PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Deep Learning

Homework 7

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: April 5, 2023

1 Experiment

In this homework, we do the following:

1. Create our own pizza-generating Generative Adversarial Networks (GANs) using Binary Cross Entropy and Wasserstein distance.

2. Evaluate images generated by both the models qualitatively, and quantitatively using the Frechet Inception Distance (FID).

I have referred to Prof. Kak's presentation on Generative Adversarial Networks for Data Modeling for this assignment.

1.1 Dataset

I first downloaded the provided training and validation dataset of pizzas. There were 8213 training and 1000 validation images of size (64×64).

1.2 Model Training

The architecture of the Generator and Discriminator models are similar to the Deep Convolutional Generative Adversarial Networks (DCGANs) presented by Professor Kak in class. The optimizer used was Adam with beta values of (0.5,0.999), the learning rate was 0.0001 for 40 epochs and the batch size was 16.
The Generator creates images of size $(3 \times 64 \times 64)$ by taking in a random noise vector of shape $(100 \times 1 \times 1)$ by utilizing transpose convolutions. The Discriminator has the typical architecture of a binary classifier.
During transpose convolutions, learnable filters are applied to the input tensor. The output tensor generated has a larger spatial dimension than the input tensor. The relationship is given as follows:

$$output\_dim = (input_dim - 1) \times stride + kernel\_size - 2padding$$

1.2.1 Training

The Discriminator is trained to maximize the probability of assigning the correct label to the image. Meanwhile the Generator is trained to minimize the probability that the Discriminator assigns the correct label to the fake image. During this minimax game, we initially update the parameters of the Discriminator on real and fake images followed by updating the Generator's parameters on the fake images.

The following are the two different losses used:

1. Binary Cross Entropy
2. Wasserstein distance

1.2.2 Evaluation

This is done both qualitatively and quantitatively. For the former, we plot 16 fake images generated by both models. For the latter, we use Frechet Inception Distance (FID) which is calculated as follows:

1. We use a pre-trained Inception network to encode the 1000 real test images and 1000 fake images.
2. Then we model the distribution of the resulting feature vectors for both sets of images using multivariate Gaussian distribution.
3. Finally, we calculate the Frechet distance between these two distributions.

2 Results

2.1 Dataset

The following are 16 real images of pizzas from the training images.

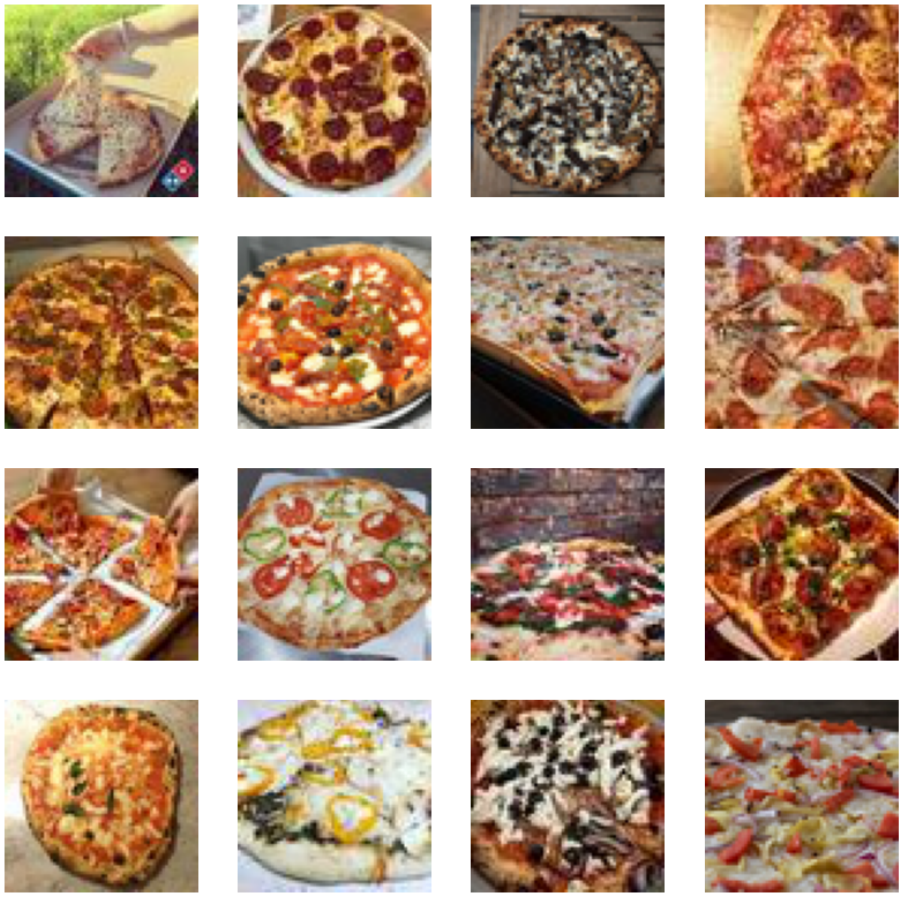


Figure 1: Real Pizza images

2.2 Model Training

2.2.1 For Loss

Below, I have plotted the loss vs epochs graph for the training dataset. The Generator loss and Discriminator loss are plotted.

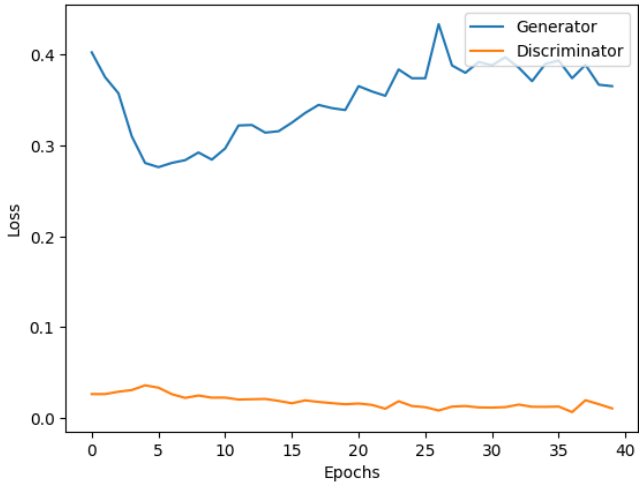


Figure 2: For BCE-GAN

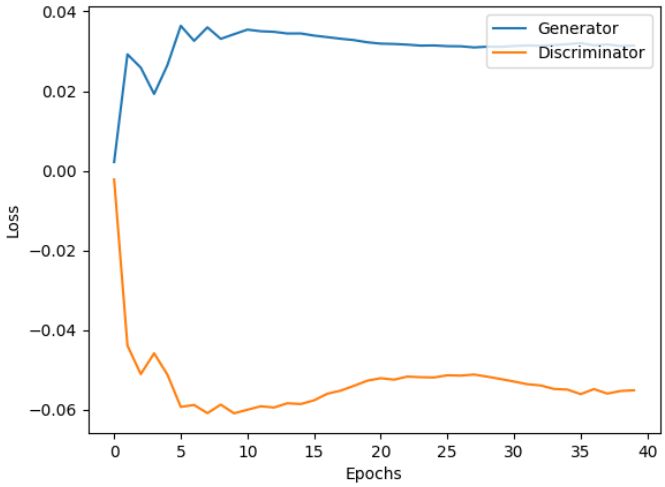


Figure 3: For W-GAN

2.2.2 Qualitative Assessment

Below are 16 fake images generated by BCE-GAN and W-GAN.

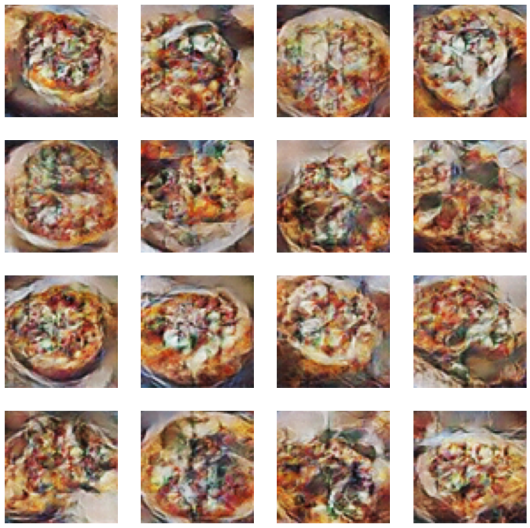


Figure 4: For BCE-GAN

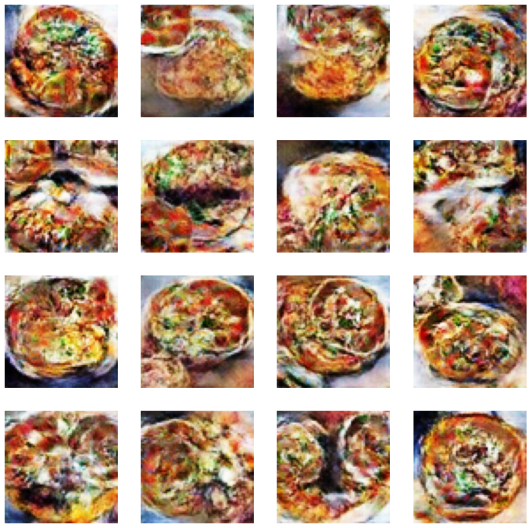


Figure 5: For W-GAN

2.2.3 Quantitative Assessment

The following are the FID value vs epochs graph on the test dataset

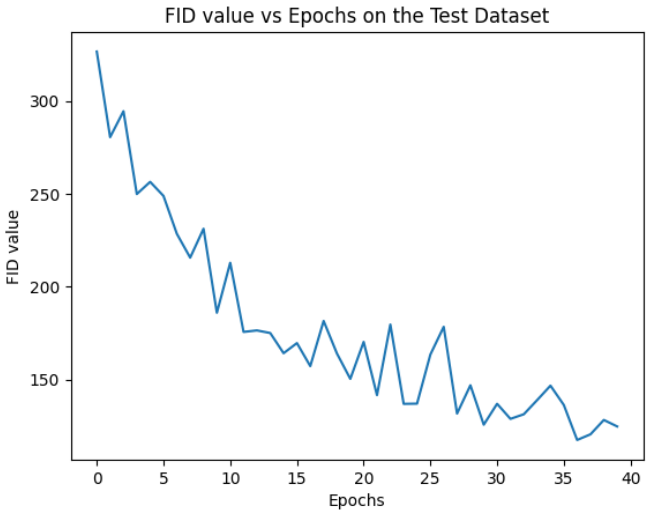


Figure 6: For BCE-GAN. The FID value after 40 epochs is 124.69

Figure 7: For W-GAN. The FID value after 40 epochs is 148.35

3 Conclusion

Quantitatively, the FID value of BCE-GAN is 124.69 and W-GAN is 148.35. The lower this value, the better is the performance of the model

Qualitatively, the shape of the pizzas in the images produced by BCE-GAN appear better than those generated by W-GAN. Whereas the images produced by W-GAN appear more vibrant than those produced by BCE-GAN.

So overall, I would say that the BCE-GAN performed better than W-GAN for this particular task.

In [ ]:

```
pip install pytorch_fid
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.p
kg.dev/colab-wheels/public/simple/)
Requirement already satisfied: pytorch_fid in /usr/local/lib/python3.9/di
st-packages (0.3.0)
Requirement already satisfied: torch>=1.0.1 in /usr/local/lib/python3.9/d
ist-packages (from pytorch_fid) (2.0.0+cu118)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-pac
kages (from pytorch_fid) (1.22.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-pa
ckages (from pytorch_fid) (8.4.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-pac
kages (from pytorch_fid) (1.10.1)
Requirement already satisfied: torchvision>=0.2.2 in /usr/local/lib/pytho
n3.9/dist-packages (from pytorch_fid) (0.15.1+cu118)
Requirement already satisfied: networkx in /usr/local/lib/python3.9/dist-
packages (from torch>=1.0.1->pytorch_fid) (3.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python
3.9/dist-packages (from torch>=1.0.1->pytorch_fid) (4.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-
packages (from torch>=1.0.1->pytorch_fid) (3.10.7)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.9/
dist-packages (from torch>=1.0.1->pytorch_fid) (2.0.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.9/dist-pac
kages (from torch>=1.0.1->pytorch_fid) (1.11.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-pa
ckages (from torch>=1.0.1->pytorch_fid) (3.1.2)
Requirement already satisfied: cmake in /usr/local/lib/python3.9/dist-pac
kages (from triton==2.0.0->torch>=1.0.1->pytorch_fid) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packa
ges (from triton==2.0.0->torch>=1.0.1->pytorch_fid) (16.0.0)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-
packages (from torchvision>=0.2.2->pytorch_fid) (2.27.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
9/dist-packages (from jinja2->torch>=1.0.1->pytorch_fid) (2.1.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/py
thon3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fid)
(1.26.15)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/li
b/python3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fi
d) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
n3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fid) (202
2.12.7)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/d
ist-packages (from requests->torchvision>=0.2.2->pytorch_fid) (3.4)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.9/d
ist-packages (from sympy->torch>=1.0.1->pytorch_fid) (1.3.0)
```

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import Dataset, DataLoader
from torchvision import models,datasets,transforms
from PIL import Image
from tqdm import tqdm

import os
import matplotlib.pyplot as plt
import numpy as np

from pytorch_fid.fid_score import calculate_activation_statistics,calculate_frechet_dist
from pytorch_fid . inception import InceptionV3
import shutil
```

In [ ]:

```
class IndexedDataset(Dataset):
    def __init__(self, dir_path):
        self.dir_path = dir_path
        #The transforms that will be applied to each image
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
          ])
        #Saving all the image locations
        self.image_filenames = []
        for (dirpath, dirnames, filenames) in os.walk(dir_path):
            self.image_filenames += [os.path.join(dirpath, file) for file in filenames]

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

    def __getitem__(self, idx):
        img_name = self.image_filenames[idx] #Getting the name of the image
        image = Image.open(img_name).convert('RGB') #opening the image
        image = self.transform(image) #Applying the transforms to the image
        return image
```

In [ ]:

```
#Inspired by https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.convt1 = nn.ConvTranspose2d( 100, 64 * 8, 4, 1, 0, bias=False)
        self.convt2 = nn.ConvTranspose2d(64 * 8, 64 * 4, 4, 2, 1, bias=False)
        self.convt3 = nn.ConvTranspose2d( 64 * 4, 64 * 2, 4, 2, 1, bias=False)
        self.convt4 = nn.ConvTranspose2d( 64 * 2, 64, 4, 2, 1, bias=False)
        self.convt5 = nn.ConvTranspose2d( 64, 3, 4, 2, 1, bias=False)
        self.bn1 = nn.BatchNorm2d(64 * 8)
        self.bn2 = nn.BatchNorm2d(64 * 4)
        self.bn3 = nn.BatchNorm2d(64*2)
        self.bn4 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(True)
        self.tanh = nn.Tanh()

    def forward(self, input):
        x = self.relu(self.bn1(self.convt1(input)))
        x = self.relu(self.bn2(self.convt2(x)))
        x = self.relu(self.bn3(self.convt3(x)))
        x = self.relu(self.bn4(self.convt4(x)))
        x = self.tanh(self.convt5(x))
        return x
```

In [ ]:

```
#Inspired by https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, 4, 2, 1, bias=False)
        self.conv2 = nn.Conv2d(64, 64 * 2, 4, 2, 1, bias=False)
        self.conv3 = nn.Conv2d(64 * 2, 64 * 4, 4, 2, 1, bias=False)
        self.conv4 = nn.Conv2d(64 * 4, 64 * 8, 4, 2, 1, bias=False)
        self.conv5 = nn.Conv2d(64 * 8, 1, 4, 1, 0, bias=False)
        self.bn2 = nn.BatchNorm2d(64 * 2)
        self.bn3 = nn.BatchNorm2d(64 * 4)
        self.bn4 = nn.BatchNorm2d(64 * 8)
        self.relu = nn.LeakyReLU(0.2, inplace=True)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input):
        x = self.relu(self.conv1(input))
        x = self.relu(self.bn2(self.conv2(x)))
        x = self.relu(self.bn3(self.conv3(x)))
        x = self.relu(self.bn4(self.conv4(x)))
        x = self.sigmoid(self.conv5(x))
        return x
```

In [ ]:

```
#Initializing the weights
#From https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

