            opt_G.step()

            # ---- Accumulate losses ----
            g_epoch += g_loss.item()
            d_epoch += d_loss.item()
            recon_epoch += recon_loss.item()
            kl_epoch += kl_loss.item()

        # ---- Average losses ----
        g_epoch /= len(loader)
        d_epoch /= len(loader)
        recon_epoch /= len(loader)
        kl_epoch /= len(loader)

        gan_losses.append(g_epoch)
        recon_losses2.append(recon_epoch)
        kl_losses2.append(kl_epoch)

        # ---- Save image safely ----
        img_path = os.path.join(gan_img_dir, f"epoch_{epoch+1}.png")
        utils.save_image(fake[:16], img_path, normalize=True)

        # ---- Save model safely ----
        model_path = os.path.join(gan_model_dir, f"vaegan_epoch_{epoch+1}.
        torch.save({
            "encoder": encoder.state_dict(),
```

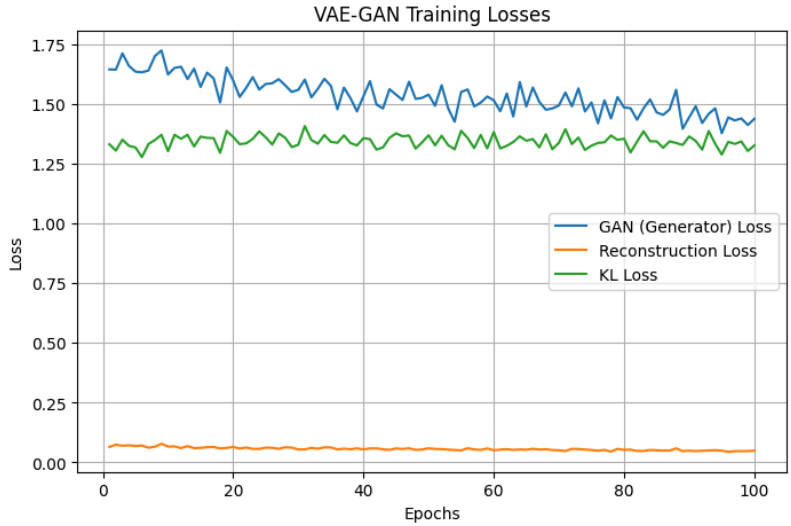
```
            "decoder": decoder.state_dict(),
            "discriminator": disc.state_dict()
        }, model_path)

        # ---- Print log ----
        print(f"Epoch [{epoch+1}/{epochs}] | "
              f"G: {g_epoch:.4f} | "
              f"D: {d_epoch:.4f} | "
              f"Recon: {recon_epoch:.4f} | "
              f"KL: {kl_epoch:.4f}")
```

```
100%|██████████| 32/32 [00:03<00:00,  9.33it/s]
Epoch [97/100] | G: 1.4328 | D: 1.3751 | Recon: 0.0469 | KL: 1.3343
100%|██████████| 32/32 [00:03<00:00,  8.48it/s]
Epoch [98/100] | G: 1.4410 | D: 1.3521 | Recon: 0.0470 | KL: 1.3447
100%|██████████| 32/32 [00:05<00:00,  6.11it/s]
Epoch [99/100] | G: 1.4135 | D: 1.3665 | Recon: 0.0476 | KL: 1.3048
100%|██████████| 32/32 [00:03<00:00,  9.11it/s]Epoch [100/100] | G: 1.
```

```
import pandas as pd

df_gan = pd.DataFrame({
    "GAN Loss": gan_losses,
    "Reconstruction Loss": recon_losses2,
    "KL Loss": kl_losses2
})

csv_path = "/content/drive/MyDrive/flower_gan_results/gan_losses.csv"
df_gan.to_csv(csv_path, index=False)

print("Losses saved to:", csv_path)
```

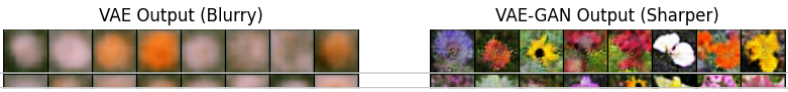
```
Losses saved to: /content/drive/MyDrive/flower_gan_results/gan_losses.c
```

```
import matplotlib.pyplot as plt

epochs_range = range(1, len(gan_losses)+1)

plt.figure(figsize=(8,5))
plt.plot(epochs_range, gan_losses, label="GAN (Generator) Loss")
plt.plot(epochs_range, recon_losses2, label="Reconstruction Loss")
plt.plot(epochs_range, kl_losses2, label="KL Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("VAE-GAN Training Losses")
plt.legend()
plt.grid(True)

plot_path = "/content/drive/MyDrive/flower_gan_results/loss_plot.png"
plt.savefig(plot_path)
plt.show()

print("Plot saved to:", plot_path)
```




```
Plot saved to: /content/drive/MyDrive/flower_gan_results/loss_plot.png
```

```
from PIL import Image
import matplotlib.pyplot as plt
import os

vae_img_path = "/content/drive/MyDrive/flower_results/vae_images/recon_e
gan_img_path = "/content/drive/MyDrive/flower_gan_results/images/epoch_9

vae_img = Image.open(vae_img_path)
gan_img = Image.open(gan_img_path)

