Assignment Information

Field	Details
Name	Pros Loung
Course	AAI-521 Applied Computer Vision for AI
Assignment	6.1 - GAN Models
GitHub Repository	https://github.com/ploung/AAI_521_ComputerVision_Module6.git

**References: - https://www.tensorflow.org/datasets/keras_example

Part 1- Digit generator

In [2]:
```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time
import tensorflow as tf
import tensorflow_datasets as tfds

from IPython import display
```

In [5]:
```
#a- Reading data
import tensorflow as tf

# Load the mnist dataset from tf.keras
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()

# Print the shape of the training images and labels
print(f"Shape of train_images: {train_images.shape}")
print(f"Type of train_images: {train_images.dtype}")
print(f"Shape of train_labels: {train_labels.shape}")

# Reshape and normalize the images
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float

# Print the new shape of the training images
print(f"\nNew shape of train_images: {train_images.shape}")
print(f"New type of train_images: {train_images.dtype}")

train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1] by
BUFFER_SIZE = 60000
BATCH_SIZE = 256

# Batch and shuffle the data
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZ
```

```
Shape of train_images: (60000, 28, 28)
Type of train_images: uint8
Shape of train_labels: (60000,)

New shape of train_images: (60000, 28, 28, 1)
New type of train_images: float32
```

In [6]:
```
# Creating the generative model function
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,))) # 7*7*256
    model.add(layers.BatchNormalization()) # Normalize the activations of the previ
    model.add(layers.LeakyReLU()) # Activation function

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256) # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', u
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', us
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use
    assert model.output_shape == (None, 28, 28, 1)
```

```
    return model
```

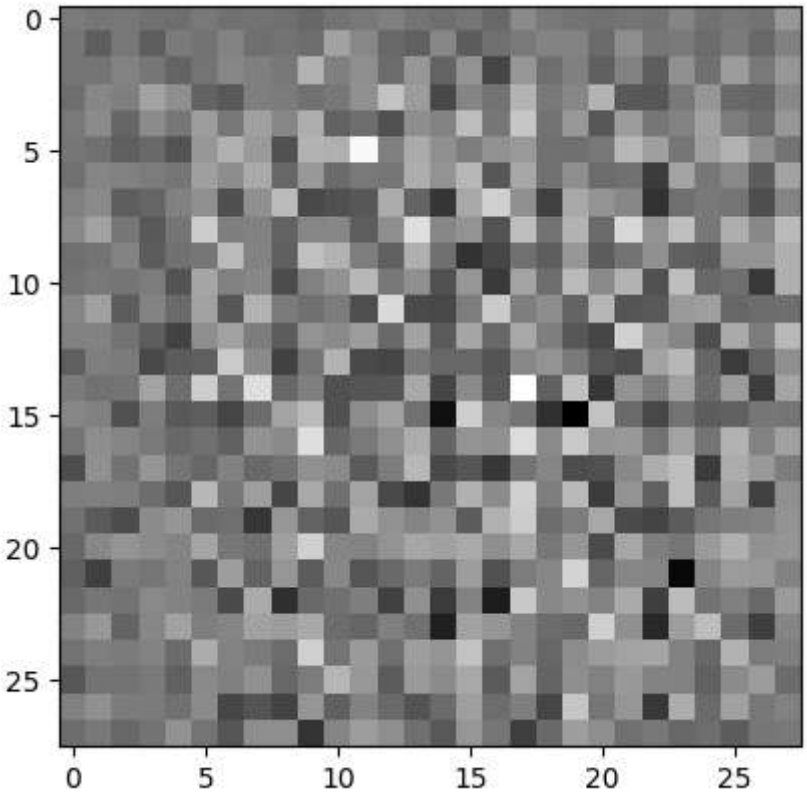
In [7]:
```
# Use the generator model which is not trained yet to create an image
generator = make_generator_model() # Call the function here

noise = tf.random.normal([1, 100])  #This creates the noise
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
c:\Users\Loung\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:95: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Out[7]: <matplotlib.image.AxesImage at 0x22633414410>



In [8]:
```
# Print the shape and values of the noise and generated image
print(f"Noise shape: {noise.shape}")
print(noise[0, :10])  # Print first 10 values of the noise vector
print(f"\nGenerated image shape: {generated_image.shape}")
print(generated_image[0, :, :, 0])  # Print a slice of the generated image
```

```
Noise shape: (1, 100)
tf.Tensor(
[-0.70617163 -2.303288    0.00804554  0.65766406  0.5829907  -0.01637366
 -0.74897283 -0.14905055 -1.3065504  -0.16746531], shape=(10,), dtype=float32)

Generated image shape: (1, 28, 28, 1)
tf.Tensor(
[[ 1.29587250e-03 -1.05199299e-03 -5.59672539e-04 -3.76481144e-03
  -5.21604763e-03  7.43262237e-04 -4.01080446e-03 -2.72726174e-03
  -4.51503042e-03 -8.86405632e-03 -2.87012523e-03 -2.99190637e-04
   4.48864745e-03 -2.06396054e-03 -5.35070011e-03 -1.59372529e-03
  -7.80910626e-03  1.00859189e-02  8.35723244e-04 -2.24940106e-03
  -3.36319720e-03 -1.99445710e-03 -3.10740597e-03  7.35094538e-03
  -2.79301824e-03  4.89060767e-05 -2.70618242e-03  1.76661126e-02]
 [-1.82912580e-03 -1.43200792e-02  2.58823857e-05 -1.47101535e-02
   1.31852273e-03 -3.06532276e-03  6.25269627e-03 -4.82290378e-03
  -2.93353619e-03 -5.82865439e-03  2.34378036e-02  8.81427247e-03
  -8.36085808e-03 -1.29865622e-02  8.08811653e-03 -1.50761139e-02
  -4.30556852e-03  1.15827331e-03  5.95201412e-03  6.46042777e-03
  -7.57854432e-03  1.19792949e-02 -1.29536772e-03  5.52958692e-04
  -6.53422764e-03  1.46899722e-03 -7.85809383e-03  4.63354960e-03]
 [-1.70897995e-03 -1.52088387e-03  6.81370590e-03 -5.72568562e-04
  -1.13257710e-02 -2.47814809e-03  8.29551462e-03  3.90319200e-03
  -4.47314233e-05  3.18421125e-02  4.46692249e-03  1.25319231e-02
  -6.54583611e-03  1.73192304e-02 -1.12782894e-02  1.54752443e-02
  -2.78671533e-02  1.64214093e-02 -5.24212467e-03  4.14053677e-03
  -1.34025719e-02  5.12027275e-03 -1.42988516e-02  1.37745859e-02
  -4.74306382e-03  1.96437072e-02  4.23237012e-04  1.71031095e-02]
 [-5.03391586e-03  8.44989996e-03  3.57950712e-03  2.42103506e-02
   1.28596574e-02 -1.16427457e-02 -1.76116675e-02  3.65197286e-03
   1.89237739e-03 -3.67996050e-03  5.08315861e-05  1.30483964e-02
   4.16957624e-02  2.00050473e-02 -2.56986991e-02  6.86873635e-03
  -2.75053177e-03  3.30238342e-02  1.14263722e-03  8.10600445e-03
   3.32601964e-02 -1.70141887e-02 -1.82938594e-02  4.96998837e-04
   1.68699846e-02 -8.48774891e-03 -1.07096443e-02  9.42462403e-03]
 [ 1.03477226e-03  1.10207992e-02 -9.84692015e-03  1.06522534e-02
  -8.68000090e-04  2.02894285e-02 -3.03825364e-04  2.23189313e-02
   8.10132176e-03  3.01338211e-02 -1.19864857e-02 -5.47555601e-03
  -2.18250006e-02  1.21849421e-02  6.15933677e-03  3.83523218e-02
  -8.96623707e-04  4.32991348e-02 -3.15649202e-03  1.74907651e-02
  -1.78646967e-02  1.98222194e-02 -7.89065729e-04  5.90577861e-03
   2.34218705e-02  2.03350373e-03 -6.29590685e-03  1.75494701e-02]
 [-9.31275892e-04 -4.06915834e-03 -1.39964037e-02 -7.73556577e-03
  -2.07943842e-02  1.68696456e-02  3.15481871e-02  1.93378739e-02
  -2.09928583e-02  3.25422026e-02  3.09416093e-02  7.13183433e-02
   4.19576745e-03  2.83003356e-02  1.29506160e-02  3.18154250e-03
   1.43477935e-02  1.86200589e-02 -3.20357061e-03 -4.40536719e-03
  -1.07911183e-04  3.47507000e-02  2.16069762e-02 -1.06241519e-03
   2.15640794e-02  3.22659798e-02  1.27784507e-02 -2.22615059e-03]
 [-3.32172122e-03  7.65132578e-03  4.60903905e-03  2.24205828e-03
  -1.23311696e-03  2.14323830e-02  1.17148506e-02  2.73978226e-02
  -9.34531540e-03  1.82216596e-02 -7.21319858e-03  1.17135956e-03
  -1.02826068e-03  3.01047899e-02  1.46610821e-02  3.47753614e-02
  -1.83911622e-02  2.60984227e-02 -4.24023904e-03  1.22942859e-02
  -6.60657929e-03 -3.18473368e-03 -3.30335349e-02  2.38131974e-02
  -9.51598515e-04  9.20303166e-03 -1.52600612e-02  2.40111649e-02]
```

```
[ 4.70294338e-03  1.06206555e-02 -1.47878034e-02 -1.03519838e-02
  4.73436248e-03  1.33620687e-02 -2.25190967e-02  1.30253304e-02
  3.72809321e-02 -2.64717564e-02 -2.10135449e-02 -1.78341996e-02
  2.88162492e-02 -1.10281017e-02 -3.65799479e-02  2.73430664e-02
  4.85910438e-02  1.32709388e-02 -3.07953395e-02  2.66647637e-02
  1.54329157e-02  8.13276134e-03 -3.89777347e-02 -5.08735329e-03
  1.12909393e-03 -2.14034808e-03 -2.32833233e-02  6.90941140e-03]
[ 1.00845285e-02  2.30957605e-02 -5.67783485e-04 -1.72680374e-02
 -1.95895135e-03  4.72353697e-02  3.52875772e-03  5.59153920e-03
 -1.17933629e-02  7.54348980e-03  8.15962348e-03 -1.45717673e-02
  1.27586182e-02  5.71572334e-02  8.42508767e-03  1.52731743e-02
 -2.03924999e-02  3.84927988e-02 -3.11941188e-03  3.26856114e-02
 -7.32723111e-03  5.29173501e-02  1.50650172e-02  3.95275727e-02
 -1.14747789e-04  3.00339479e-02  8.84971209e-03  3.71358097e-02]
[-4.91567003e-03 -2.07380275e-03  6.21137396e-03 -1.57293621e-02
 -3.58529994e-03 -7.02084391e-04  3.66136432e-02  6.04240922e-03
 -1.33911651e-02  4.01464738e-02  3.33340950e-02  3.16182501e-03
 -1.27697121e-02  3.11364550e-02 -4.73591173e-03 -3.87291722e-02
 -2.81374864e-02 -3.22666741e-03 -1.44871548e-02  1.68373920e-02
 -1.53752891e-02 -3.09212529e-03  1.36231408e-02 -1.33572202e-02
 -1.61293782e-02  1.54686570e-02  1.51591869e-02  3.08593493e-02]
[-2.68349331e-03  3.43888532e-04 -2.04764144e-03  1.97161688e-03
 -2.02096533e-02  2.56539881e-02  5.51259052e-03  8.67790263e-03
 -2.45090090e-02  4.69309231e-03  1.42516969e-02  3.54720578e-02
  1.28684565e-04  1.32159684e-02 -2.16700844e-02  1.64091382e-02
 -3.47576030e-02  2.44059525e-02 -5.93442284e-03  3.65585908e-02
  9.84472409e-03  3.25505324e-02 -2.17779297e-02  3.88062485e-02
 -8.20697006e-03 -5.21692447e-03 -3.56483907e-02  3.10001709e-02]
[ 7.82836881e-03  2.23699678e-02 -1.49515308e-02  6.70316722e-03
 -7.58919446e-03  2.30125729e-02 -1.86380222e-02  3.33678722e-02
  1.31827546e-03 -4.59499843e-03  2.15608068e-03 -2.28519235e-02
  5.43870069e-02 -2.49705035e-02 -2.67685968e-02  3.33487545e-03
  4.62053157e-02  3.64415604e-03  1.12803448e-02 -1.14832763e-02
  3.45710739e-02 -1.98321585e-02 -1.71278380e-02  1.88033730e-02
  2.17092279e-02 -9.52034071e-03 -5.99131314e-03 -5.04998816e-03]
[ 4.77697747e-03  5.35683800e-03 -3.12065589e-03 -1.47969602e-02
 -2.95144152e-02  1.23339472e-02  2.33378317e-02  2.77005299e-03
 -1.49215851e-02  1.55193349e-02  9.92365647e-03  2.05602236e-02
 -8.32768995e-03  2.36957110e-02 -1.60620306e-02  2.52229832e-02
 -8.79054330e-03  2.66766436e-02  3.99373192e-03 -1.87018998e-02
 -2.76761912e-02  4.93475497e-02  1.77731849e-02  7.41253234e-03
 -2.32009478e-02  2.77789142e-02  2.87128496e-03  3.62228490e-02]
[-1.37711391e-02  3.71076679e-03  1.47389120e-03 -2.58780625e-02
 -1.55896246e-02 -1.41812153e-02  4.56367731e-02  1.04382169e-02
 -3.04959826e-02 -1.46246783e-03  3.33958492e-02 -2.59387866e-02
 -2.85469275e-02  8.27630807e-04 -9.56530962e-03 -9.76884831e-03
 -2.03246213e-02  8.61128047e-03  1.53494356e-02  8.39655986e-04
 -1.91125702e-02 -2.37753782e-02  2.43828483e-02  3.55817117e-02
 -1.00912815e-02 -3.29994224e-02 -1.11160856e-02  1.34277930e-02]
[ 1.93629717e-03 -4.15866543e-03 -2.47566798e-03  2.39051618e-02
 -5.50722750e-03  4.78054397e-02 -2.46728724e-03  5.63291945e-02
 -9.11319815e-03  4.31959517e-03 -2.20230408e-02 -1.96702331e-02
 -1.88558698e-02  2.70684958e-02 -2.78525408e-02  1.02709951e-02
 -1.78895108e-02  7.51214102e-02 -1.23371677e-02  4.05552536e-02
 -3.56064253e-02  1.53457737e-02  2.15912308e-03  2.23894678e-02
 -6.80551864e-03  2.00730208e-02 -3.23857889e-02  2.50173397e-02]
```

```
[ 8.06587469e-03  4.97019058e-03 -2.11767741e-02  2.61313142e-03
 -1.65393241e-02 -1.44706974e-02 -2.84707323e-02 -6.00110460e-03
  2.45542005e-02  3.65697481e-02 -2.11912282e-02  1.14648389e-02
  2.26975027e-02 -3.41841881e-03 -5.78753389e-02  4.75343727e-02
  7.26688979e-03 -1.52720383e-03 -3.96744944e-02 -6.70345426e-02
  4.10303921e-02 -7.52581656e-03 -2.65417844e-02 -3.14970338e-03
 -1.52407354e-02 -1.26522444e-02 -8.89020681e-04 -1.97654450e-03]
[-1.12365757e-03  1.20614963e-02  7.10469531e-03  1.23264058e-03
 -7.68260239e-03 -4.28951904e-03 -1.31469816e-02  1.51106445e-02
  9.67373420e-03  5.70133813e-02 -1.02005657e-02  2.75593251e-04
  1.30269229e-02  2.90565174e-02 -1.04009686e-02  1.56834647e-02
  1.36420568e-02  5.52887321e-02  1.02007892e-02  4.47700806e-02
  1.57450140e-02  1.79775320e-02 -2.48192321e-03  2.43115555e-02
 -1.91597559e-03  3.21599655e-02  5.75749762e-03  2.28326190e-02]
[-2.34549828e-02  1.35772107e-02 -2.83268257e-03  1.65358521e-02
 -1.67307432e-03 -9.60322283e-03  5.26861101e-03 -8.39495659e-03
 -3.31028481e-03  1.38823148e-02  1.02282576e-02 -1.64470356e-02
  2.57251854e-03  3.56780067e-02 -2.55661737e-02 -1.89057533e-02
 -3.50413918e-02 -1.93482649e-03  7.60123180e-03 -2.38003526e-02
 -2.10769456e-02  9.26411990e-03  2.96217874e-02  3.87497842e-02
 -3.33515666e-02  2.95759495e-02  1.82348005e-02  2.74082762e-03]
[ 3.85328894e-03  4.13064845e-03  2.94505269e-03 -6.20766543e-03
 -1.86901893e-02  3.54643911e-02 -1.75493304e-04  2.07050834e-02
 -2.79295463e-02  2.69468874e-02 -4.52396460e-03  2.37767417e-02
 -2.56198589e-02 -3.70600969e-02 -1.35905109e-04  3.14200372e-02
  1.10049276e-02  4.85419109e-02  3.39936698e-03  3.57819907e-02
 -3.27091217e-02  1.50567573e-02 -1.21792872e-02  3.80223207e-02
 -1.59161761e-02  2.37812176e-02 -3.03430762e-02  1.30636655e-02]
[-3.86797567e-03 -1.60422567e-02 -2.43957620e-02  1.10514108e-02
  1.65291373e-02 -7.27064721e-03 -5.34395687e-03 -3.69516797e-02
  1.72805116e-02 -1.16555551e-02 -1.90357994e-02  2.76531447e-02
  1.43724903e-02  7.13825505e-03  1.45214470e-02 -1.52591364e-02
  3.17370296e-02  4.62593772e-02 -6.28171815e-03  3.10062990e-03
  2.66452301e-02 -2.50352826e-02 -2.91586332e-02 -1.56176751e-02
 -1.12230983e-03  3.85432271e-03  5.43608610e-03  1.56829990e-02]
[-7.85047095e-03  8.85084085e-03  1.58798862e-02  1.10134063e-02
  7.36510707e-03  2.49205176e-02 -3.00005078e-04 -3.19017936e-03
  2.21886691e-02  4.84941006e-02  7.54292775e-03  6.55706320e-03
  1.46177206e-02  2.62406822e-02  2.23700684e-02  2.73816623e-02
  1.27883069e-02  2.63095275e-02 -3.42566520e-04  1.70338154e-02
 -2.46900916e-02  2.62430534e-02  3.90147651e-03 -1.34900527e-03
  1.90566424e-02  3.11319046e-02  1.43954344e-02  1.66467037e-02]
[-1.31925484e-02 -3.12820487e-02  1.71330897e-03 -9.86515079e-04
  6.13843370e-03 -1.79215241e-02  2.08883956e-02 -1.19859846e-02
  1.67707447e-02 -1.55014321e-02  8.65990203e-03 -1.89970620e-02
 -8.00947752e-03  1.42508640e-03 -1.08684134e-02  2.07596757e-02
 -2.14992110e-02  2.32977443e-03  7.96529837e-03  5.02833277e-02
 -1.01702446e-02  8.26890301e-03  7.23348418e-03 -6.21057935e-02
  7.33853830e-03  1.94614511e-02  1.78443678e-02  5.93039114e-03]
[-7.81717710e-03 -1.34946534e-03 -2.80731544e-03  9.29834135e-03
  5.84712857e-03  1.65134785e-03 -2.53215227e-02  2.82445326e-02
 -4.03985307e-02 -7.90739059e-03 -3.91517021e-03  3.78039386e-03
 -3.06593422e-02  1.38226775e-02 -3.40837203e-02  6.58990769e-03
 -5.04411645e-02  4.43868153e-02  8.50271992e-03  1.42728714e-02
  5.84286451e-03  2.84216572e-02 -3.17852981e-02  3.78704779e-02
 -3.05971084e-03  4.32671141e-03 -8.05111974e-03  1.93155464e-02]
```

```
 [ 5.96914208e-03  1.76449884e-02 -1.14898253e-02  7.98658188e-03
   2.31964663e-02  3.53593729e-03  7.36326212e-03  2.30566878e-02
   2.02969890e-02  2.72186082e-02 -6.02912763e-03 -1.02548106e-02
   4.09747707e-03 -3.91995534e-03 -4.82181460e-02  2.32248511e-02
   1.63026005e-02  6.47097034e-03 -6.37272373e-03 -8.62308126e-03
   4.87347133e-02  1.37459431e-02 -4.43631709e-02  2.11394485e-02
   3.82523462e-02 -6.86142547e-03 -3.19041386e-02  7.84905627e-03]
 [-2.88222358e-03  3.83658567e-03  1.70309329e-03  5.71256876e-03
  -6.89562643e-03  2.85353642e-02  6.69324026e-03  1.55205128e-03
  -6.56467583e-03  4.80118990e-02 -1.81145011e-03  1.87583119e-02
  -5.18658059e-03  2.12544128e-02  1.57160405e-02  4.08625603e-02
  -5.24634868e-03  5.51196560e-03 -9.80710052e-03  1.18083479e-02
   1.98366195e-02  2.37144753e-02  2.56624855e-02  5.49658760e-03
  -9.15983412e-03  3.19082625e-02  1.97050720e-03  3.95671837e-03]
 [-1.85337123e-02 -2.79418286e-03 -3.11783492e-03  2.67298892e-04
  -1.26568889e-02  9.56410542e-03  4.57463926e-03  1.39884222e-02
  -8.97848513e-03  2.16160808e-03  3.44935395e-02  1.76501218e-02
  -1.54949697e-02  1.86550785e-02  1.02785612e-02  2.67586466e-02
  -1.61187742e-02  2.44778171e-02 -1.09618418e-02  1.26611823e-02
   5.14799682e-03  1.82507448e-02  6.07137103e-03  5.45747718e-03
  -1.25035401e-02  1.05413534e-02  2.00942494e-02 -6.90895272e-03]
 [ 4.92060883e-03  1.36085786e-02  2.08623125e-03  1.80973567e-03
  -3.47900833e-03  1.02326134e-02 -2.74596624e-02 -2.03272719e-02
  -2.94618141e-02  1.63176656e-02 -1.23173650e-02  6.00580871e-03
  -8.44959728e-03 -2.02382170e-02 -7.43273739e-03  2.17457376e-02
  -6.34623645e-03  2.96633365e-03 -2.77471934e-02  4.29523252e-02
  -1.03504770e-03  1.63896009e-02 -3.54571827e-02  2.95358188e-02
  -7.52937607e-03  2.24899147e-02  8.83991888e-05  5.40603977e-03]
 [-6.30620122e-03  5.19750582e-04 -8.73901416e-03 -3.16255214e-03
   1.43422689e-02 -2.36630533e-03 -1.90377496e-02  1.18650170e-02
   1.22430092e-02 -3.83310206e-02  7.17884162e-03  2.05853358e-02
   1.21160615e-02 -4.43583308e-03 -1.66177116e-02  1.63634792e-02
  -1.40515238e-03 -3.20174359e-02 -7.93281198e-03  2.08674520e-02
   1.25873769e-02 -3.35111166e-03 -2.91145407e-04 -9.03224293e-03
  -5.45793725e-03 -1.52029870e-02 -8.64219561e-04  2.02511437e-04]], shape=(28, 28),
dtype=float32)
```

what is this image?

This is a digital image generated by the untrained generator model from normalized random noise of size (1, 100) created by the tf.random.normal method. The noise is fed into the make_generator_model function which in turns performed up sampling (3x times) and produced a generated grayscale image of size (1, 28, 28, 1).

In [9]:
```
#c- Create discriminator model
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                     input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
```

```
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

In [10]:
```
# Applying discriminative model on the untrained image which was the output of gene
#image_path = "Pros_RRM.jpg"
#image = tf.keras.preprocessing.image.load_img(image_path, color_mode = "grayscale"
#image = tf.keras.preprocessing.image.img_to_array(image)
#image = (image - 127.5) / 127.5  # Normalize the image to [-1, 1]
#image = np.expand_dims(image, axis=0)  # Add batch dimension

discriminator = make_discriminator_model() # Call discriminator function
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[-0.0008708]], shape=(1, 1), dtype=float32)
c:\Users\Loung\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\l
ayers\convolutional\base_conv.py:113: UserWarning: Do not pass an `input_shape`/`inp
ut_dim` argument to a layer. When using Sequential models, prefer using an `Input(sh
ape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [11]:
```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
cross_entropy
```

Out[11]:
```
<LossFunctionWrapper(<function binary_crossentropy at 0x000002262FC3C360>, kwargs=
{'from_logits': True, 'label_smoothing': 0.0, 'axis': -1})>
```

In [12]:
```
#d- Compute the loss
# Discriminator loss This method quantifies how well the discriminator is able to d
# It compares the discriminator's predictions on real images to an array of 1s, and
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output) #Same comman
    total_loss = real_loss + fake_loss
    return total_loss
```

In [13]:
```
#Generator loss The generator's loss quantifies how well it was able to trick the d
#Intuitively, if the generator is performing well, the discriminator will classify
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

In [14]:
```
#The discriminator and the generator optimizers are different since we will train t
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4) #same adam optimizer here
```

In [15]:
```
# e- Model design
# Let's save the model for future references
checkpoint_dir = '/content/training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
```

```
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

In [16]:
```
# Model parametrs
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# We will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

In [17]:
```
#The training loop begins with generator receiving a random seed as input. That see
#The discriminator is then used to classify real images (drawn from the training se
#The loss is calculated for each of these models, and the gradients are used to upd
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    '''
    Arguments:
    images -- real images from the dataset
    '''
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variab
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.traina

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.train
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discrim
```

In [18]:
```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as we go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)
```

```
    # Save the model every 15 epochs
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Generate after the final epoch
  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

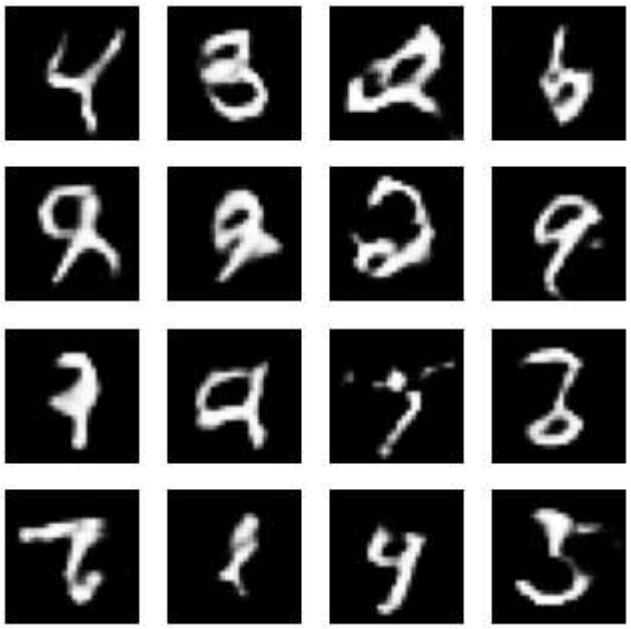
In [19]:
```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4,4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

In [23]:
```
#Call the function in proper way
train(train_dataset, epochs=50)  #Put appropriate values here and call the function
```



In [22]:
```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[22]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x226680dcfd0>
```

In [6]:
```
# You can use these commands to save the evpolution as a GIF file
anim_file = 'GAN_digit.gif'

import imageio.v2 as imageio

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('image*.png')
  filenames = sorted(filenames)
  last = -1
  for i,filename in enumerate(filenames):
    frame = 2*(i**0.5)
    if round(frame) > round(last):
      last = frame
    else:
      continue
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)

#import IPython.display
#if IPython.version_info > (6,2,0,''):
display.Image(filename=anim_file)
```

Out[6]:
```
<IPython.core.display.Image object>
```

what is your understanding from this GIF file?

The codes block above creates an animated GIF from a series of .PNG images resulted from the Generator training process. It shows the evolution of generated digits from the GAN training process. The generator's abiliyt to generate realistic digits improved as training epochs increases.

- Frame 1-10: Random noise → Blobby shapes
- Frame 11-30: Recognizable but distorted digits
- Frame 31-50: Clear digits with some artifacts
- Frame 51+: Clean, realistic MNIST digits

Part 2- Pokemon characters and GAN

In [21]:
```
#Install d2l package
# Uncomment the below line to install d2l package
# !pip install d2l==1.0.0-alpha1.post0
```

In [1]:
```
import tensorflow as tf
from d2l import tensorflow as d2l
```

In [2]:
```
#a- Read the dataset and store it
d2l.DATA_HUB['pokemon'] = (d2l.DATA_URL + 'pokemon.zip',
                           'c065c0e2593b8b161a2d7873e42418bf6a21106c')
```

```
# We store the directory in this variable
data_dir = d2l.download_extract('pokemon')
batch_size = 256
# Use preprocessing from Keras to read images from directory using given batch size
pokemon = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir, batch_size=batch_size, image_size=(64, 64))
```

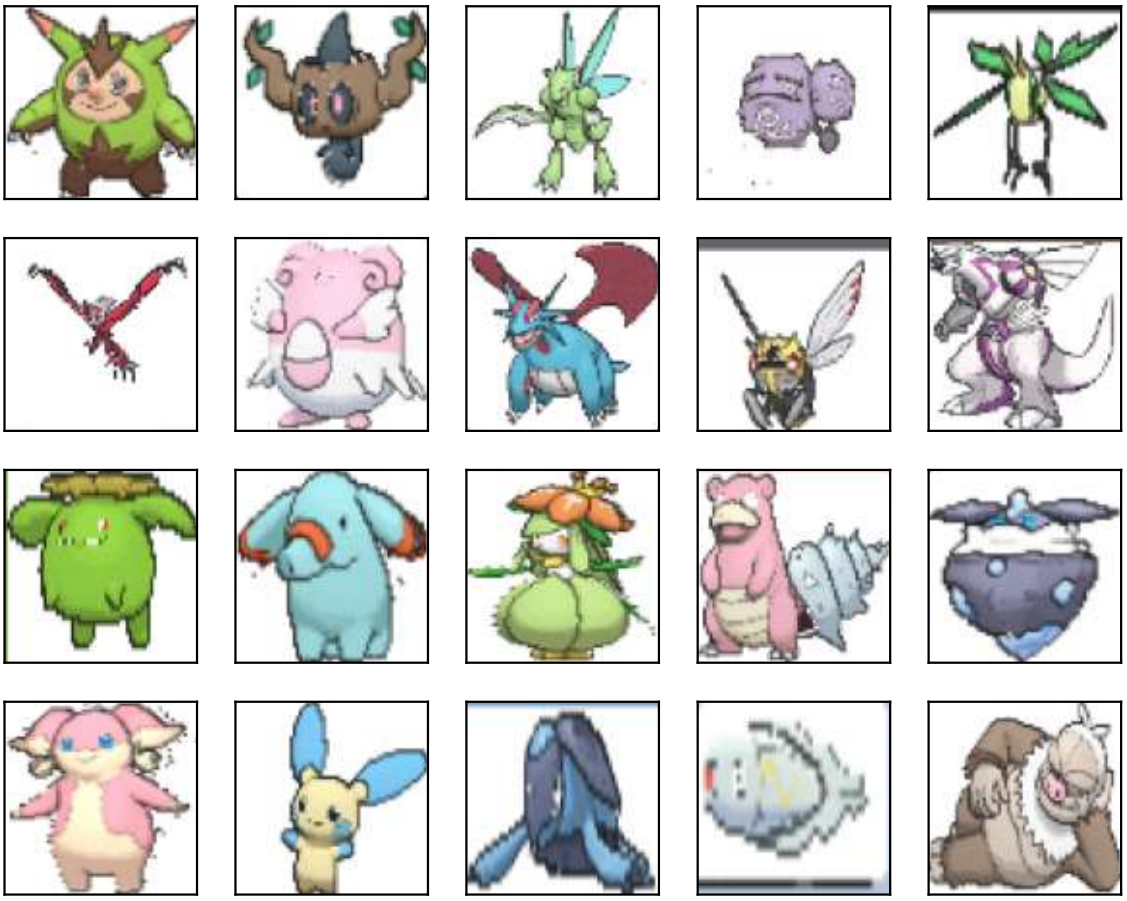
```
Found 40597 files belonging to 721 classes.
```

In [ ]:
```
#Normalize the data between [-1,1]
# We normalize the data with  0.5  mean and  0.5  standard deviation to match the v
# Write a function to normalize the images by dividing them to 255 and use gaussian
def transform_func(X):
    '''
    Write the function here in which it normalied the X in the way explained above.

    ARGUMENTS:
    X -- input image
    '''
    X = X / 255.0
    X = (X - 0.5) / 0.5
    return X

data_iter = pokemon.map(lambda x, y: (transform_func(x), y),
                        num_parallel_calls=tf.data.experimental.AUTOTUNE)
data_iter = data_iter.cache().shuffle(buffer_size=1000).prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE)
```

In [ ]:
```
#b- Visualize first 20 images
d2l.set_figsize(figsize=(4, 4))
for X, y in data_iter.take(1):
    imgs = X[:20, :, :, :] / 2 + 0.5
    d2l.show_images(imgs, num_rows=4, num_cols=5)
```



The Generator

The generator needs to map the noise variable to a RGB image with width and height to be 64×64 .

In [ ]:
```
#c- Creating the generator
class G_block(tf.keras.layers.Layer):
    '''
        Creates a generator block with Conv2DTranspose, BatchNormalization, and ReL

        Arguments:
        out_channels -- number of output channels
        kernel_size -- size of the convolution kernel (default 4)
        strides -- stride of the convolution (default 2)
        padding -- padding type (default "same")
        X -- input image
    '''
    def __init__(self, out_channels, kernel_size=4, strides=2, padding="same",
                 **kwargs):
        super().__init__(**kwargs)
        self.conv2d_trans = tf.keras.layers.Conv2DTranspose(
            out_channels, kernel_size, strides, padding, use_bias=False)
        self.batch_norm = tf.keras.layers.BatchNormalization()
        self.activation = tf.keras.layers.ReLU()
```

```
    def call(self, X):
        return self.activation(self.batch_norm(self.conv2d_trans(X)))
```

Explain your understanding of the G_block here.

The G_block class creates a generator block with Conv2DTranspose, BatchNormalization, and ReLU activation. It takes four input arguments: "out_channels", "kernel_size", "strides", and "padding". The out_channels is the number of output maps/filters. The kernel_size is the size of the convolution filter, which defaults to a 4x4 filter. The strides parameter is the upsampling factor - when strides equal 2, it doubles the spatial dimensions. Padding = "same" preserves spatial dimensions after convolution. The output of the Conv2DTranspose layer is then normalized and fed into the activation layer that uses ReLU for non-linearity. The call method takes in X (input image), upsamples it, normalizes it, and then applies ReLU activation for non-linear transformation.

The G_block is used for upsampling low-resolution images to high-resolution images.

In [ ]:
```
x = tf.zeros((2, 16, 16, 3))  # Input based on the instruction
#Call the G_block package and use 20 as the number of layers (input of the class)
g_blk = G_block(20)
# Show the dimension of the output
g_blk(x).shape
```

Out[ ]:
```
TensorShape([2, 32, 32, 20])
```

The generator consists of four basic blocks that increase input's both width and height from 1 to 32. At the same time, it first projects the latent variable into 64×8 channels, and then halve the channels each time. At last, a transposed convolution layer is used to generate the output. It further doubles the width and height to match the desired 64×64 shape, and reduces the channel size to 3. The tanh activation function is applied to project output values into the $(-1, 1)$ range.

In [ ]:
```
n_G = 64
net_G = tf.keras.Sequential([
    # Output: (4, 4, 64 * 8)
    G_block(out_channels=n_G*8, strides=1, padding="valid"),
    G_block(out_channels=n_G*4), # Output: (8, 8, 64 * 4)
    G_block(out_channels=n_G*2), # Output: (16, 16, 64 * 2)
    G_block(out_channels=n_G), # Output: (32, 32, 64)
    # Output: (64, 64, 3)
    tf.keras.layers.Conv2DTranspose(
        3, kernel_size=4, strides=2, padding="same", use_bias=False,
        activation="tanh")
])
```

In [ ]:
```
# Generate a 100 dimensional latent variable to verify the generator's output shape
x = tf.zeros((1, 1, 1, 100))
net_G(x).shape
```

Out[ ]: TensorShape([1, 64, 64, 3])

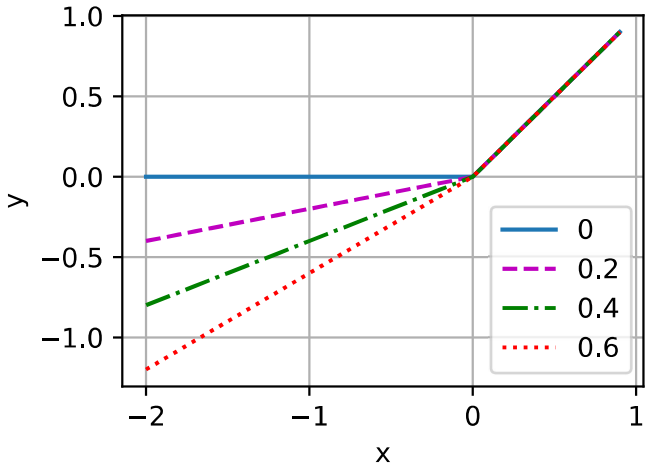
Discriminator

The discriminator is a normal convolutional network network except that it uses a leaky ReLU as its activation function. Given $\alpha \in [0, 1]$, its definition is

$$\text{leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases}.$$

As it can be seen, it is normal ReLU if $\alpha = 0$, and an identity function if $\alpha = 1$. For $\alpha \in (0, 1)$, leaky ReLU is a nonlinear function that give a non-zero output for a negative input. It aims to fix the "dying ReLU" problem that a neuron might always output a negative value and therefore cannot make any progress since the gradient of ReLU is 0.

In [ ]:
```
alphas = [0, .2, .4, .6, .8, 1]
x = tf.range(-2, 1, 0.1)
Y = [tf.keras.layers.LeakyReLU(alpha)(x).numpy() for alpha in alphas]
d2l.plot(x.numpy(), Y, 'x', 'y', alphas)
```



The basic block of the discriminator is a convolution layer followed by a batch normalization layer and a leaky ReLU activation. The hyperparameters of the convolution layer are similar to the transpose convolution layer in the generator block

In [ ]:
```
# d- Block of Discriminator
class D_block(tf.keras.layers.Layer):
    def __init__(self, out_channels, kernel_size=4, strides=2, padding="same",
                 alpha=0.2, **kwargs):
        super().__init__(**kwargs)
        self.conv2d = tf.keras.layers.Conv2D(out_channels, kernel_size,
                                             strides, padding, use_bias=False)
        self.batch_norm = tf.keras.layers.BatchNormalization()
        self.activation = tf.keras.layers.LeakyReLU(alpha)
```

```
    def call(self, X):
        return self.activation(self.batch_norm(self.conv2d(X)))
```

Explain your understanding of D_block here.

The code defines a discriminator block for a GAN. It consists of three layers:

- Conv2D: A standard convolutional layer with a 4x4 filter that extracts features and reduces spatial dimensions.
- BatchNormalization: Stabilizes training by normalizing activations.
- LeakyReLU: An activation function that allows small negative values through, as defined by the alpha value.

The class takes the following arguments:

- out_channels: Number of output feature maps/filters
- kernel_size=4: Size of the convolution filter (4x4)
- strides=2: Downsampling factor (halves the spatial dimensions)
- padding="same": Preserves spatial dimensions after convolution
- alpha=0.2: Slope for LeakyReLU's negative side

The call method takes in the input, performs convolution filtering on it to extract features and downsample it, normalizes the output, and then applies the LeakyReLU activation function with a small negative slope.

In [ ]:
```
x = tf.zeros((2, 16, 16, 3))
#Call the G_block package and use 20 as the number of layers (input of the class)
d_blk = D_block(20)
# Show the dimension of the output
d_blk(x).shape
```

Out[ ]:
```
TensorShape([2, 8, 8, 20])
```

The discriminator is a mirror of the generator.

In [ ]:
```
n_D = 64
net_D = tf.keras.Sequential([
    D_block(n_D), # Output: (32, 32, 64)
    D_block(out_channels=n_D*2), # Output: (16, 16, 64 * 2)
    D_block(out_channels=n_D*4), # Output: (8, 8, 64 * 4)
    D_block(out_channels=n_D*8), # Outupt: (4, 4, 64 * 8)
    # Output: (1, 1, 1)
    tf.keras.layers.Conv2D(1, kernel_size=4, use_bias=False)
])
```

In [ ]:
```
#It uses a convolution layer with output channel  1  as the last layer to obtain a
x = tf.zeros((1, 64, 64, 3))
net_D(x).shape
```

Out[ ]:
```
TensorShape([1, 1, 1, 1])
```

Training

In [ ]:
```
def train(net_D, net_G, data_iter, num_epochs, lr, latent_dim,
          device=d2l.try_gpu()):
    '''
    Trains the GAN model.

    Arguments:
    net_D -- Discriminator network
    net_G -- Generator network
    data_iter -- Data iterator
    num_epochs -- Number of epochs
    lr -- Learning rate
    latent_dim -- Dimension of latent variable
    device -- Device to run the training on (default is GPU if available)
    '''

    loss = tf.keras.losses.BinaryCrossentropy(
        from_logits=True, reduction=tf.keras.losses.Reduction.SUM)

    for w in net_D.trainable_variables:
        w.assign(tf.random.normal(mean=0, stddev=0.02, shape=w.shape))
    for w in net_G.trainable_variables:
        w.assign(tf.random.normal(mean=0, stddev=0.02, shape=w.shape))

    optimizer_hp = {"learning_rate": lr, "beta_1": 0.5, "beta_2": 0.999} # Changed
    optimizer_D = tf.keras.optimizers.Adam(**optimizer_hp)
    optimizer_G = tf.keras.optimizers.Adam(**optimizer_hp)

    animator = d2l.Animator(xlabel='epoch', ylabel='loss',
                            xlim=[1, num_epochs], nrows=2, figsize=(5, 5),
                            legend=['discriminator', 'generator'])
    animator.fig.subplots_adjust(hspace=0.3)

    for epoch in range(1, num_epochs + 1):
        # Train one epoch
        timer = d2l.Timer()
        metric = d2l.Accumulator(3) # loss_D, loss_G, num_examples
        for X, _ in data_iter:
            batch_size = X.shape[0]
            Z = tf.random.normal(mean=0, stddev=1,
                                 shape=(batch_size, 1, 1, latent_dim))
            metric.add(d2l.update_D(X, Z, net_D, net_G, loss, optimizer_D),
                       d2l.update_G(Z, net_D, net_G, loss, optimizer_G),
                       batch_size)

        # Show generated examples
        Z = tf.random.normal(mean=0, stddev=1, shape=(21, 1, 1, latent_dim))
        # Normalize the synthetic data to N(0, 1)
        fake_x = net_G(Z) / 2 + 0.5
        imgs = tf.concat([tf.concat([fake_x[i * 7 + j] for j in range(7)],
                                    axis=1)
                          for i in range(len(fake_x) // 7)], axis=0)
        animator.axes[1].cla()
```

```
        animator.axes[1].imshow(imgs)
        # Show the losses
        loss_D, loss_G = metric[0] / metric[2], metric[1] / metric[2]
        animator.add(epoch, (loss_D, loss_G))
    print(f'loss_D {loss_D:.3f}, loss_G {loss_G:.3f}, '
          f'{metric[2] / timer.stop():.1f} examples/sec on {str(device._device_name
```

Explain your understanding from this model here?

Note: When I initially ran the train() function, it failed. To fix this, I updated the optimizer hyperparameter from "lr": lr to "learning_rate": lr.

The codes implement training loop for a GAN. The training pipeline consists of:

1. Function definition and input arguments:

- net_D: Discriminator network, it distinguishes real and fake image.
- net_G: Generator network, it creates fake images from random noise.
- data_iter: It provides real training images.
- Num_epochs: Number training epochs.
- Lr: learning rate for optimizers
- latent_dim: Dimension of the random noise vector (generator input)

2. Loss function and weight initialization:

- BinaryCrossentroy: It uses Binary Cross Entropy loss for binary classification (real vs fake).
- from_logits=True: The networks output raw logits, no softmax/sigmoid.
- Initializes all weight with normal distribution (mean = 0, stddev = 0.02).

3. Optimizer Set Up: Set up the optimizer parameters for the discriminator and generator networks with an Adam optimizer.
4. Training Loop:

- Outer loop: Iterates through epochs
- Inner loop: Processes each batch of real images X
- Z: Random noise input for generator (batch_size × latent_dim)
- update_D(): Updates discriminator to better distinguish real vs fake
- update_G(): Updates generator to create more convincing fakes
- metric: Tracks discriminator loss, generator loss, and example count

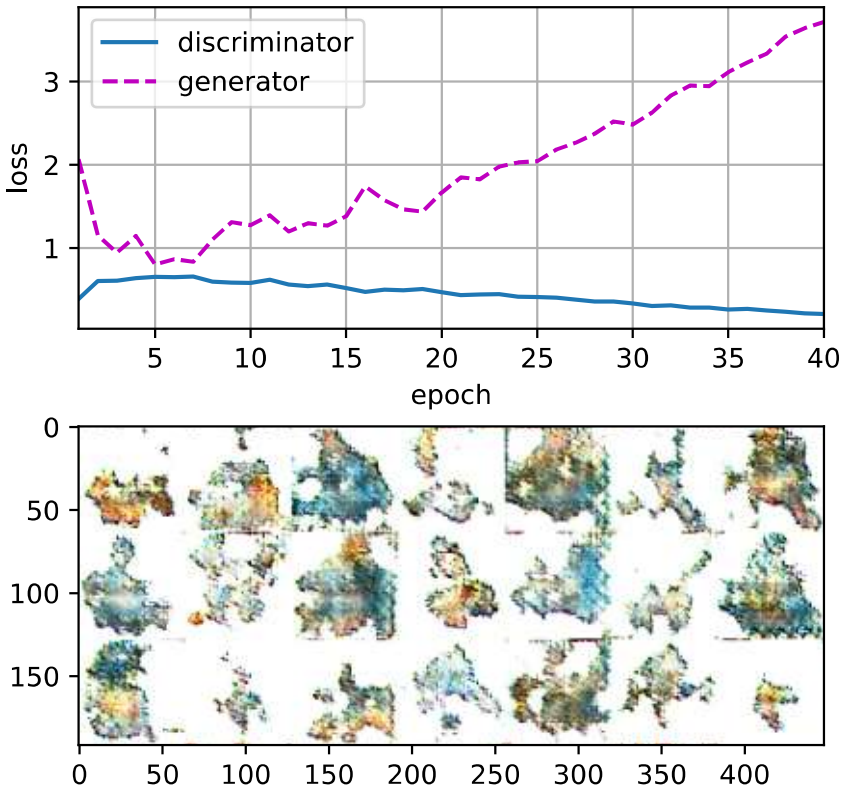
5. Visualization and Monitoring:

- Generate and display sample images
- Plot losses for the discriminator and generator

In [ ]:
```
#We train the model with a small number of epochs just for demonstration. For bette
latent_dim, lr, num_epochs = 100, 0.0005, 40
```

```
train(net_D, net_G, data_iter=data_iter, num_epochs=num_epochs, lr=lr, latent_dim=l
```

```
loss_D 0.208, loss_G 3.717, 700.0 examples/sec on /GPU:0
```



what is your understanding from this learning curve?

Generator Loss (Magenta dashed line):

- Starts around 2.0 and briefly decreases to ~1.5 in the first 5 epochs
- After epoch 7, it increases dramatically to extremely high values (~150)
- This extremely high loss indicates the generator is producing poor quality, easily detectable fake images.

Discriminator Loss (Blue solid line):

- Rapidly decreases to near-zero within the first few epochs
- Remains near zero throughout training
- This indicates the discriminator achieves near-perfect accuracy at distinguishing real vs fake images

In summary, the generator is not getting better at producing realistic images as it is indicated with high generator loss. And the discriminator is excelling at distinguishing real vs fake images. To fix this, we can train with the different learning rates, or add smoothing + noise, or train the discriminator less frequently.