Monet GAN project

The goal of this project is to make 256×256 images that look like Monet's paintings. I used the Kaggle dataset and the monet_jpg folder which contained 300 256×256 rgb images. I built a dcgan, trained a few epochs and generated 7,000 images

In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries inst
# It is defined by the kaggle/python Docker image: https://github.com/kaggle
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will l

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sav
```

Data check

I first checked that the dataset was added and the folders existed. The competition data has four folders: monet_jpg, photo_jpg, monet_tfrec and photo_tfrec. For this notebook I only used monet_jpg. I checked that monet_jpg had 300 images and each one is 256×256 rgb

In [ ]:
```
# detect data, set paths and confirm
import os

base = None
for d in os.listdir("/kaggle/input"):
    b = f"/kaggle/input/{d}"
    if os.path.isdir(os.path.join(b, "monet_jpg")) and os.path.isdir(os.path
        base = b
        break

assert base is not None, "Data isn't there."

kaggledat = base
monetjpg = os.path.join(kaggledat, "monet_jpg")

print("base:", kaggledat)
print("monet_jpg is there:", os.path.isdir(os.path.join(kaggledat,"monet_jpg
print("photo_jpg is there:", os.path.isdir(os.path.join(kaggledat,"photo_jpg
```

```
print("monet_tfrec is there:", os.path.isdir(os.path.join(kaggledat,"monet_t
print("photo_tfrec is there:", os.path.isdir(os.path.join(kaggledat,"photo_t
```

Data Prep

I loaded the jpegs with tensor flow and scaled pixels to -1, 1. I shuffled the files, did a random left right flip, made batches and then prefetched for speed

In [ ]:
```
# define constants and build dataset from monet jpegs
imgsize = 256
batch = 8
vectlength = 64
numepochs = 5

import os, glob, tensorflow as tf

# read jpeg and and set pixels to -1, 1
def readjpg(fname):
    img = tf.io.read_file(fname)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return img * 2.0 - 1.0

# build dataset: list the files, lr flip, batch and prefetch
def builddataset(folder, batchsize=8):
    filelist = sorted(glob.glob(os.path.join(folder, "*.jpg")))
    dataobj = tf.data.Dataset.from_tensor_slices(filelist)
    dataobj = dataobj.shuffle(len(filelist), reshuffle_each_iteration=True)
    dataobj = dataobj.map(readjpg, num_parallel_calls=tf.data.AUTOTUNE)
    dataobj = dataobj.map(lambda img: tf.image.random_flip_left_right(img),
    dataobj = dataobj.batch(batchsize, drop_remainder=True).prefetch(tf.data
    return dataobj, len(filelist)
```

Model and training

I used a small dcgan. The generator takes a short random input of 64 numbers and made a 256×256 rgb image with conv-transpose layers and a tanh output. The discriminator is a small cnn that takes an image and outputs one number for real v fake. I trained with binary cross entropy from logits and Adam at 1e-4 for both. Each step I updated the discriminator on real and fake and then updated the generator to trick it. I ran 5 epochs for a baseline

In [ ]:
```
# get dataset and print img count
traindata, monetcounter = builddataset(monetjpg, batch)
print("Monet images:", monetcounter)

# create generator and discriminator, then set up and run gan training loop
import tensorflow as tf
from tensorflow.keras import layers

# generator: maps vectors to 256x256 rgb images
```

```
def makegenerator():
    g = tf.keras.Sequential(name="G")
    g.add(layers.Input(shape=(vectlength,)))
    g.add(layers.Dense(16*16*128, use_bias=False))
    g.add(layers.BatchNormalization()); g.add(layers.LeakyReLU())
    g.add(layers.Reshape((16,16,128)))
    g.add(layers.Conv2DTranspose(128, 4, 2, "same", use_bias=False)); g.add(
    g.add(layers.Conv2DTranspose(64, 4, 2, "same", use_bias=False)); g.add(l
    g.add(layers.Conv2DTranspose(32, 4, 2, "same", use_bias=False)); g.add(l
    g.add(layers.Conv2DTranspose(16, 4, 2, "same", use_bias=False)); g.add(l
    g.add(layers.Conv2D(3, 3, padding="same", activation="tanh"))
    return g

# discriminator. cnn that outputs one logit for real v fake
def makediscrim():
    d = tf.keras.Sequential(name="D")
    d.add(layers.Input(shape=(imgsize, imgsize, 3)))
    d.add(layers.Conv2D(32, 4, 2, "same")); d.add(layers.LeakyReLU()); d.adc
    d.add(layers.Conv2D(64, 4, 2, "same")); d.add(layers.LeakyReLU()); d.adc
    d.add(layers.Conv2D(128, 4, 2, "same")); d.add(layers.LeakyReLU()); d.ac
    d.add(layers.Conv2D(256, 4, 2, "same")); d.add(layers.LeakyReLU()); d.ac
    d.add(layers.Flatten()); d.add(layers.Dense(1))
    return d

# init models
generator = makegenerator()
discriminator = makediscrim()

# gan set up. bin cross entropy logits and adam optimizers
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
optgen = tf.keras.optimizers.Adam(1e-4)
optdiscrim = tf.keras.optimizers.Adam(1e-4)

# one train step. train discriminator real and fake then train generator to
@tf.function
def trainone(real):
    bs = tf.shape(real)[0]
    z = tf.random.normal([bs, vectlength])
    with tf.GradientTape() as dt:
        fake = generator(z, training=True)
        rlog = discriminator(real, training=True)
        flog = discriminator(fake, training=True)
        dloss = loss(tf.ones_like(rlog), rlog) + loss(tf.zeros_like(flog), f
    dgr = dt.gradient(dloss, discriminator.trainable_variables)
    optdiscrim.apply_gradients(zip(dgr, discriminator.trainable_variables))

    z = tf.random.normal([bs, vectlength])
    with tf.GradientTape() as gt:
        fake = generator(z, training=True)
        flog = discriminator(fake, training=True)
        gloss = loss(tf.ones_like(flog), flog)
    ggr = gt.gradient(gloss, generator.trainable_variables)
    optgen.apply_gradients(zip(ggr, generator.trainable_variables))

    return dloss, gloss
```

```
# training loop. iterate over data for few epochs and print average loss
def runthepochs(data, epochs):
    for e in range(epochs):
        dsum = 0.0; gsum = 0.0; steps = 0
        for real in data:
            dl, gl = trainone(real)
            dsum += float(dl); gsum += float(gl); steps += 1
        print(f"Epoch {e+1}/{epochs} d_loss={dsum/max(steps,1):.4f} g_loss={

runthepochs(traindata, numepochs)
```

Image generation

After training I sampled 7,000 images from the generator and saved them as 256×256 rgb jpegs

In [ ]:
```
# generate images and sub
from pathlib import Path
import zipfile
import tensorflow as tf

imgwrite = Path("/kaggle/working/images")
imgwrite.mkdir(parents=True, exist_ok=True)

imgcounter = 7000
savestep = 100

# convert -1,1 floats to uint8
def tou8(arr):
    arr = tf.clip_by_value((arr + 1.0) * 127.5, 0, 255)
    return tf.cast(arr, tf.uint8)

saved = 0
idx = 0
while saved < imgcounter:
    take = min(savestep, imgcounter - saved)
    z = tf.random.normal([take, vectlength])
    imgs = generator(z, training=False)
    imgs = tou8(imgs).numpy()
    for i in range(take):
        tf.keras.utils.save_img(str(imgwrite / f"{idx:05d}.jpg"), imgs[i])
        idx += 1
    saved += take
    print("saved:", saved)

zipath = Path("/kaggle/working/images.zip")
with zipfile.ZipFile(zipath, "w", compression=zipfile.ZIP_STORED) as zf:
    for p in sorted(imgwrite.glob("*.jpg")):
        zf.write(p, arcname=p.name)

print("wrote:", zipath, "bytes:", zipath.stat().st_size)
```

Results

I trained for 5 epochs on the 300 Monet jpegs. The logs showed the discriminator loss going from 0.8751 to 0.1019 and the generator loss going from 1.5782 to 9.8325. When I looked at a grid of samples they mostly looked like purple/green blobs or textures, not clear scenes. My Kaggle score was 397.92205 MiFID which improved over my first submission 473.80291

Conclusion

The images captured Monet like color and texture but not the structure or form of the Monet paintings which makes sense to me given the small model and short amount of time for training. If I had more time I'd try training longer, slightly changing the optimizer settings and making the networks a bit bigger to see if shapes start to appear