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
plt.imshow(vae_img)
plt.title("VAE Output (Blurry)")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(gan_img)
plt.title("VAE-GAN Output (Sharper)")
plt.axis("off")

plt.show()
```



Start coding or generate with AI.

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
from torchvision import utils

# ===== Paths =====
GAN_OUTPUT_DIR = "/content/drive/MyDrive/flower_gan_results"
gan_img_dir = os.path.join(GAN_OUTPUT_DIR, "images")
gan_model_dir = os.path.join(GAN_OUTPUT_DIR, "models")

os.makedirs(gan_img_dir, exist_ok=True)
os.makedirs(gan_model_dir, exist_ok=True)

# ===== Load last checkpoint =====
last_epoch = 100   # change this to your last trained epoch number
checkpoint_path = f"{gan_model_dir}/vaegan_epoch_{last_epoch}.pt"

ckpt = torch.load(checkpoint_path, map_location=device)

encoder.load_state_dict(ckpt["encoder"])
decoder.load_state_dict(ckpt["decoder"])
disc.load_state_dict(ckpt["discriminator"])

print("Checkpoint loaded from epoch", last_epoch)

# ===== Hyperparameters =====
more_epochs = 30
lambda_recon = 10.0
lambda_kl = 0.1
lambda_adv = 1.0

# ===== Optimizers =====
opt_G = optim.Adam(list(encoder.parameters()) + list(decoder.parameters(
opt_D = optim.Adam(disc.parameters(), lr=5e-5)

bce = nn.BCELoss()

# ===== Resume Training =====
for epoch in range(last_epoch, last_epoch + more_epochs):
    g_epoch = 0.0
    d_epoch = 0.0
    recon_epoch = 0.0
    kl_epoch = 0.0

    for imgs, _ in tqdm(loader):
        imgs = imgs.to(device)

        # ---- Encode & Decode ----
```

```
            # ---- Encode & Decode ----
            mu, logvar = encoder(imgs)
            std = torch.exp(0.5 * logvar)
            z = mu + std * torch.randn_like(std)
            fake = decoder(z)

            # ---- Train Discriminator ----
            d_real = disc(imgs)
            d_fake = disc(fake.detach())

            real_labels = 0.9 * torch.ones_like(d_real)
            fake_labels = 0.1 * torch.zeros_like(d_fake)

            d_loss = bce(d_real, real_labels) + bce(d_fake, fake_labels)

            opt_D.zero_grad()
            d_loss.backward()
            opt_D.step()

            # ---- Train Generator ----
            d_fake = disc(fake)
            adv_loss = bce(d_fake, torch.ones_like(d_fake))

            recon_loss, kl_loss = vae_loss(fake, imgs, mu, logvar)

            g_loss = (
                lambda_recon * recon_loss +
                lambda_kl * kl_loss +
                lambda_adv * adv_loss
            )

            opt_G.zero_grad()
            g_loss.backward()
            opt_G.step()

            g_epoch += g_loss.item()
            d_epoch += d_loss.item()
            recon_epoch += recon_loss.item()
            kl_epoch += kl_loss.item()

        # ---- Average losses ----
        g_epoch /= len(loader)
        d_epoch /= len(loader)
        recon_epoch /= len(loader)
        kl_epoch /= len(loader)

        # ---- Save image ----
        img_path = os.path.join(gan_img_dir, f"epoch_{epoch+1}.png")
        utils.save_image(fake[:16], img_path, normalize=True)

        # ---- Save checkpoint ----
        model_path = os.path.join(gan_model_dir, f"vaegan_epoch_{epoch+1}.pt
        torch.save({
            "encoder": encoder.state_dict(),
            "decoder": decoder.state_dict(),
            "discriminator": disc.state_dict()
        }, model_path)
```

```
        print(f"Epoch [{epoch+1}] | "
              f"G: {g_epoch:.4f} | "
              f"D: {d_epoch:.4f} | "
              f"Recon: {recon_epoch:.4f} | "
              f"KL: {kl_epoch:.4f}")
```

```
Checkpoint loaded from epoch 100
100%|██████████| 32/32 [00:05<00:00,  5.74it/s]
Epoch [101] | G: 1.4217 | D: 1.3593 | Recon: 0.0470 | KL: 1.2944
100%|██████████| 32/32 [00:03<00:00,  9.16it/s]
Epoch [102] | G: 1.4213 | D: 1.3770 | Recon: 0.0476 | KL: 1.3254
100%|██████████| 32/32 [00:03<00:00,  9.28it/s]
Epoch [103] | G: 1.5104 | D: 1.3136 | Recon: 0.0473 | KL: 1.3448
100%|██████████| 32/32 [00:05<00:00,  5.87it/s]
Epoch [104] | G: 1.4475 | D: 1.3530 | Recon: 0.0475 | KL: 1.3168
100%|██████████| 32/32 [00:03<00:00,  8.92it/s]
Epoch [105] | G: 1.4111 | D: 1.4011 | Recon: 0.0474 | KL: 1.2894
100%|██████████| 32/32 [00:03<00:00,  9.10it/s]
Epoch [106] | G: 1.4639 | D: 1.3740 | Recon: 0.0484 | KL: 1.3271
100%|██████████| 32/32 [00:04<00:00,  6.63it/s]
Epoch [107] | G: 1.5445 | D: 1.3618 | Recon: 0.0532 | KL: 1.3216
100%|██████████| 32/32 [00:04<00:00,  7.99it/s]
Epoch [108] | G: 1.4819 | D: 1.3452 | Recon: 0.0472 | KL: 1.3740
100%|██████████| 32/32 [00:04<00:00,  7.32it/s]
Epoch [109] | G: 1.4130 | D: 1.4311 | Recon: 0.0506 | KL: 1.2944
100%|██████████| 32/32 [00:04<00:00,  7.20it/s]
Epoch [110] | G: 1.4460 | D: 1.3390 | Recon: 0.0448 | KL: 1.3325
100%|██████████| 32/32 [00:04<00:00,  7.05it/s]
Epoch [111] | G: 1.5141 | D: 1.3356 | Recon: 0.0478 | KL: 1.3618
100%|██████████| 32/32 [00:03<00:00,  8.84it/s]
Epoch [112] | G: 1.3944 | D: 1.4910 | Recon: 0.0545 | KL: 1.2719
100%|██████████| 32/32 [00:03<00:00,  8.94it/s]
Epoch [113] | G: 1.4238 | D: 1.3354 | Recon: 0.0443 | KL: 1.3208
100%|██████████| 32/32 [00:05<00:00,  6.03it/s]
Epoch [114] | G: 1.5377 | D: 1.3227 | Recon: 0.0475 | KL: 1.3542
100%|██████████| 32/32 [00:03<00:00,  9.01it/s]
Epoch [115] | G: 1.4261 | D: 1.4316 | Recon: 0.0520 | KL: 1.3144
100%|██████████| 32/32 [00:03<00:00,  8.73it/s]
Epoch [116] | G: 1.3615 | D: 1.4562 | Recon: 0.0479 | KL: 1.2698
100%|██████████| 32/32 [00:04<00:00,  6.59it/s]
Epoch [117] | G: 1.3885 | D: 1.3325 | Recon: 0.0411 | KL: 1.3107
100%|██████████| 32/32 [00:03<00:00,  8.16it/s]
Epoch [118] | G: 1.4473 | D: 1.3172 | Recon: 0.0420 | KL: 1.3634
100%|██████████| 32/32 [00:03<00:00,  8.86it/s]
Epoch [119] | G: 1.4410 | D: 1.3686 | Recon: 0.0462 | KL: 1.3625
100%|██████████| 32/32 [00:03<00:00,  8.47it/s]
Epoch [120] | G: 1.4783 | D: 1.4072 | Recon: 0.0497 | KL: 1.3025
100%|██████████| 32/32 [00:05<00:00,  6.29it/s]
Epoch [121] | G: 1.4356 | D: 1.4039 | Recon: 0.0494 | KL: 1.3313
100%|██████████| 32/32 [00:03<00:00,  8.83it/s]
Epoch [122] | G: 1.4208 | D: 1.3373 | Recon: 0.0454 | KL: 1.3224
100%|██████████| 32/32 [00:03<00:00,  9.03it/s]
Epoch [123] | G: 1.4095 | D: 1.4193 | Recon: 0.0487 | KL: 1.3204
100%|██████████| 32/32 [00:05<00:00,  6.08it/s]
Epoch [124] | G: 1.4017 | D: 1.4096 | Recon: 0.0462 | KL: 1.3154
100%|██████████| 32/32 [00:03<00:00,  8.15it/s]
Epoch [125] | G: 1.4018 | D: 1.3389 | Recon: 0.0416 | KL: 1.3134
100%|██████████| 32/32 [00:03<00:00,  8.86it/s]
Epoch [126] | G: 1.3926 | D: 1.3697 | Recon: 0.0443 | KL: 1.3465
```

```
100%|██████████| 32/32 [00:04<00:00,  7.13it/s]
Epoch [127] | G: 1.3993 | D: 1.3707 | Recon: 0.0454 | KL: 1.3670
100%|██████████| 32/32 [00:04<00:00,  7.82it/s]
Epoch [128] | G: 1.4264 | D: 1.4078 | Recon: 0.0481 | KL: 1.2898
```

```
from PIL import Image
import matplotlib.pyplot as plt
import os

vae_img_path = "/content/drive/MyDrive/flower_results/vae_images/recon_
gan_img_path = "/content/drive/MyDrive/flower_gan_results/images/epoch_

vae_img = Image.open(vae_img_path)
gan_img = Image.open(gan_img_path)

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
plt.imshow(vae_img)
plt.title("VAE Output (Blurry)")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(gan_img)
plt.title("VAE-GAN Output (Sharper)")
plt.axis("off")

plt.show()
```

Start coding or generate with AI.