In [ ]:

```
#Function to train the model
def training(epochs,optimizerD,optimizerG,criterion,netG,netD,train_data_loader,device,b
  #Storing the losses of the Generator and Discriminator
  train_lossesG = []
  train_lossesD = []
  fid_score = []

  #Assigning labels to the real and fake images
  real_label = 1
  fake_label = 0

  for epoch in range(epochs):
    running_lossG = 0.0
    running_lossD = 0.0
    for i,data in enumerate(train_data_loader):
      #Part 1: Training the Discriminator
      #Part A: On the real images
      optimizerD.zero_grad()
      inputs_real = data.to(device) #Getting the real image
      num_images = data.shape[0] #Getting the batch size
      labels_real = torch.full((num_images,),real_label,dtype=torch.float).to(device)  #

      outputs_real = netD(inputs_real).view(-1) #Getting the predicted labels for the re
      loss_real = criterion(outputs_real, labels_real) #Calculating the loss on real ima
      loss_real.backward()

      #Part B: On the fake images
      noise = torch.randn((num_images,100,1,1)).to(device) #Generating the loss
      inputs_fake = netG(noise) #Generating the fake images from this noise
      labels_fake = torch.full((num_images,),fake_label,dtype=torch.float).to(device) #G
      outputs_fake = netD(inputs_fake.detach()).view(-1) #Getting the predicted labels f
      loss_fake = criterion(outputs_fake, labels_fake) #Calculating the loss on fake ima
      loss_fake.backward()

      optimizerD.step()
      running_lossD += loss_real.cpu().item() + loss_fake.cpu().item()

      #Part 2: Training the Generator
      optimizerG.zero_grad()
      outputs_fake_G = netD(inputs_fake).view(-1) #Genrating the predicted labels for th
      loss_G = criterion(outputs_fake_G, labels_real) #In the optimum case the fake imag
      loss_G.backward()
      optimizerG.step()
      running_lossG += loss_G.cpu().item()

    print("[epoch: %d, batch: %5d] Discriminator loss: %.3f Generator loss: %.3f" % (epo
    train_lossesG.append(running_lossG/len(train_data_loader.dataset))
    train_lossesD.append(running_lossD/len(train_data_loader.dataset))
    fid = calc_fid(netG,device,m1,s1,model)
    fid_score.append(fid)
  return netG, train_lossesG, train_lossesD, fid_score
```

In [ ]:

```
#Function to Plot a real image
def plot_image_real(dataset,index):
  img = dataset[index] #Getting the image
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  return ci.astype(int)  #returns the image
```

In [ ]:

```
#Function to Plot a fake image
def plot_image_fake(netG,device):
  noise = torch.randn((1,100,1,1)).to(device)
  img = netG(noise).cpu().detach() #Getting the image
  img = img.squeeze()
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  return ci.astype(int)  #returns the image
```

In [ ]:

```
#Function to save a fake image
def save_image_fake(img,i):
  im = Image.fromarray(img.astype(np.uint8))
  im.save("fake/"+str(i)+".jpeg")
```

In [ ]:

```
#Function to calculate FID
def calc_fid(netG,device,m1,s1,model):
  #Generating 1000 fake images
  for i in range(1000):
    img = plot_image_fake(netG,device)
    save_image_fake(img,i)
  fake_paths = ["fake/" +str(i)+".jpeg" for i in range(0, 1000)] #Generating the path fo
  #Obtaining the feature encoding for the real test images
  m2 , s2 = calculate_activation_statistics(fake_paths, model,device = device)
  fid_value = calculate_frechet_distance(m1,s1,m2,s2) #Calculating FID value
  print("FID: "+str(fid_value))
  return fid_value
```

In [ ]:

```
#Function to Plot 4x4 images
def plot_collage(dataset,device,type_of="real",model=None):
  images = []
  if type_of=="real": #If you want real image generated
    indices = np.random.randint(0,len(dataset),16) #Getting 16 random indices
    for i in indices:
      images.append(plot_image_real(dataset,i)) #Saving the 16 images

  elif type_of=="fake": #If you want fake image generated
    for i in range(16):
      images.append(plot_image_fake(model,device)) #Saving the 16 images

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

  ax11 = fig.add_subplot(4,4,1)
  ax12 = fig.add_subplot(4,4,2)
  ax13 = fig.add_subplot(4,4,3)
  ax14 = fig.add_subplot(4,4,4)
  ax21 = fig.add_subplot(4,4,5)
  ax22 = fig.add_subplot(4,4,6)
  ax23 = fig.add_subplot(4,4,7)
  ax24 = fig.add_subplot(4,4,8)
  ax31 = fig.add_subplot(4,4,9)
  ax32 = fig.add_subplot(4,4,10)
  ax33 = fig.add_subplot(4,4,11)
  ax34 = fig.add_subplot(4,4,12)
  ax41 = fig.add_subplot(4,4,13)
  ax42 = fig.add_subplot(4,4,14)
  ax43 = fig.add_subplot(4,4,15)
  ax44 = fig.add_subplot(4,4,16)

  ax11.axis('off')
  ax12.axis('off')
  ax13.axis('off')
  ax14.axis('off')
  ax21.axis('off')
  ax22.axis('off')
  ax23.axis('off')
  ax24.axis('off')
  ax31.axis('off')
  ax32.axis('off')
  ax33.axis('off')
  ax34.axis('off')
  ax41.axis('off')
  ax42.axis('off')
  ax43.axis('off')
  ax44.axis('off')

  ax11.imshow(images[0])
  ax12.imshow(images[1])
  ax13.imshow(images[2])
  ax14.imshow(images[3])
  ax21.imshow(images[4])
  ax22.imshow(images[5])
  ax23.imshow(images[6])
  ax24.imshow(images[7])
  ax31.imshow(images[8])
  ax32.imshow(images[9])
  ax33.imshow(images[10])
```

```
  ax34.imshow(images[11])
  ax41.imshow(images[12])
  ax42.imshow(images[13])
  ax43.imshow(images[14])
  ax44.imshow(images[15])
```

In [ ]:

```
train_dataset = IndexedDataset("/content/drive/MyDrive/pizzas/train")
test_dataset = IndexedDataset("/content/drive/MyDrive/pizzas/eval")

train_dataloader = DataLoader(train_dataset,batch_size=16,num_workers=64)
test_dataloader = DataLoader(test_dataset,batch_size=16,num_workers=64)

print(len(train_dataloader))
print(len(test_dataloader))
```

```
514
63

/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:56
1: UserWarning: This DataLoader will create 64 worker processes in total.
Our suggested max number of worker in current system is 2, which is small
er than what this DataLoader is going to create. Please be aware that exc
essive worker creation might get DataLoader running slow or even freeze,
lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

In [ ]:

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

Out[14]:

```
device(type='cuda')
```

In [ ]:

```
netG = Generator().to(device)
netG.apply(weights_init)
netD = Discriminator().to(device)
netD.apply(weights_init)

optimizerG = torch.optim.Adam(netG.parameters(), lr=0.0001, betas = (0.5, 0.999))
optimizerD = torch.optim.Adam(netD.parameters(), lr=0.0001, betas = (0.5, 0.999))
criterion = nn.BCELoss()
epochs = 40
```

In [ ]:

```
#The number of parameters in the Generator
sum(p.numel() for p in netG.parameters() if p.requires_grad)
```

Out[16]:

```
3576704
```

In [ ]:

```
#The number of parameters in the Discriminator
sum(p.numel() for p in netD.parameters() if p.requires_grad)
```

Out[17]:

```
2765568
```

In [ ]:

```
#Creating an instance of the InceptionV3 model
dims = 2048
block_idx = InceptionV3.BLOCK_INDEX_BY_DIM[dims]
model = InceptionV3([block_idx]).to(device)
```

In [ ]:

```
#Obtaining the feature encoding for the real test images
real_paths = test_dataset.image_filenames
m1 , s1 = calculate_activation_statistics(real_paths,model,device = device)
```

```
100%|██████████| 20/20 [00:05<00:00,  3.47it/s]
```

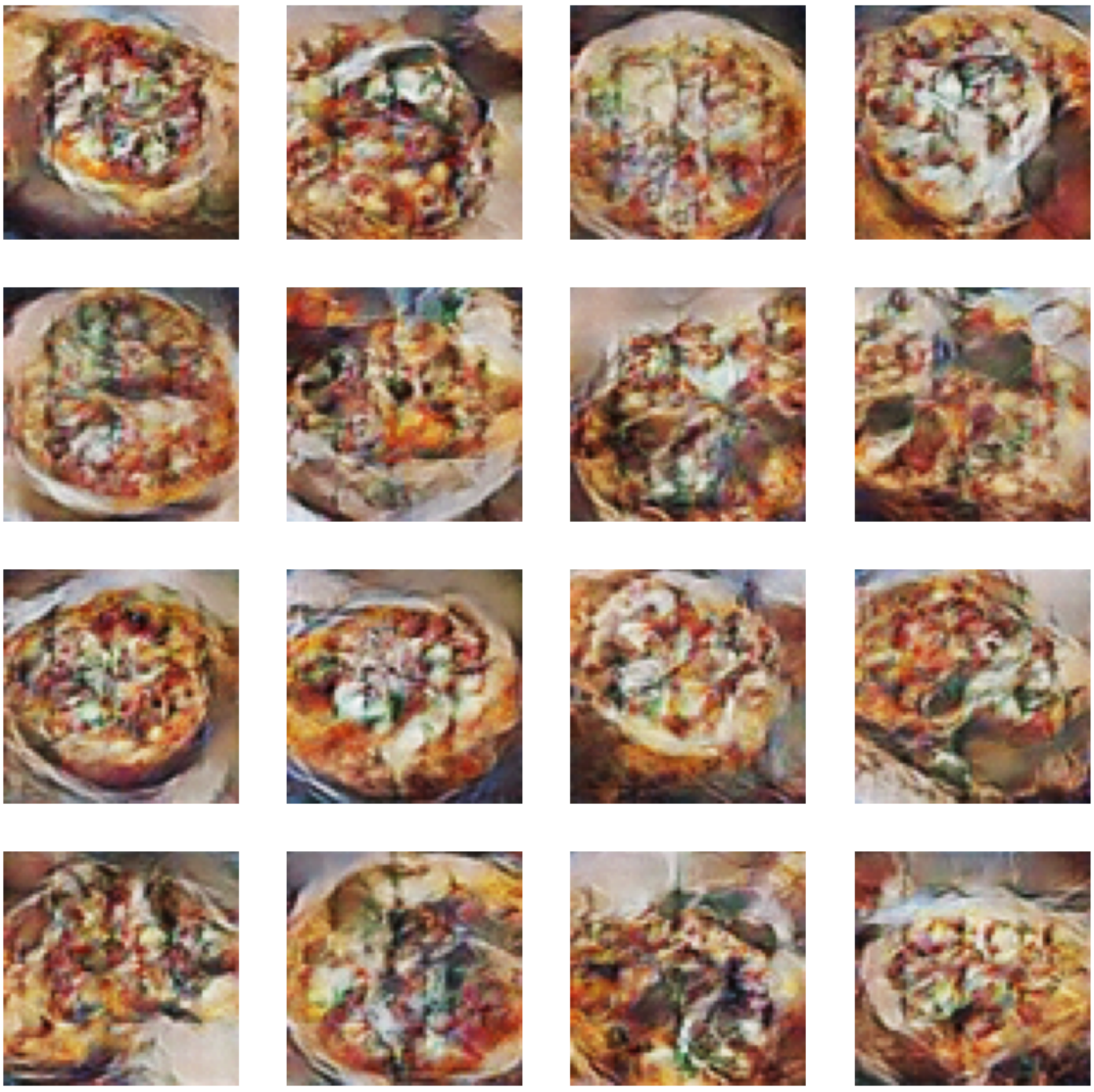
In [ ]:

```
trained_generator,train_lossesG, train_lossesD,fid_score = training(epochs,optimizerD,op
```

```
[epoch: 1, batch:   514] Discriminator loss: 0.424 Generator loss: 6.4
33

100%|██████████| 20/20 [00:03<00:00,  5.13it/s]

FID: 326.65443838158586
[epoch: 2, batch:   514] Discriminator loss: 0.424 Generator loss: 5.9
97

100%|██████████| 20/20 [00:03<00:00,  5.10it/s]

FID: 280.54213066301094
[epoch: 3, batch:   514] Discriminator loss: 0.465 Generator loss: 5.7
14

100%|██████████| 20/20 [00:03<00:00,  5.09it/s]

FID: 294.5241699903557
[epoch: 4, batch:   514] Discriminator loss: 0.495 Generator loss: 4.9
57
```

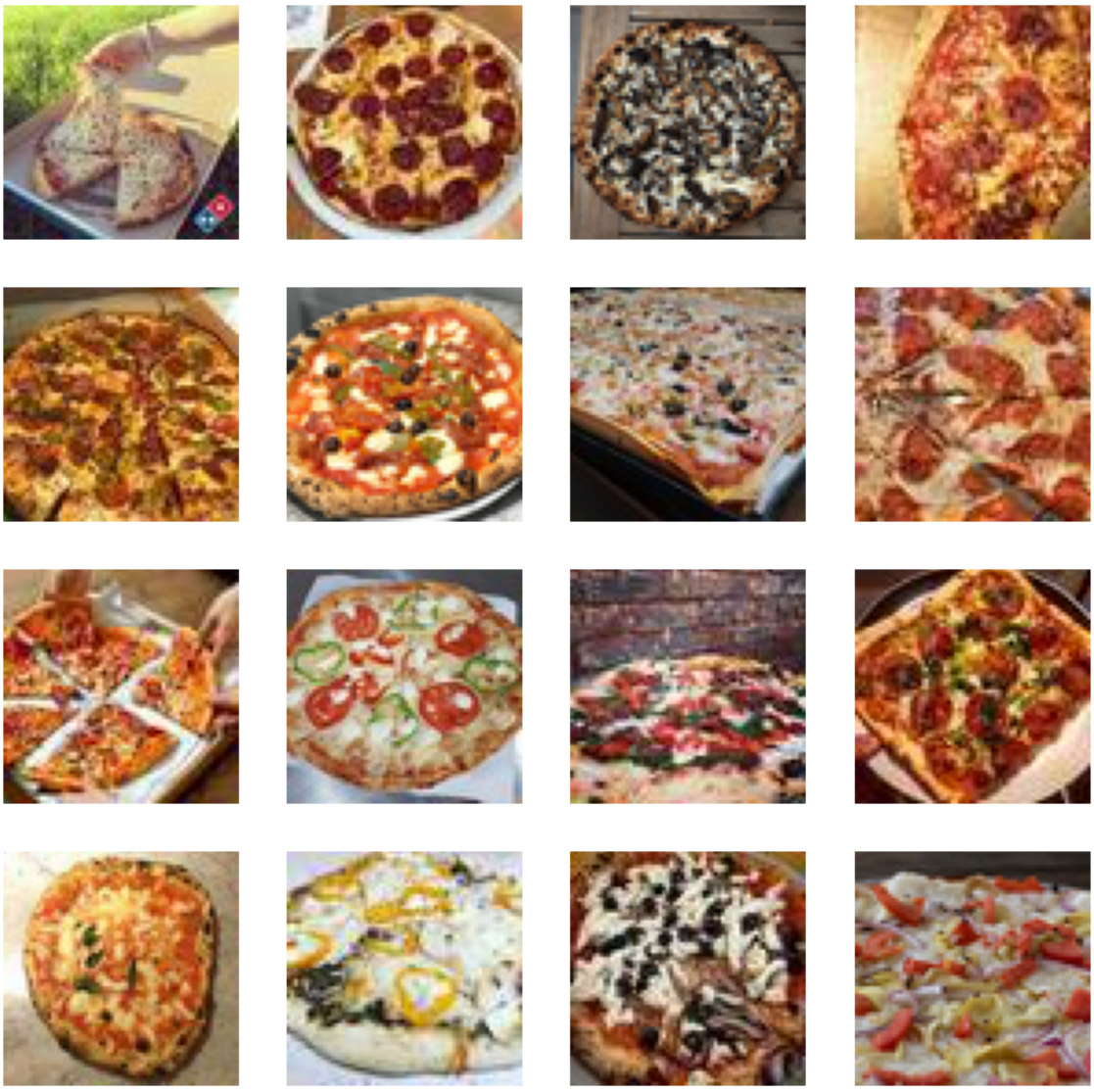
In [ ]:

```
#Fake images
plot_collage(train_dataset,device,type_of="fake",model=trained_generator)
```



In [ ]:

```
#Real Images
plot_collage(train_dataset,device,type_of="real",model=None)
```

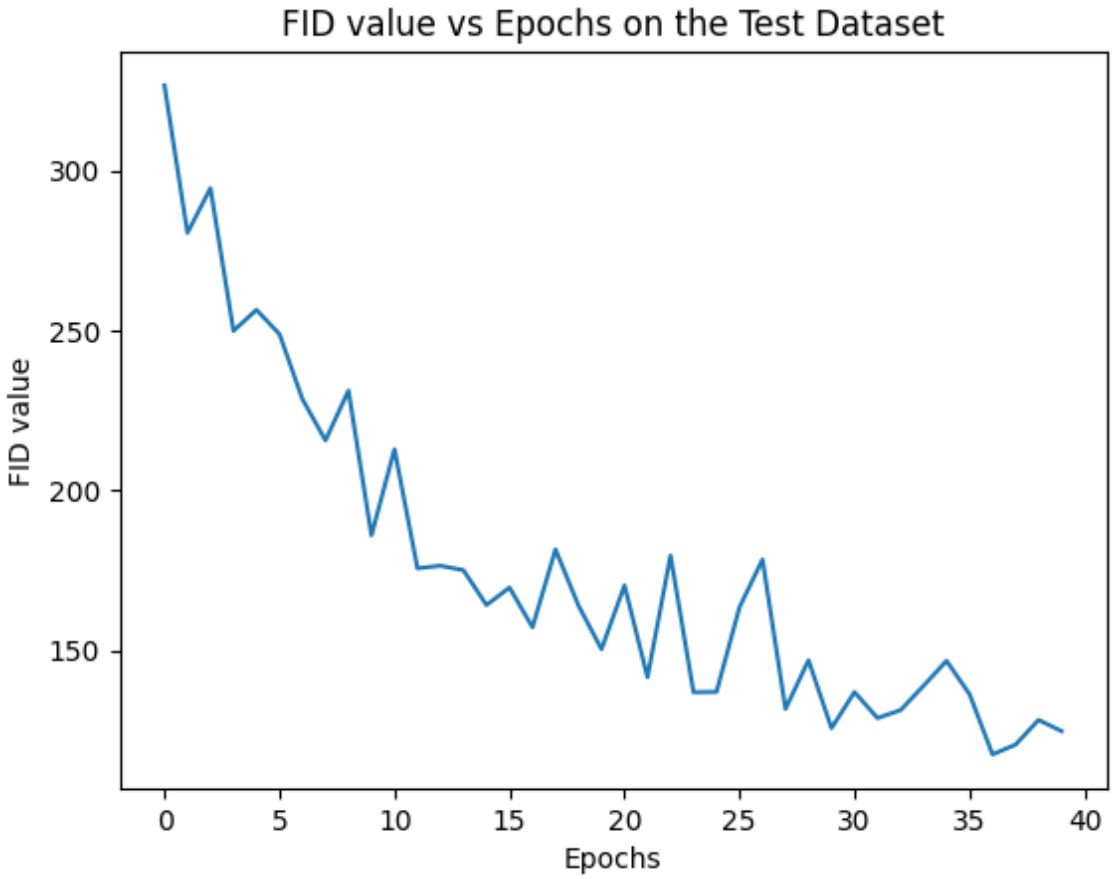


In [ ]:

```
plt.xlabel("Epochs")
plt.ylabel("FID value")
plt.title("FID value vs Epochs on the Test Dataset")
plt.plot(fid_score)
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x7f9a983d7c10>]
```

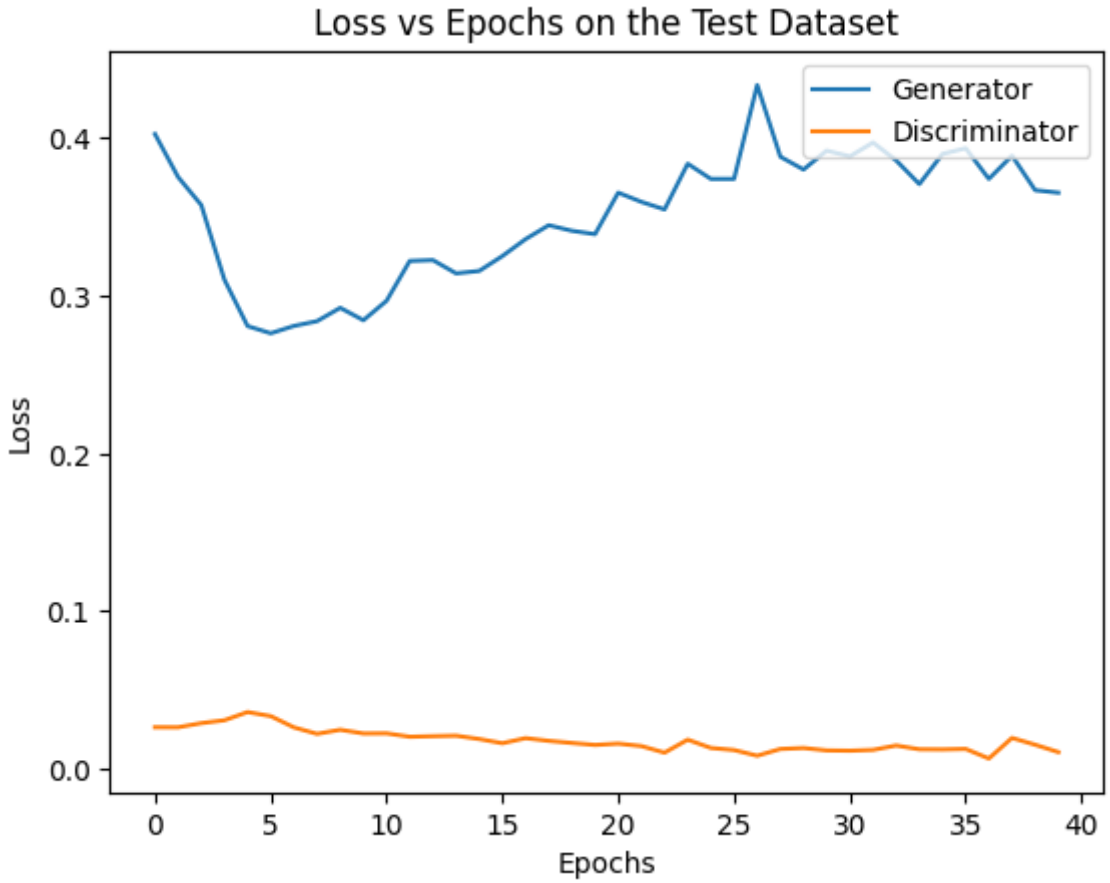


In [ ]:

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs on the Test Dataset") #This should be training dataset
plt.plot(train_lossesG,label="Generator")
plt.plot(train_lossesD, label="Discriminator")
plt.legend(loc = "upper right")
```

Out[35]:

```
<matplotlib.legend.Legend at 0x7f9a9804a460>
```



In [ ]:

In [1]:

```
pip install pytorch_fid
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) h
ttps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.p
kg.dev/colab-wheels/public/simple/)
Requirement already satisfied: pytorch_fid in /usr/local/lib/python3.9/di
st-packages (0.3.0)
Requirement already satisfied: torch>=1.0.1 in /usr/local/lib/python3.9/d
ist-packages (from pytorch_fid) (2.0.0+cu118)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-pac
kages (from pytorch_fid) (1.10.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-pa
ckages (from pytorch_fid) (8.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-pac
kages (from pytorch_fid) (1.22.4)
Requirement already satisfied: torchvision>=0.2.2 in /usr/local/lib/pytho
n3.9/dist-packages (from pytorch_fid) (0.15.1+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-
packages (from torch>=1.0.1->pytorch_fid) (3.10.7)
Requirement already satisfied: sympy in /usr/local/lib/python3.9/dist-pac
kages (from torch>=1.0.1->pytorch_fid) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.9/dist-
packages (from torch>=1.0.1->pytorch_fid) (3.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-pa
ckages (from torch>=1.0.1->pytorch_fid) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.9/
dist-packages (from torch>=1.0.1->pytorch_fid) (2.0.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python
3.9/dist-packages (from torch>=1.0.1->pytorch_fid) (4.5.0)
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packa
ges (from triton==2.0.0->torch>=1.0.1->pytorch_fid) (16.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.9/dist-pac
kages (from triton==2.0.0->torch>=1.0.1->pytorch_fid) (3.25.2)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-
packages (from torchvision>=0.2.2->pytorch_fid) (2.27.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
9/dist-packages (from jinja2->torch>=1.0.1->pytorch_fid) (2.1.2)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
n3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fid) (202
2.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/li
b/python3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fi
d) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/d
ist-packages (from requests->torchvision>=0.2.2->pytorch_fid) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/py
thon3.9/dist-packages (from requests->torchvision>=0.2.2->pytorch_fid)
(1.26.15)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.9/d
ist-packages (from sympy->torch>=1.0.1->pytorch_fid) (1.3.0)
```

In [2]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import Dataset, DataLoader
from torchvision import models,datasets,transforms
from PIL import Image
from tqdm import tqdm

import os
import matplotlib.pyplot as plt
import numpy as np

from pytorch_fid.fid_score import calculate_activation_statistics,calculate_frechet_dist
from pytorch_fid . inception import InceptionV3
import shutil
```

In [3]:

```
class IndexedDataset(Dataset):
    def __init__(self, dir_path):
        self.dir_path = dir_path
        #The transforms that will be applied to each image
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
          ])
        #Saving all the image locations
        self.image_filenames = []
        for (dirpath, dirnames, filenames) in os.walk(dir_path):
            self.image_filenames += [os.path.join(dirpath, file) for file in filenames]

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

    def __getitem__(self, idx):
        img_name = self.image_filenames[idx] #Getting the name of the image
        image = Image.open(img_name).convert('RGB') #opening the image
        image = self.transform(image) #Applying the transforms to the image
        return image
```

In [4]:

```
#Inspired from https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix
def WGAN_Loss(prediction, target_is_real):
  if target_is_real:
    loss = -prediction.mean()
  else:
    loss = prediction.mean()

  return loss
```

In [5]:

```
#Inspired by https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.convt1 = nn.ConvTranspose2d( 100, 64 * 8, 4, 1, 0, bias=False)
        self.convt2 = nn.ConvTranspose2d(64 * 8, 64 * 4, 4, 2, 1, bias=False)
        self.convt3 = nn.ConvTranspose2d( 64 * 4, 64 * 2, 4, 2, 1, bias=False)
        self.convt4 = nn.ConvTranspose2d( 64 * 2, 64, 4, 2, 1, bias=False)
        self.convt5 = nn.ConvTranspose2d( 64, 3, 4, 2, 1, bias=False)
        self.bn1 = nn.BatchNorm2d(64 * 8)
        self.bn2 = nn.BatchNorm2d(64 * 4)
        self.bn3 = nn.BatchNorm2d(64*2)
        self.bn4 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(True)
        self.tanh = nn.Tanh()

    def forward(self, input):
        x = self.relu(self.bn1(self.convt1(input)))
        x = self.relu(self.bn2(self.convt2(x)))
        x = self.relu(self.bn3(self.convt3(x)))
        x = self.relu(self.bn4(self.convt4(x)))
        x = self.tanh(self.convt5(x))
        return x
```

In [6]:

```
#Inspired by https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, 4, 2, 1, bias=False)
        self.conv2 = nn.Conv2d(64, 64 * 2, 4, 2, 1, bias=False)
        self.conv3 = nn.Conv2d(64 * 2, 64 * 4, 4, 2, 1, bias=False)
        self.conv4 = nn.Conv2d(64 * 4, 64 * 8, 4, 2, 1, bias=False)
        self.conv5 = nn.Conv2d(64 * 8, 1, 4, 1, 0, bias=False)
        self.bn2 = nn.BatchNorm2d(64 * 2)
        self.bn3 = nn.BatchNorm2d(64 * 4)
        self.bn4 = nn.BatchNorm2d(64 * 8)
        self.relu = nn.LeakyReLU(0.2, inplace=True)

    def forward(self, input):
        x = self.relu(self.conv1(input))
        x = self.relu(self.bn2(self.conv2(x)))
        x = self.relu(self.bn3(self.conv3(x)))
        x = self.relu(self.bn4(self.conv4(x)))
        x = self.conv5(x)
        return x
```

In [7]:

```
#Initializing the weights
#From https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

In [8]:

```
#Function to train the model
def training(epochs,optimizerD,optimizerG,criterion,netG,netD,train_data_loader,device,b
  #Storing the losses of the Generator and Discriminator
  train_lossesG = []
  train_lossesD = []
  fid_score = []

  #Assigning labels to the real and fake images
  real_label = 1
  fake_label = 0

  for epoch in range(epochs):
    running_lossG = 0.0
    running_lossD = 0.0
    for i,data in enumerate(train_data_loader):
      #Part 1: Training the Discriminator
      #Part A: On the real images
      optimizerD.zero_grad()
      inputs_real = data.to(device) #Getting the real image
      num_images = data.shape[0] #Getting the batch size

      outputs_real = netD(inputs_real).view(-1) #Getting the predicted labels for the re
      loss_real = criterion(outputs_real, real_label) #Calculating the loss on real imag
      loss_real.backward()

      #Part B: On the fake images
      noise = torch.randn((num_images,100,1,1)).to(device) #Generating the loss
      inputs_fake = netG(noise) #Generating the fake images from this noise
      outputs_fake = netD(inputs_fake.detach()).view(-1) #Getting the predicted labels f
      loss_fake = criterion(outputs_fake, fake_label) #Calculating the loss on fake imag
      loss_fake.backward()

      optimizerD.step()
      running_lossD += loss_real.cpu().item() + loss_fake.cpu().item()

      clipping_thresh = 0.01
      #Weight clipping which enforces 1-Lipschitz constraint
      for p in netD.parameters():
        p.data.clamp_(-clipping_thresh, clipping_thresh)

      #Part 2: Training the Generator
      optimizerG.zero_grad()
      outputs_fake_G = netD(inputs_fake).view(-1) #Genrating the predicted labels for th
      loss_G = criterion(outputs_fake_G, real_label) #In the optimum case the fake image
      loss_G.backward()
      optimizerG.step()
      running_lossG += loss_G.cpu().item()

    print("[epoch: %d, batch: %5d] Discriminator loss: %.3f Generator loss: %.3f" % (epo
    train_lossesG.append(running_lossG/len(train_data_loader.dataset))
    train_lossesD.append(running_lossD/len(train_data_loader.dataset))
    fid = calc_fid(netG,device,m1,s1,model)
    fid_score.append(fid)
  return netG, train_lossesG, train_lossesD, fid_score
```

In [9]:

```
#Function to Plot a real image
def plot_image_real(dataset,index):
  img = dataset[index] #Getting the image
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  return ci.astype(int)  #returns the image
```

In [10]:

```
#Function to Plot a fake image
def plot_image_fake(netG,device):
  noise = torch.randn((1,100,1,1)).to(device)
  img = netG(noise).cpu().detach() #Getting the image
  img = img.squeeze()
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  return ci.astype(int)  #returns the image
```

In [11]:

```
#Function to save a fake image
def save_image_fake(img,i):
  im = Image.fromarray(img.astype(np.uint8))
  im.save("fake/"+str(i)+".jpeg")
```

In [12]:

```
#Function to calculate FID
def calc_fid(netG,device,m1,s1,model):
  #Generating 1000 fake images
  for i in range(1000):
    img = plot_image_fake(netG,device)
    save_image_fake(img,i)
  fake_paths = ["fake/" +str(i)+".jpeg" for i in range(0, 1000)] #Generating the path fo
  #Obtaining the feature encoding for the real test images
  m2 , s2 = calculate_activation_statistics(fake_paths, model,device = device)
  fid_value = calculate_frechet_distance(m1,s1,m2,s2) #Calculating FID value
  print("FID: "+str(fid_value))
  return fid_value
```

In [13]:

```
#Function to Plot 4x4 images
def plot_collage(dataset,device,type_of="real",model=None):
  images = []
  if type_of=="real": #If you want real image generated
    indices = np.random.randint(0,len(dataset),16) #Getting 16 random indices
    for i in indices:
      images.append(plot_image_real(dataset,i)) #Saving the 16 images

  elif type_of=="fake": #If you want fake image generated
    for i in range(16):
      images.append(plot_image_fake(model,device)) #Saving the 16 images

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

  ax11 = fig.add_subplot(4,4,1)
  ax12 = fig.add_subplot(4,4,2)
  ax13 = fig.add_subplot(4,4,3)
  ax14 = fig.add_subplot(4,4,4)
  ax21 = fig.add_subplot(4,4,5)
  ax22 = fig.add_subplot(4,4,6)
  ax23 = fig.add_subplot(4,4,7)
  ax24 = fig.add_subplot(4,4,8)
  ax31 = fig.add_subplot(4,4,9)
  ax32 = fig.add_subplot(4,4,10)
  ax33 = fig.add_subplot(4,4,11)
  ax34 = fig.add_subplot(4,4,12)
  ax41 = fig.add_subplot(4,4,13)
  ax42 = fig.add_subplot(4,4,14)
  ax43 = fig.add_subplot(4,4,15)
  ax44 = fig.add_subplot(4,4,16)

  ax11.axis('off')
  ax12.axis('off')
  ax13.axis('off')
  ax14.axis('off')
  ax21.axis('off')
  ax22.axis('off')
  ax23.axis('off')
  ax24.axis('off')
  ax31.axis('off')
  ax32.axis('off')
  ax33.axis('off')
  ax34.axis('off')
  ax41.axis('off')
  ax42.axis('off')
  ax43.axis('off')
  ax44.axis('off')

  ax11.imshow(images[0])
  ax12.imshow(images[1])
  ax13.imshow(images[2])
  ax14.imshow(images[3])
  ax21.imshow(images[4])
  ax22.imshow(images[5])
  ax23.imshow(images[6])
  ax24.imshow(images[7])
  ax31.imshow(images[8])
  ax32.imshow(images[9])
  ax33.imshow(images[10])
```

```
  ax34.imshow(images[11])
  ax41.imshow(images[12])
  ax42.imshow(images[13])
  ax43.imshow(images[14])
  ax44.imshow(images[15])
```

In [14]:

```
train_dataset = IndexedDataset("/content/drive/MyDrive/pizzas/train")
test_dataset = IndexedDataset("/content/drive/MyDrive/pizzas/eval")

train_dataloader = DataLoader(train_dataset,batch_size=16,num_workers=64)
test_dataloader = DataLoader(test_dataset,batch_size=16,num_workers=64)

print(len(train_dataloader))
print(len(test_dataloader))
```

```
514
63

/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:56
1: UserWarning: This DataLoader will create 64 worker processes in total.
Our suggested max number of worker in current system is 2, which is small
er than what this DataLoader is going to create. Please be aware that exc
essive worker creation might get DataLoader running slow or even freeze,
lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

In [15]:

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

Out[15]:

```
device(type='cuda')
```

In [16]:

```
netG = Generator().to(device)
netG.apply(weights_init)
netD = Discriminator().to(device)
netD.apply(weights_init)

optimizerG = torch.optim.Adam(netG.parameters(), lr=0.0001, betas = (0.5, 0.999))
optimizerD = torch.optim.Adam(netD.parameters(), lr=0.0001, betas = (0.5, 0.999))
criterion = WGAN_Loss
epochs = 40
```

In [17]:

```
#The number of parameters in the Generator
sum(p.numel() for p in netG.parameters() if p.requires_grad)
```

Out[17]:

```
3576704
```

In [18]:

```
#The number of parameters in the Discriminator
sum(p.numel() for p in netD.parameters() if p.requires_grad)
```

Out[18]:

```
2765568
```

In [19]:

```
#Creating an instance of the InceptionV3 model
dims = 2048
block_idx = InceptionV3.BLOCK_INDEX_BY_DIM[dims]
model = InceptionV3([block_idx]).to(device)
```

In [20]:

```
#Obtaining the feature encoding for the real test images
real_paths = test_dataset.image_filenames
m1 , s1 = calculate_activation_statistics(real_paths,model,device = device)
```

```
100%|██████████| 20/20 [00:08<00:00,  2.38it/s]
```

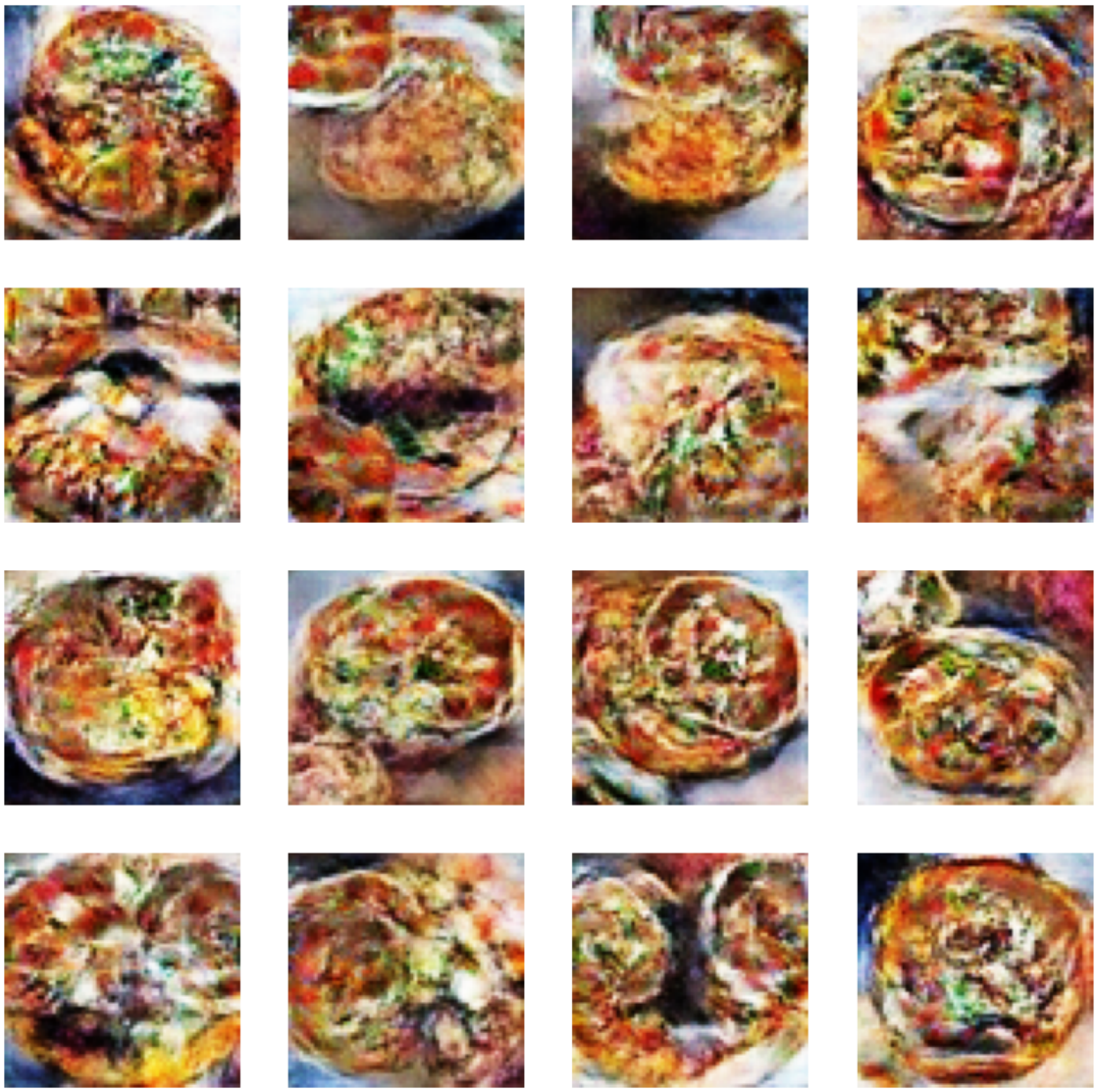
In [21]:

```
trained_generator,train_lossesG, train_lossesD,fid_score = training(epochs,optimizerD,op
```

```
[epoch: 1, batch:   514] Discriminator loss: -0.035 Generator loss: 0.
035

100%|██████████| 20/20 [00:04<00:00,  4.42it/s]

FID: 312.71542824865946
[epoch: 2, batch:   514] Discriminator loss: -0.702 Generator loss: 0.
467

100%|██████████| 20/20 [00:04<00:00,  4.27it/s]

FID: 319.2183011375573
[epoch: 3, batch:   514] Discriminator loss: -0.817 Generator loss: 0.
414

100%|██████████| 20/20 [00:04<00:00,  4.39it/s]

FID: 273.9275918819934
[epoch: 4, batch:   514] Discriminator loss: -0.733 Generator loss: 0.
308
```

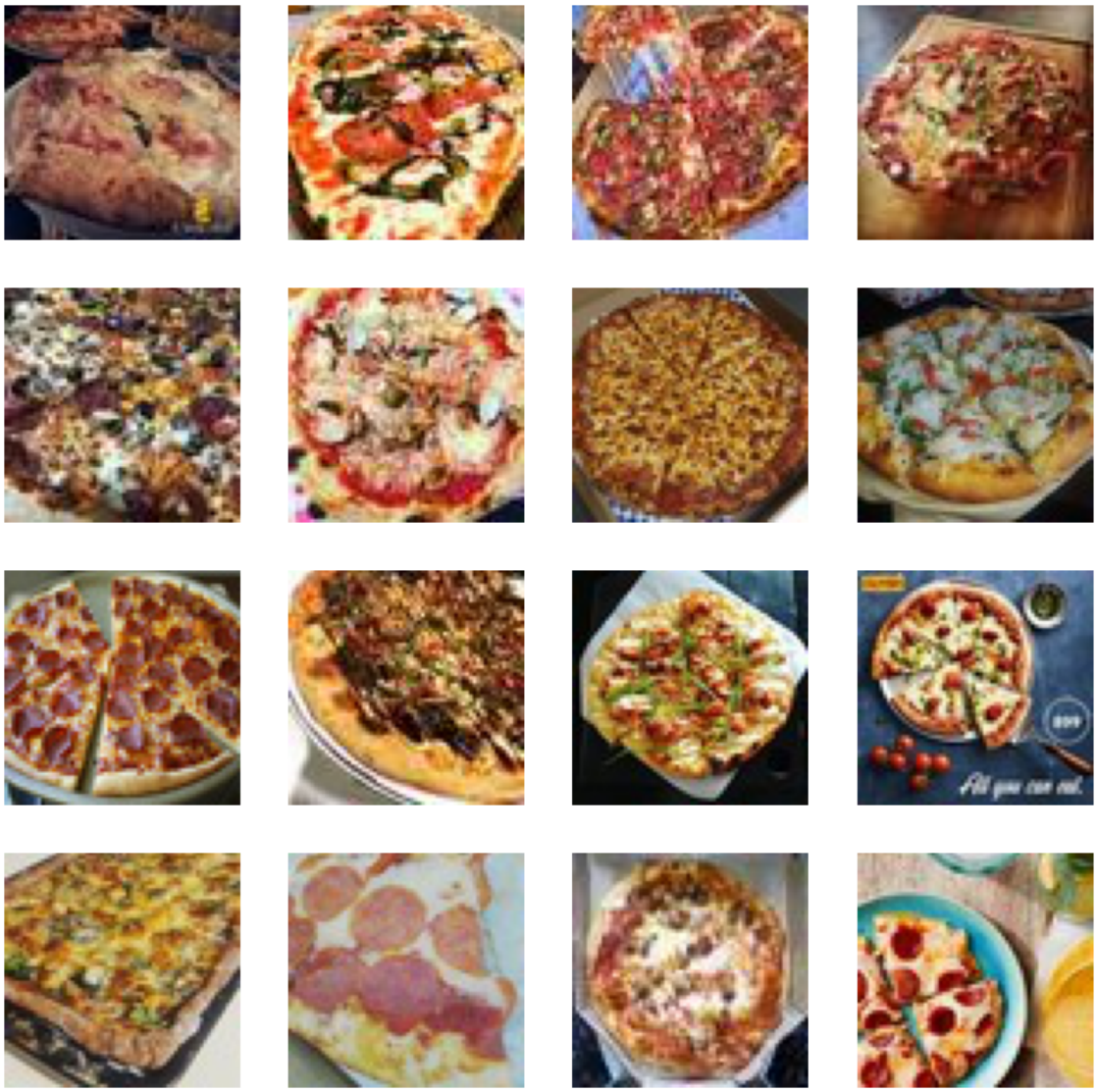
In [22]:

```
#Fake images
plot_collage(train_dataset,device,type_of="fake",model=trained_generator)
```



In [23]:

```
#Real Images
plot_collage(train_dataset,device,type_of="real",model=None)
```

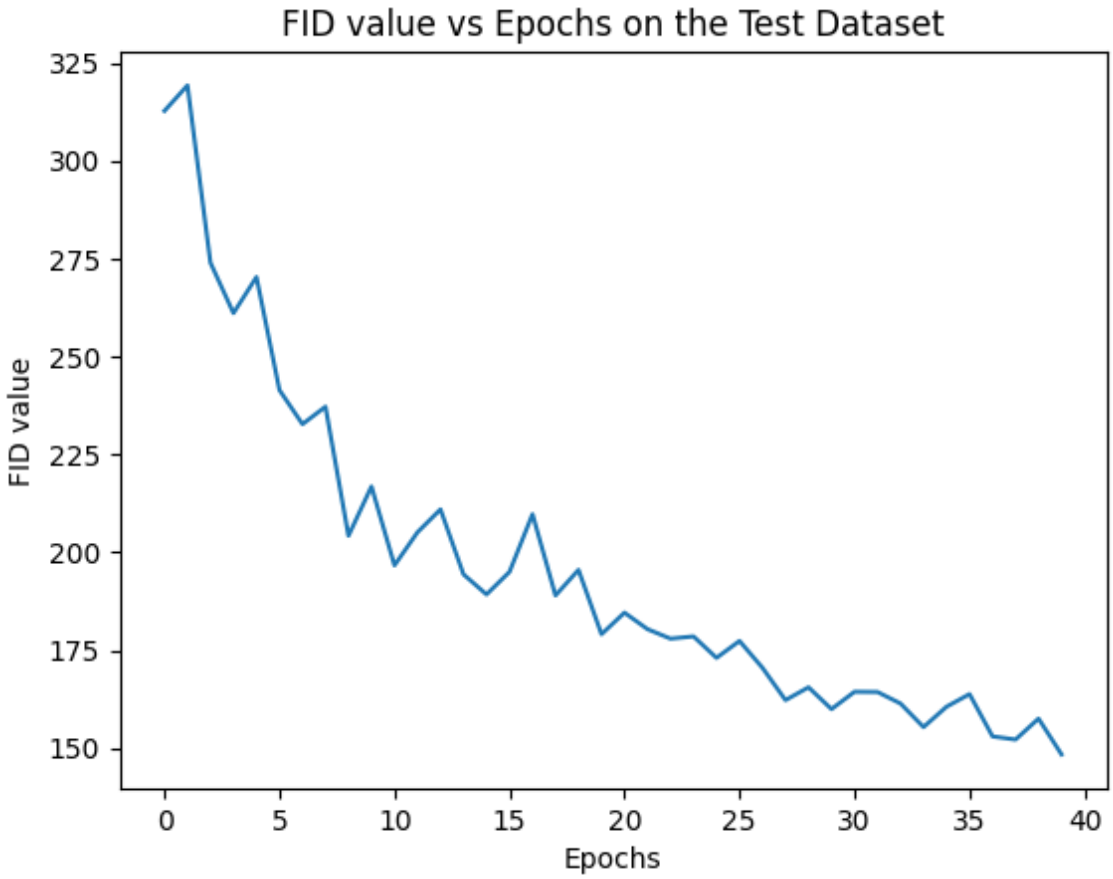


In [24]:

```
plt.xlabel("Epochs")
plt.ylabel("FID value")
plt.title("FID value vs Epochs on the Test Dataset")
plt.plot(fid_score)
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x7f53ba193880>]
```

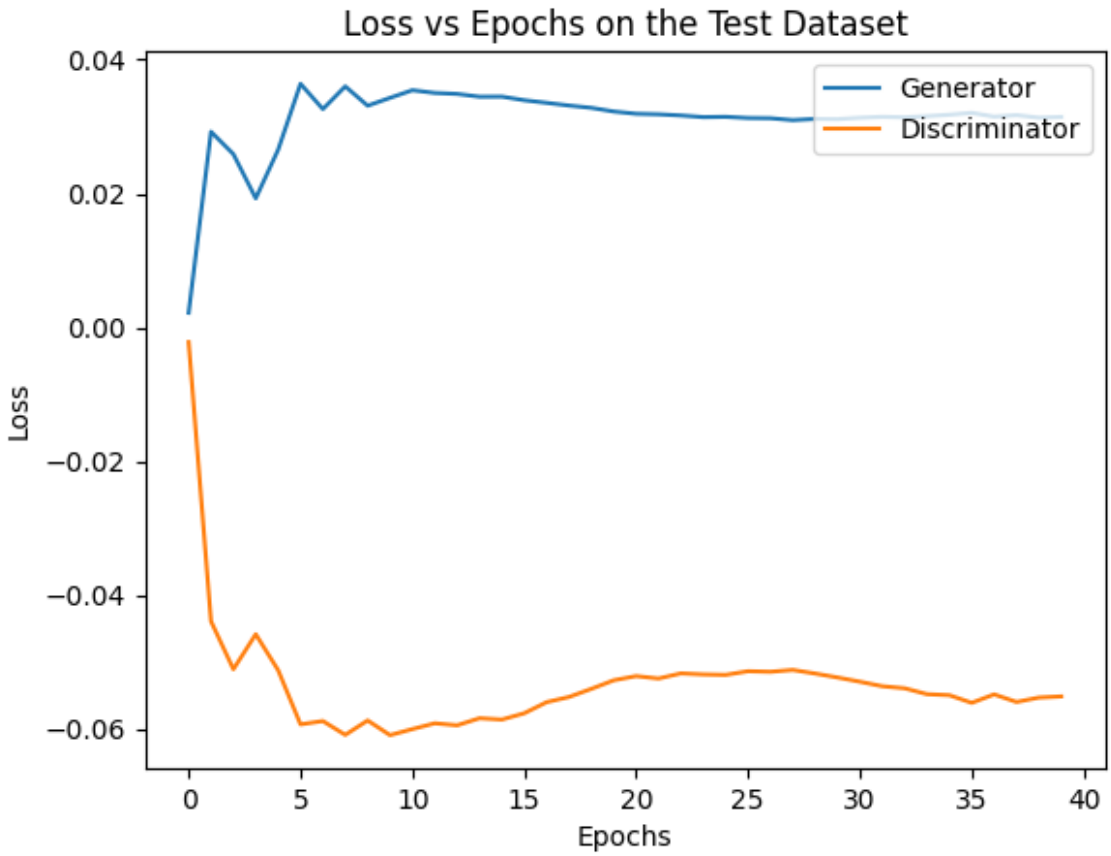


In [25]:

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss vs Epochs on the Test Dataset") #This should be training dataset
plt.plot(train_lossesG,label="Generator")
plt.plot(train_lossesD, label="Discriminator")
plt.legend(loc = "upper right")
```

Out[25]:

```
<matplotlib.legend.Legend at 0x7f53ba7fa610>
```



In [ ]: