In [2]:
```
! pip install torchgan

# Imports
import keras
import random
import numpy as np
import matplotlib.pyplot as plt

# Pytorch and Torchvision Imports
import torch
import torch.nn as nn
import torchvision
from torch.optim import Adam
import torch.nn as nn
import torch.utils.data as data
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torchvision.utils as vutils
from torchgan.metrics import *

# Torchgan Imports
import torchgan
from torchgan.models import *
from torchgan.losses import *
from torchgan.trainer import Trainer
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.d
ev/colab-wheels/public/simple/
Collecting torchgan
  Downloading torchgan-0.1.0-py3-none-any.whl (71 kB)
     |████████████████████████████████| 71 kB 6.9 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dis
t-packages (from torchgan) (1.21.6)
Requirement already satisfied: torch>=1.2 in /usr/local/lib/python3.
8/dist-packages (from torchgan) (1.12.1+cu113)
Requirement already satisfied: torchvision>=0.4 in /usr/local/lib/py
thon3.8/dist-packages (from torchgan) (0.13.1+cu113)
Requirement already satisfied: pillow in /usr/local/lib/python3.8/di
st-packages (from torchgan) (7.1.2)
Collecting wget
  Downloading wget-3.2.zip (10 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dis
t-packages (from torchgan) (1.7.3)
Requirement already satisfied: typing-extensions in /usr/local/lib/p
ython3.8/dist-packages (from torch>=1.2->torchgan) (4.1.1)
Requirement already satisfied: requests in /usr/local/lib/python3.8/
dist-packages (from torchvision>=0.4->torchgan) (2.23.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/p
ython3.8/dist-packages (from requests->torchvision>=0.4->torchgan)
(3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
python3.8/dist-packages (from requests->torchvision>=0.4->torchgan)
(2022.9.24)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.2
1.1 in /usr/local/lib/python3.8/dist-packages (from requests->torchv
ision>=0.4->torchgan) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python
3.8/dist-packages (from requests->torchvision>=0.4->torchgan) (2.10)
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=96
74 sha256=6ce70696767e06b89f1e5d78b5cb5ce281959c846e2f17978e360bbe53
bdd733
  Stored in directory: /root/.cache/pip/wheels/bd/a8/c3/3cf2c14a1837
a4e04bd98631724e81f33f462d86a1d895fae0
Successfully built wget
Installing collected packages: wget, torchgan
Successfully installed torchgan-0.1.0 wget-3.2
```

In [43]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [3]:
```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
BATCH_SIZE=64
# device_name="cpu"
device_name="cuda:0"
noise_dim=100
```

Loading the MINST dataset

In [4]:
```
dataset = dsets.MNIST(
    root="./mnist",
    train=True,
    transform=transforms.Compose(
        [
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
            transforms.Normalize(mean=(0.5,), std=(0.5,)),
        ]
    ),
    download=True,
)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=BATCH_
SIZE,num_workers=2)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyt
e.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyt
e.gz to ./mnist/MNIST/raw/train-images-idx3-ubyte.gz


Extracting ./mnist/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist/M
NIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyt
e.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyt
e.gz to ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz


Extracting ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist/M
NIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.
gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.
gz to ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz


Extracting ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist/MN
IST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.
gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.
gz to ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz


Extracting ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist/MN
IST/raw
```

In [5]:
```
# Plot some of the training images
real_batch = next(iter(dataloader))
plt.figure(figsize=(8, 8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(
    np.transpose(vutils.make_grid(real_batch[0][:64], padding=2, no
rmalize=True).cpu(), (1, 2, 0))
)
plt.show()
```

Part 1 - DCGAN Network

In [19]:
```
# Setting up the DCGAN model
#### In the imports we see that ADAM has been imported so using tha
t
dcgan_network = {
    "generator": {
            "name": DCGANGenerator,
            "args": {
            "step_channels": 32,
            "out_channels": 1,
            "encoding_dims": 100,
            "nonlinearity": nn.ReLU(),
            "last_nonlinearity": nn.Tanh(),
        },
        "optimizer": {"name": Adam, "args": {"lr": 0.0002, "betas":
(0.5, 0.999)}},
    },
    "discriminator": {
            "name": DCGANDiscriminator,
            "args": {
            "step_channels": 32,
            "in_channels": 1,
            "nonlinearity": nn.LeakyReLU(negative_slope=0.01),
            "last_nonlinearity": nn.LeakyReLU(negative_slope=0.01),
        },
        "optimizer": {"name": Adam, "args": {"lr": 0.0002, "betas":
(0.5, 0.999)}},
    },
}
## LR = 0.0002 was specified in that paper mentioned in question
```

In [20]:
```
# Defining the loss for DCGAN. The chosen Loss type is leasr square
method.

## TORCHGAN Documentation listing Different losses availible to use
-> https://torchgan.readthedocs.io/en/latest/modules/losses.html  #
###
lsgan_losses = [LeastSquaresGeneratorLoss(),LeastSquaresDiscriminat
orLoss()]
```

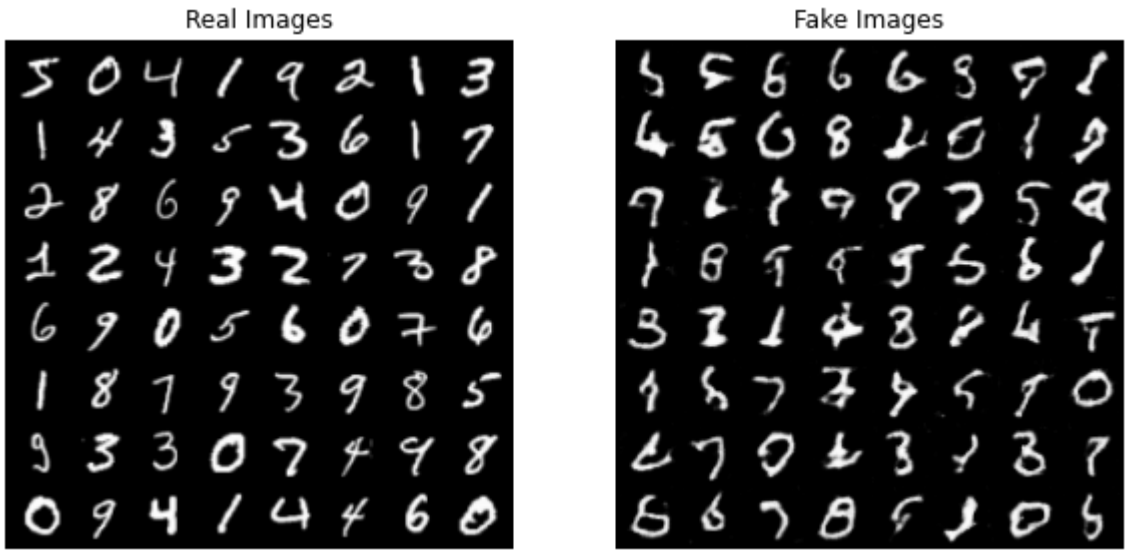
Training the DCGAN network.

In [21]:
```
# Training the DCGAN network
### -- Documentation -> https://torchgan.readthedocs.io/en/latest/modules/trainer.html -- ###
trainer = Trainer(dcgan_network,lsgan_losses,sample_size=64,epochs=10, device=torch.device(device_name))
## Change device to GPU on colab later
trainer(dataloader)
```

```
Saving Model at './model/gan0.model'
Epoch 1 Summary
Epoch time duration : 25.771738529205322
generator Mean Gradients : 0.8961336107026084
discriminator Mean Gradients : 9.28365720917641
Mean Running Discriminator Loss : 0.026963782051939573
Mean Running Generator Loss : 0.5104897319952816
Generating and Saving Images to ./images/epoch1_generator.png

Saving Model at './model/gan1.model'
Epoch 2 Summary
Epoch time duration : 26.745871543884277
generator Mean Gradients : 0.4808404021648147
discriminator Mean Gradients : 6.824486963757665
Mean Running Discriminator Loss : 0.016831896070868515
Mean Running Generator Loss : 0.5086588448902437
Generating and Saving Images to ./images/epoch2_generator.png

Saving Model at './model/gan2.model'
Epoch 3 Summary
Epoch time duration : 25.61440134048462
generator Mean Gradients : 0.3291051158949311
discriminator Mean Gradients : 6.119759897749864
Mean Running Discriminator Loss : 0.01299032666126376
Mean Running Generator Loss : 0.5073907406366541
Generating and Saving Images to ./images/epoch3_generator.png

Saving Model at './model/gan3.model'
Epoch 4 Summary
Epoch time duration : 25.75883674621582
generator Mean Gradients : 0.5201450960750504
discriminator Mean Gradients : 13.830062831404607
Mean Running Discriminator Loss : 0.032639496750154866
Mean Running Generator Loss : 0.4936792839552834
Generating and Saving Images to ./images/epoch4_generator.png

Saving Model at './model/gan4.model'
Epoch 5 Summary
Epoch time duration : 25.85092854499817
generator Mean Gradients : 0.9293769355901895
discriminator Mean Gradients : 18.721281537782076
Mean Running Discriminator Loss : 0.03503490104362356
Mean Running Generator Loss : 0.4812737105322926
Generating and Saving Images to ./images/epoch5_generator.png

Saving Model at './model/gan0.model'
Epoch 6 Summary
Epoch time duration : 25.677429676055908
generator Mean Gradients : 1.1896735132163563
discriminator Mean Gradients : 20.9113717344266
Mean Running Discriminator Loss : 0.03438656126915917
Mean Running Generator Loss : 0.4764829030976642
Generating and Saving Images to ./images/epoch6_generator.png

Saving Model at './model/gan1.model'
Epoch 7 Summary
```

```
Epoch time duration : 25.694812774658203
generator Mean Gradients : 1.411381671118484
discriminator Mean Gradients : 22.20448649733988
Mean Running Discriminator Loss : 0.033349327896967755
Mean Running Generator Loss : 0.4740057589268942
Generating and Saving Images to ./images/epoch7_generator.png

Saving Model at './model/gan2.model'
Epoch 8 Summary
Epoch time duration : 26.125723123550415
generator Mean Gradients : 1.6089352414978884
discriminator Mean Gradients : 23.272007133792158
Mean Running Discriminator Loss : 0.032392717069720474
Mean Running Generator Loss : 0.4724598320822583
Generating and Saving Images to ./images/epoch8_generator.png

Saving Model at './model/gan3.model'
Epoch 9 Summary
Epoch time duration : 26.254284143447876
generator Mean Gradients : 1.7633886005251647
discriminator Mean Gradients : 23.578841853175597
Mean Running Discriminator Loss : 0.03124242885310692
Mean Running Generator Loss : 0.47190348862157816
Generating and Saving Images to ./images/epoch9_generator.png

Saving Model at './model/gan4.model'
Epoch 10 Summary
Epoch time duration : 25.643065452575684
generator Mean Gradients : 1.877158661135371
discriminator Mean Gradients : 23.622914546643972
Mean Running Discriminator Loss : 0.030153500444421083
Mean Running Generator Loss : 0.47172904474251687
Generating and Saving Images to ./images/epoch10_generator.png

Training of the Model is Complete
```

In [35]:
```
# Visual comparison of the real and the fake images.

# Grab a batch of real images from the dataloader
real_batch = next(iter(dataloader))
device = torch.device(device_name)
# Plot the real images
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.axis("off")
plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:
64], padding=5, normalize=True).cpu(),(1, 2, 0),))

# Plot the fake images from the last epoch
plt.subplot(1, 2, 2)
plt.axis("off")
plt.title("Fake Images")
## -- format seen in verbose  -> Generating and Saving Images to ./
images/epoch4_generator.png -- ##
plt.imshow(plt.imread("./images/epoch10_generator.png"))
plt.show()
```



In [24]:
```
# plt.imread("./images/epoch5_generator.png")
# len(real_batch[0][0])
real_batch[0].shape
```

Out[24]:
```
torch.Size([64, 1, 32, 32])
```

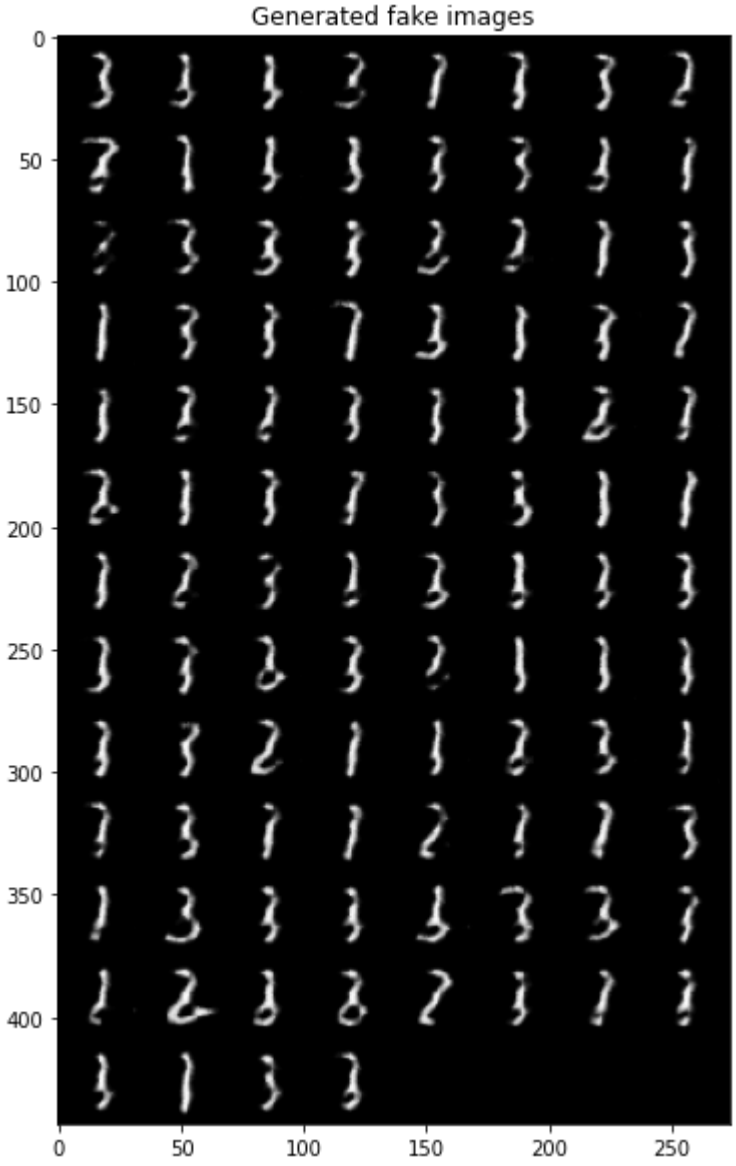
In [25]:
```
# x = torch.rand(100, noise_dim, 1, 1, device=device)
```

In [25]:
```

```

In [26]:
```
# x[0].shape
```

Accuracy of our trained DCGAN network

In [27]:
```
# Accuracy of Descriminator on fake images
# Generate 100 fake images for testing model accuracy

# Randomly initialize the 100 images
x = torch.rand(100, noise_dim, 1, 1, device=device)

# Pass the images to the generator to generate the random images
out_gen = trainer.generator.forward(x)

# Plot the random images generated
plt.figure(figsize=(10, 10))
plt.imshow(np.transpose(vutils.make_grid(out_gen.to(device), paddin
g=2, normalize=True).cpu(),(1, 2, 0),))
plt.title("Generated fake images")
plt.show()

# Get the discriminator prediction of these images
out_disc = trainer.discriminator.forward(out_gen)

# Compute accuracy based on these predictions
out_disc_np = out_disc.cpu().detach().numpy()
condition = out_disc_np < 0.5
accuracy = np.extract(condition,out_disc_np)
accuracy_fake = accuracy.size/len(x)
print("Accuracy of the model on fake images = " ,  accuracy_fake)
```




```
Accuracy of the model on fake images =  1.0
```

In [29]:
```
#Testing accuracy of benign inputs by choosing random samples from
the actual dataset

i = 0
# Load batches of 64 images 100 times to get accuracy
correct_detection = 0
while (i<100):
  real_batch = next(iter(dataloader))
  out_disc = trainer.discriminator.forward(real_batch[0].to(torch.d
evice(device_name)))
  out_disc_np = out_disc.cpu().detach().numpy()
  condition = out_disc_np > 0.5
  accuracy = np.extract(condition,out_disc_np)
  correct_detection += len(accuracy)
  i = i+1
accuracy_benign = correct_detection / (64*100)
print ("Accuracy of Descriminator on benign samples = " , accuracy_
benign)
```

```
Accuracy of Descriminator on benign samples =  0.046875
```

In [30]:
```
# Total accuracy of the discriminator

total_accuracy = (accuracy_fake+accuracy_benign)/2
print ("Accuracy of the Discriminator = ", total_accuracy)
```

```
Accuracy of the Discriminator =  0.5234375
```

In [31]:
```
# Computing the accuracy of Generator.
# Accuracy of the generator is computed using Inspection score whic
h is a score of how accurately can human eyes identify the images a
s real.

acc_generator = torchgan.metrics.ClassifierScore()
acc_generator.classifier=trainer.generator
score = acc_generator.calculate_score(out_gen)
print("Inspector score for DCGAN: " , score)
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:
135: UserWarning: Using 'weights' as positional parameter(s) is depr
ecated since 0.13 and will be removed in 0.15. Please use keyword pa
rameter(s) instead.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:
223: UserWarning: Arguments other than a weight enum or `None` for '
weights' are deprecated since 0.13 and will be removed in 0.15. The
current behavior is equivalent to passing `weights=Inception_V3_Weig
hts.IMAGENET1K_V1`. You can also use `weights=Inception_V3_Weights.D
EFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/inception_v3_googl
e-0cc3c7bd.pth" to /root/.cache/torch/hub/checkpoints/inception_v3_g
oogle-0cc3c7bd.pth


Inspector score for DCGAN:  tensor(1., device='cuda:0')
```

Part 2 - CGAN Network

In [33]:
```
# Setting up the CGAN model

## Reusign same code and architecture with addition of classes for
CGAN ###

cgan_network = {
    "generator": {
            "name": ConditionalGANGenerator,
            "args": {
            "step_channels": 32,
            "out_channels": 1,
            "encoding_dims": 100,
            "nonlinearity": nn.ReLU(),
            "last_nonlinearity": nn.Tanh(),
            "num_classes": 10
        },
        "optimizer": {"name": Adam, "args": {"lr": 0.0002, "betas":
(0.5, 0.999)}},
    },
    "discriminator": {
            "name": ConditionalGANDiscriminator,
            "args": {
            "step_channels": 32,
            "in_channels": 1,
            "nonlinearity": nn.LeakyReLU(negative_slope=0.01),
            "last_nonlinearity": nn.LeakyReLU(negative_slope=0.01),
            "num_classes": 10
        },
        "optimizer": {"name": Adam, "args": {"lr": 0.0002, "betas":
(0.5, 0.999)}},
    },
}
```

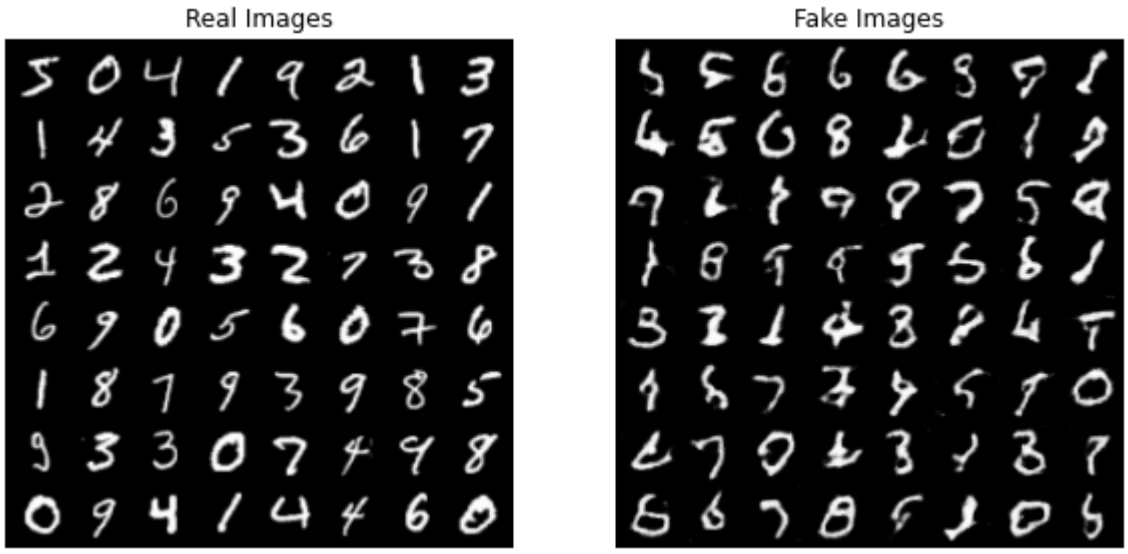
Training the CGAN network.

In [34]:
```
# Training the CGAN network
## Reusign same code and architecture with addition of classes for
CGAN ###
trainer_cgan = Trainer(dcgan_network,lsgan_losses,sample_size=64,ep
ochs=10, device=torch.device(device_name))
trainer_cgan(dataloader)
```

```
Saving Model at './model/gan0.model'
Epoch 1 Summary
Epoch time duration : 26.11587905883789
generator Mean Gradients : 2.230824444541612
discriminator Mean Gradients : 21.358279390272276
Mean Running Discriminator Loss : 0.04177248262629501
Mean Running Generator Loss : 0.48929661639463673
Generating and Saving Images to ./images/epoch1_generator.png

Saving Model at './model/gan1.model'
Epoch 2 Summary
Epoch time duration : 25.487582445144653
generator Mean Gradients : 2.324126114987307
discriminator Mean Gradients : 32.28240288074277
Mean Running Discriminator Loss : 0.04459876470096004
Mean Running Generator Loss : 0.4647232866419086
Generating and Saving Images to ./images/epoch2_generator.png

Saving Model at './model/gan2.model'
Epoch 3 Summary
Epoch time duration : 25.54073739051819
generator Mean Gradients : 2.397076249104968
discriminator Mean Gradients : 34.405218951929214
Mean Running Discriminator Loss : 0.04294404247494357
Mean Running Generator Loss : 0.45773998807054
Generating and Saving Images to ./images/epoch3_generator.png

Saving Model at './model/gan3.model'
Epoch 4 Summary
Epoch time duration : 25.588478803634644
generator Mean Gradients : 2.5547603907150043
discriminator Mean Gradients : 35.56219498466236
Mean Running Discriminator Loss : 0.041261233851191466
Mean Running Generator Loss : 0.45487504352185965
Generating and Saving Images to ./images/epoch4_generator.png

Saving Model at './model/gan4.model'
Epoch 5 Summary
Epoch time duration : 25.6128146648407
generator Mean Gradients : 2.751664238956644
discriminator Mean Gradients : 36.04927268150096
Mean Running Discriminator Loss : 0.039600434168271706
Mean Running Generator Loss : 0.4541840295436413
Generating and Saving Images to ./images/epoch5_generator.png

Saving Model at './model/gan0.model'
Epoch 6 Summary
Epoch time duration : 26.578289031982422
generator Mean Gradients : 2.9578208510569404
discriminator Mean Gradients : 36.16481735878709
Mean Running Discriminator Loss : 0.038141835866910 99
Mean Running Generator Loss : 0.4542294938869208
Generating and Saving Images to ./images/epoch6_generator.png

Saving Model at './model/gan1.model'
Epoch 7 Summary
```

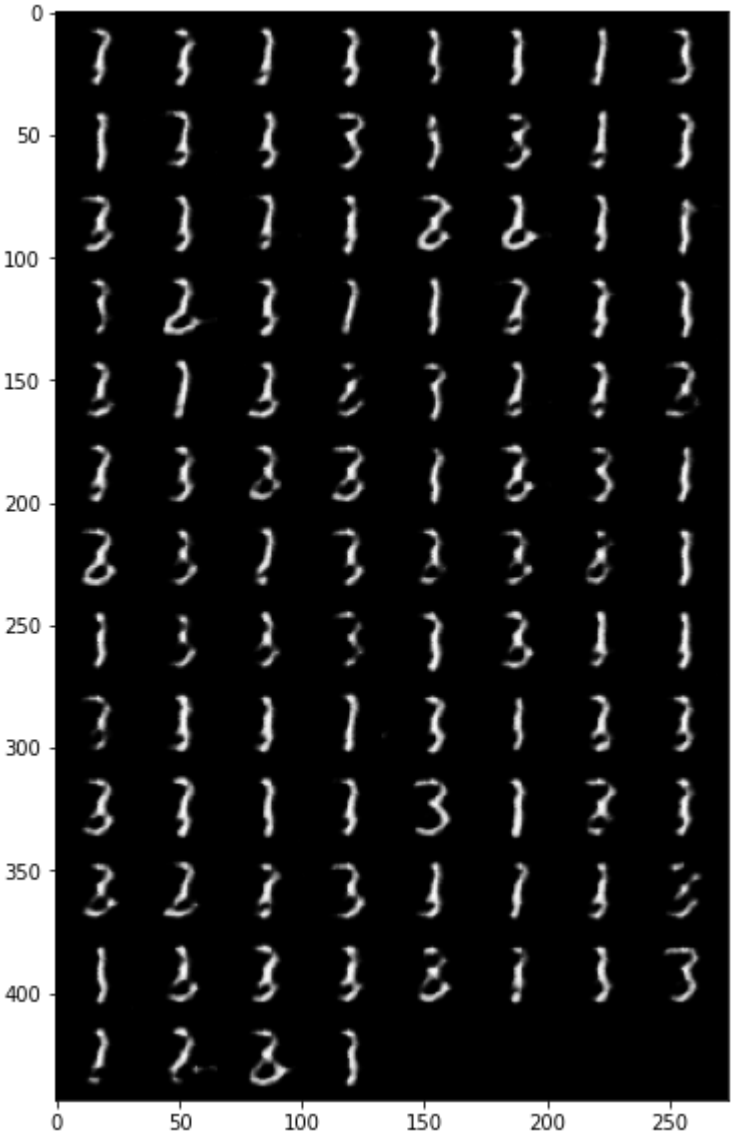
```
Epoch time duration : 25.485820293426514
generator Mean Gradients : 3.1311539742290244
discriminator Mean Gradients : 36.04297561782168
Mean Running Discriminator Loss : 0.036857989116421776
Mean Running Generator Loss : 0.45464747226273655
Generating and Saving Images to ./images/epoch7_generator.png

Saving Model at './model/gan2.model'
Epoch 8 Summary
Epoch time duration : 25.524399280548096
generator Mean Gradients : 3.273740740934719
discriminator Mean Gradients : 35.65717283701687
Mean Running Discriminator Loss : 0.03565887447773242
Mean Running Generator Loss : 0.45529749424460886
Generating and Saving Images to ./images/epoch8_generator.png

Saving Model at './model/gan3.model'
Epoch 9 Summary
Epoch time duration : 25.49646830558777
generator Mean Gradients : 3.3780838763027052
discriminator Mean Gradients : 35.30812230698212
Mean Running Discriminator Loss : 0.03458776797819507
Mean Running Generator Loss : 0.45609521223075494
Generating and Saving Images to ./images/epoch9_generator.png

Saving Model at './model/gan4.model'
Epoch 10 Summary
Epoch time duration : 25.73739504814148
generator Mean Gradients : 3.4791515825993895
discriminator Mean Gradients : 34.89228928058254
Mean Running Discriminator Loss : 0.033632918852252336
Mean Running Generator Loss : 0.4566837656671114
Generating and Saving Images to ./images/epoch10_generator.png

Training of the Model is Complete
```

In [36]:
```
# Visual comparison of the real and the fake images.
# Grab a batch of real images from the dataloader
real_batch = next(iter(dataloader))
# Plot the real images
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.axis("off")
plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:
64], padding=5, normalize=True).cpu(),(1, 2, 0),))

# Plot the fake images from the last epoch
plt.subplot(1, 2, 2)
plt.axis("off")
plt.title("Fake Images")
plt.imshow(plt.imread("./images/epoch10_generator.png"))
plt.show()
```



Accuracy of our trained CGAN network

In [39]:
```
# Accuracy of Descriminator on fake images
# Generate 100 fake images for testing model accuracy

# Randomly initialize the 100 images
x = torch.rand(100, noise_dim, 1, 1, device=device)
# Generate labels for the images
y = torch.randint(0,10,[100,],device=device)

# Pass the images to the generator to generate the random images
out_gen = trainer.generator.forward(x, y)

# Plot the random images generated
plt.figure(figsize=(10, 10))
plt.imshow(np.transpose(vutils.make_grid(out_gen.to(device), paddin
g=2, normalize=True).cpu(),(1, 2, 0),))
plt.show()


# Get the discriminator prediction of these images
out_disc = trainer.discriminator.forward(out_gen, y)

# Compute accuracy based on these predictions
out_disc_np = out_disc.cpu().detach().numpy()
condition = out_disc_np < 0.5
accuracy = np.extract(condition, out_disc_np)
accuracy_fake = accuracy.size/len(x)
print("Accuracy of the model on fake images = " ,  accuracy_fake)
```




```
Accuracy of the model on fake images =  1.0
```

In [40]:
```
#Testing accuracy of benign inputs by choosing random samples from
the actual dataset

i = 0
# Load batches of 64 images 100 times to get accuracy
correct_detection = 0
while (i<100):
  real_batch = next(iter(dataloader))
  out_disc = trainer.discriminator.forward(real_batch[0].to(torch.d
evice(device_name)), real_batch[1])
  out_disc_np = out_disc.cpu().detach().numpy()
  condition = out_disc_np > 0.5
  accuracy = np.extract(condition,out_disc_np)
  correct_detection += len(accuracy)
  i = i+1
accuracy_benign = correct_detection / (64*100)
print ("Accuracy of Descriminator on benign samples = " , accuracy_
benign)
```

```
Accuracy of Descriminator on benign samples =  0.046875
```

In [41]:
```
# Total accuracy of the discriminator
# taking average of both values
total_accuracy = (accuracy_fake+accuracy_benign)/2
print ("Accuracy of the Discriminator = ", total_accuracy)
```

```
Accuracy of the Discriminator =  0.5234375
```

In [42]:
```
# Computing the accuracy of Generator.
# Accuracy of the generator is computed using Inspection score whic
h is a score of how accurately can human eyes identify the images a
s real.

acc_generator = torchgan.metrics.ClassifierScore()
acc_generator.classifier=trainer_cgan.generator

score = acc_generator.calculate_score(out_gen)
print("Inspector score for DCGAN: " , score)
```

```
Inspector score for DCGAN:  tensor(1., device='cuda:0')
```

In [44]:
```
def preprocess(dir):
    # To deal with the error when there is [] in the path
    dir.replace('[','[[]')
    dir.replace(']','[]]')
    return dir

########################        YOUR CODE        ##################
########
your_ipynb_file_dir = '/content/drive/MyDrive/Purdue/SEM1/CS 529/As
signments/A4/Problem4_work_main_code_01.ipynb' # Example: '/content
/drive/MyDrive/ECE570/Assignment_02_Exercise.ipynb'
########################      END YOUR CODE      ##################
########

!jupyter nbconvert --to html '{preprocess(your_ipynb_file_dir)}'
```

```
[NbConvertApp] WARNING | pattern '/content/drive/MyDrive/Purdue/SEM1
/CS 529/Assignments/A4/Problem4_work_main_code_01.ipynb' matched no
files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELE
ASES.

Options
=======
The options below are convenience aliases to configurable class-opti
ons,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an e
rror and include the error message in the cell output (the default b
ehaviour is to abort conversion). This flag is only relevant if '--e
xecute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting note
book with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConve
rtApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
```

```
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConve
rtApp.export_format=notebook --FilesWriter.build_directory= --ClearO
utputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --T
emplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --
TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN',
'ERROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'not
ebook', 'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path
for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a t
ime.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each
notebook. To recover
                                    previous default behaviour (output
```

```
ting to the current
                                    working directory) use . as the fl
ag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url poin
ting to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path
to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory o
f the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#r
eveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default forma
t (probably HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex',
'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. L
aTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic' a
nd 'full'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynoteboo
k.ipynb

            You can also pipe the output to stdout, rather than a fi
le
```

```
                > jupyter nbconvert mynotebook.ipynb --stdout

                PDF is generated via latex

                > jupyter nbconvert mynotebook.ipynb --to pdf

                You can get (and serve) a Reveal.js-powered slideshow

                > jupyter nbconvert myslides.ipynb --to slides --post se
            rve

                Multiple notebooks can be given at the command line in a
            couple of
                different ways:

                > jupyter nbconvert notebook*.ipynb
                > jupyter nbconvert notebook1.ipynb notebook2.ipynb

                or you can specify the notebooks list in a config file,
            containing::

                    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

                > jupyter nbconvert --config mycfg.py

            To see all available configurables, use `--help-all`.
```

In [ ]: