```
!pip install opendatasets --upgrade --quiet
```

```
import opendatasets as od
download_url="https://www.kaggle.com/splcher/animefacedataset"
od.download(download_url)
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: jainmridul
Your Kaggle Key: ·····
  5%|▌         | 21.0M/395M [00:00<00:01, 215MB/s]Downloading animefacedataset.zip to ./animefacedataset
100%|██████████| 395M/395M [00:01<00:00, 216MB/s]
```

```
import os

DATA_DIR="/content/animefacedataset"
print(os.listdir(DATA_DIR))
```

```
['images']
```

```
print(os.listdir(DATA_DIR+"/images"))
```

```
['61422_2018.jpg', '31445_2010.jpg', '55156_2016.jpg', '21013_2008.jpg', '62363_2019.jpg', '55836_2017.jpg', '44501_2013.jpg', '26055_2009.jpg', '29096_2010.jpg', '5363_2003.jpg',
```

```
import torch
import torchvision
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
matplotlib.rcParams["figure.facecolor"]="white"
```

```
image_size=64
batch_size=128
stats=(0.5,0.5,0.5),(0.5,0.5,0.5)
```

```
train_ds=torchvision.datasets.ImageFolder(DATA_DIR,transform=torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Resize(image_size),
        torchvision.transforms.Normalize(*stats,inplace=True),
        torchvision.transforms.CenterCrop(image_size)
]))
train_dataloader=torch.utils.data.DataLoader(train_ds,batch_size,shuffle=True,num_workers=3,pin_memory=True)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:477: UserWarning: This DataLoader will create 3 worker processes in total. Our suggested max number of worke
  cpuset_checked))
```

```
def denormalize(images,device,mean,std):
    mean=torch.tensor(mean,device=device).reshape(1,3,1,1)
    std=torch.tensor(std,device=device).reshape(1,3,1,1)
    images=images.to(device)
    return images*std+mean


def show_batch(dataloader,nmax=64):
    for images,_ in dataloader:
        fig,ax=plt.subplots(figsize=(8,8))
        ax.set_xticks([])
        ax.set_yticks([])
        denorm=denormalize(images[:nmax],device,*stats)
        X=torchvision.utils.make_grid(denorm,nrow=8)
        ax.imshow(X.detach().cpu().permute(1,2,0))
        break


def get_default_device():
    if torch.cuda.is_available():
        return torch.device("cuda")
    return torch.device("cuda")


device=get_default_device()
print(device)
```

```
cuda
```

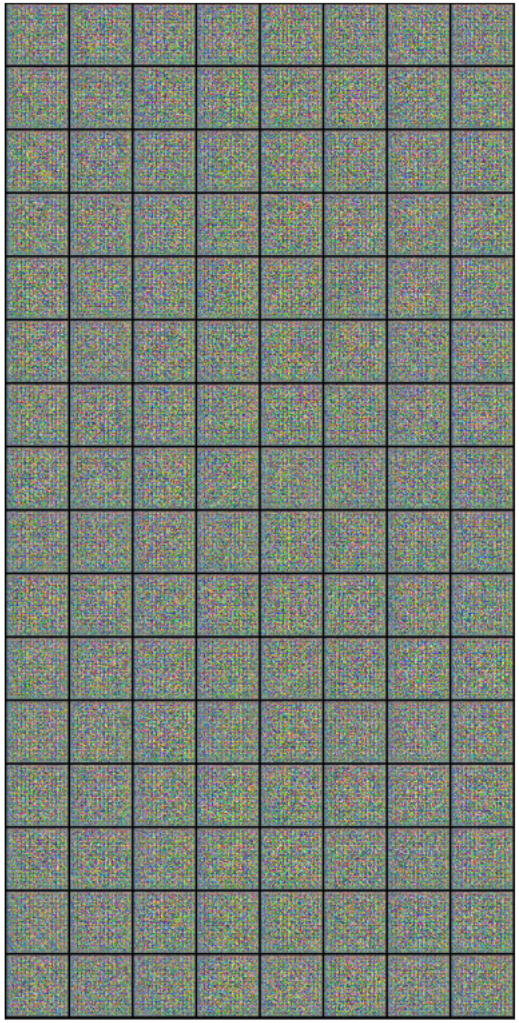
```
def to_device(data,device):
    if isinstance(data,(list,tuple)):
        return [to_device(x,device) for x in data]
    return data.to(device,non_blocking=True)


class DeviceDataLoader:
    def __init__(self,data,device):
        self.data=data
        self.device=device
    def __len__(self):
        return len(self.data)
    def __iter__(self):
        for x in self.data:
            yield to_device(x,self.device)


show_batch(train_dataloader,128)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:477: UserWarning: This DataLoader will create 3 worker processes in total. Our suggested max number of worker
  cpuset_checked))
```

```
train_loader=DeviceDataLoader(train_dataloader,device)


discriminator=torch.nn.Sequential(
    torch.nn.Conv2d(3,64,kernel_size=4,padding=1,stride=2,bias=False), # (128,3,64,64)->(128,64,32,32)
    torch.nn.BatchNorm2d(64),
    torch.nn.LeakyReLU(0.2,inplace=True),

    torch.nn.Conv2d(64,128,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(128),
    torch.nn.LeakyReLU(0.2,inplace=True),

    torch.nn.Conv2d(128,256,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(256),
    torch.nn.LeakyReLU(0.2,inplace=True),

    torch.nn.Conv2d(256,512,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(512),
    torch.nn.LeakyReLU(0.2,inplace=True),

    torch.nn.Conv2d(512,1,kernel_size=4,padding=0,stride=1,bias=False),
    torch.nn.Flatten(),
    torch.nn.Sigmoid()
)
```

```
discriminator=to_device(discriminator,device)


latent_size=128


generator=torch.nn.Sequential(
    torch.nn.ConvTranspose2d(latent_size,512,kernel_size=4,padding=0,stride=1,bias=False),
    torch.nn.BatchNorm2d(512),
    torch.nn.ReLU(inplace=True), # (512,4,4)

    torch.nn.ConvTranspose2d(512,256,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(256),
    torch.nn.ReLU(inplace=True), # (256,8,8)

    torch.nn.ConvTranspose2d(256,128,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(128),
    torch.nn.ReLU(inplace=True), # (128,16,16)

    torch.nn.ConvTranspose2d(128,64,kernel_size=4,padding=1,stride=2,bias=False),
    torch.nn.BatchNorm2d(64),
    torch.nn.ReLU(inplace=True), # (64,32,32)

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


def show_images(images):
    fig,ax=plt.subplots(figsize=(14,14))
    ax.set_xticks([])
    ax.set_yticks([])
    denorm=denormalize(images,device,*stats)
    X=torchvision.utils.make_grid(denorm,nrow=8)
    ax.imshow(X.detach().cpu().permute(1,2,0))


images=torch.randn(batch_size,latent_size,1,1)
fake_images=generator(images)
print(fake_images.shape)
show_images(fake_images)
```

```
torch.Size([128, 3, 64, 64])
```



```
generator=to_device(generator,device)
```

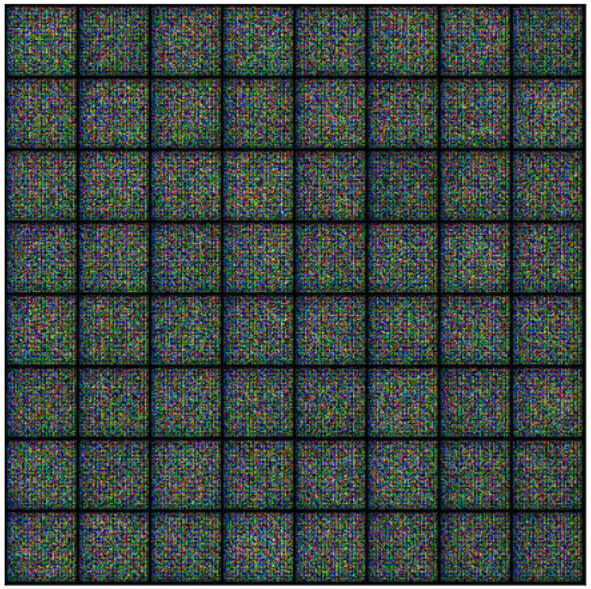
```
images1=torch.randn(100,3,64,64)
images1=to_device(images1,device)
print(images1.shape)
out=discriminator(images1)
print(out.shape)
```

```
torch.Size([100, 3, 64, 64])
torch.Size([100, 1])
```

```
def train_discriminator(real_imgs,opt_d):
    opt_d.zero_grad()
    real_preds=discriminator(real_imgs)
    real_targets=torch.ones(real_imgs.size(0),1,device=device)
    real_loss=torch.nn.functional.binary_cross_entropy(real_preds,real_targets)
    real_score=torch.mean(real_preds).item()

    latent=torch.randn(batch_size,latent_size,1,1,device=device)
    fake_images=generator(latent)

    fake_preds=discriminator(fake_images)
    fake_targets=torch.zeros(fake_images.size(0),1,device=device)
    fake_loss=torch.nn.functional.binary_cross_entropy(fake_preds,fake_targets)
    fake_score=torch.mean(fake_preds).item()

    loss=real_loss+fake_loss
    loss.backward()
    opt_d.step()
    return loss.item(),real_score,fake_score


def train_generator(opt_g):
    opt_g.zero_grad()
    latent=torch.randn(batch_size,latent_size,1,1,device=device)
    fake_images=generator(latent)
    preds=discriminator(fake_images)
    targets=torch.ones(batch_size,1,device=device)
    loss=torch.nn.functional.binary_cross_entropy(preds,targets)
    loss.backward()
    opt_g.step()
    return loss.item()


import os


sample_dir="generated-new-latest"
os.makedirs(sample_dir,exist_ok=True)


fixed_latent=torch.randn(64,latent_size,1,1,device=device)


def save_examples(index,latent_tensors,device,show=True):
    fake_images=generator(latent_tensors)
    fake_fname="generated-images-{0:0=4d}.png".format(index)
    torchvision.utils.save_image(denormalize(fake_images,device,*stats),os.path.join(sample_dir,fake_fname),nrow=8)
    print('Saving',fake_fname)
    if show:
        fig,ax=plt.subplots(figsize=(8,8))
        ax.set_xticks([]); ax.set_yticks([])
        X=torchvision.utils.make_grid(fake_images.cpu().detach(),nrow=8)
        ax.imshow(X.permute(1,2,0))
```

```
save_examples(0,fixed_latent,device=device)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Saving generated-images-0000.png
```

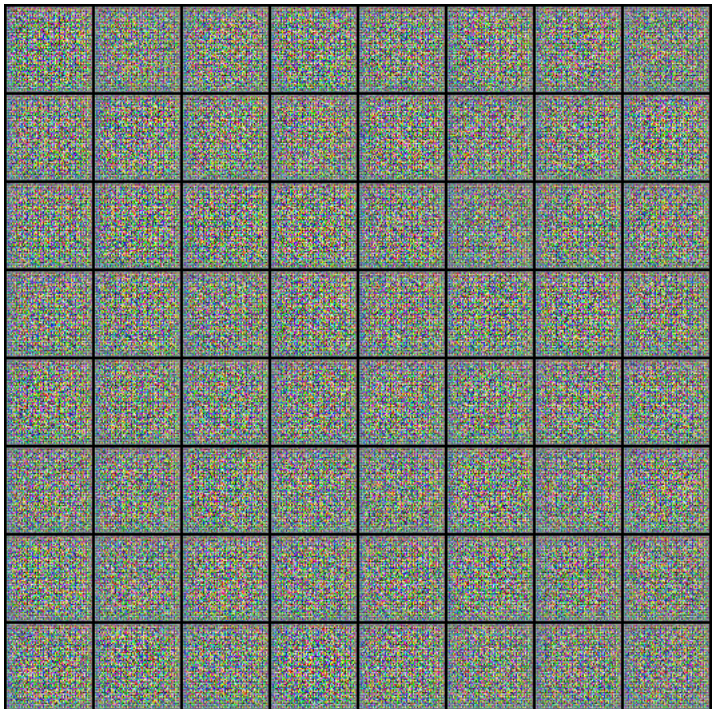


```
import tqdm


def fit(epochs,lr,start_idx=1):
    torch.cuda.empty_cache()
    losses_g=[]
    losses_d=[]
    real_scores=[]
    fake_scores=[]
    opt_g=torch.optim.Adam(generator.parameters(),lr=lr,betas=(0.5,0.9999))
    opt_d=torch.optim.Adam(discriminator.parameters(),lr=lr,betas=(0.5,0.9999))
    for epoch in range(epochs):
        for real_imgs,_ in tqdm.notebook.tqdm(train_loader):
            loss_d,real_score,fake_score=train_discriminator(real_imgs,opt_d)
            loss_g=train_generator(opt_g)
        losses_g.append(loss_g)
        losses_d.append(loss_d)
        real_scores.append(real_score)
        fake_scores.append(fake_score)
        print("Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_score: {:.4f}, fake_score: {:.4f}".format(
            epoch+1,epochs,loss_g,loss_d,real_score,fake_score))
        save_examples(epoch+start_idx,fixed_latent,device,show=False)
    return losses_g,losses_d,real_scores,fake_scores
```

```
epochs=3
lr=0.0002


history=fit(epochs,lr)
```

```
100%                                497/497 [07:02<00:00, 1.18it/s]
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:477: UserWarning: This DataLoader will create 3 worker processes in total. Our suggested max number of worke
  cpuset_checked))

Epoch [1/3], loss_g: 4.6594, loss_d: 0.8442, real_score: 0.5378, fake_score: 0.0130
Saving generated-images-0001.png
100%                                497/497 [05:03<00:00, 1.63it/s]

Epoch [2/3], loss_g: 5.5428, loss_d: 0.4316, real_score: 0.7205, fake_score: 0.0173
Saving generated-images-0002.png
100%                                497/497 [02:32<00:00, 3.26it/s]

Epoch [3/3], loss_g: 2.2972, loss_d: 0.2931, real_score: 0.8552, fake_score: 0.0976
Saving generated-images-0003.png
```

```
import IPython


IPython.display.Image("/content/generated-new-latest/generated-images-0000.png")
```



```
IPython.display.Image("/content/generated-new-latest/generated-images-0001.png")
```

```
IPython.display.Image("/content/generated-new-latest/generated-images-0002.png")
```

```
IPython.display.Image("/content/generated-new-latest/generated-images-0003.png")
```

```
import cv2
import os


vid_fname="gans_training.avi"
files=[os.path.join(sample_dir,f) for f in os.listdir(sample_dir) if "generated" in f]
files.sort()
out=cv2.VideoWriter(vid_fname,cv2.VideoWriter_fourcc(*"MP4V"),1,(530,530))
[out.write(cv2.imread(fname)) for fname in files]
out.release()
```