Prog6_WGAN_GP_Flower_Colab

November 26, 2024

1 CPSC320: Program 6 - Wasserstein Generative Adversarial Network (WGAN_GP) on Flower Dataset (Run on Google Colab)

In this programming assignment, you will build a GAN_GP for generating flower images.

Important: The notebook you will submit must be the one you have RUN all the cells (DO NOT CLEAR OUTPUTS OF ALL CELLS).

Hints: You may refere to the scripts of *10_4_wgan_gp_mnist.ipynb* and *10_5_WGAN_GP_CeleA_Faces.ipynb* to get most of he code from there. You need to make some adjustments wherever necessary.

[1]:
```
# Import TensorFlow and Keras
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras import layers, models, metrics, losses
from tensorflow.keras.layers import Input, Conv2D, Flatten, Dense, Reshape,␣
↪Conv2DTranspose, Lambda, Cropping2D
from tensorflow.keras.models import Model
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras import backend as K
import numpy as np
import matplotlib.pyplot as plt
from IPython import display
```

[2]:
```
# check whether GPU is avaiable to use
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
!nvidia-smi
```

```
Num GPUs Available:  1
Fri Nov 22 17:44:51 2024
+---------------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+-------------
```

```
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==========
========|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
0 |
| N/A   53C    P8              10W /  70W |      3MiB / 15360MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+----------
--------+

+---------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                            GPU
Memory |
|        ID   ID
Usage      |
|=========================================================================
========|
|  No running processes found
|
+---------------------------------------------------------------------------
--------+
```

1.1 1. Dataset Preparation

Note: - The setup for this section should be very similar to the section 1 of the previous program. - If you have your own machine with gpu installed, you may modify your scripts in Section 1 as you did in program 5.

1.1.1 1.1 Upload Dataset to Google Colab

[3]:
```
# mount to your google drive
from google.colab import drive
drive.mount('/content/drive')
```

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

[4]:
```
import zipfile
import os

#WARNING: YOU MUST CHANGE THE ZIP PATH SO IT READS THE ZIPPED DATASET FROM YOUR␣
 ↪GOOGLE DRIVE#

zip_path = '/content/drive/MyDrive/data/flowers_train_validation.zip'  # Change␣
 ↪this to your zip path
extract_path = '/content/'

os.makedirs(extract_path, exist_ok=True)

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Dataset unzipped successfully!")
```

```
Dataset unzipped successfully!
```

[5]:
```
IMAGE_SIZE = 64
CHANNELS = 3
NUM_FEATURES = 128
Z_DIM = 50
BATCH_SIZE = 128
input_shape = (64,64, 3)
BETA = 1
latent_dim = 50
codings_size = 50
img_shape = (64,64, 3)
```

1.1.2 1.2: Data Preprocessing

1.2.1 Using image_dataset_from_directory for constructing training dataset Task 1: Read image_dataset_from_directory:

Do the followings: - Provide your train dateset directory unzipped from Section 1.1. - Set your image_size to be (64, 64). - Set your batch_size to be 128. - Modify other arguments if necessary.

[5]:
```
# Load the train data
# We only use training dataset to build our VAE model
train_data = tf.keras.utils.image_dataset_from_directory(
    None,  # this is the train dataset folder on your google colab
    color_mode= "rgb",
    image_size= None,
    batch_size=None,
    labels=None,
    shuffle=True,
    seed=42,
```

```
    interpolation="bilinear",
)

print(type(train_data))
```

1.2.2 Preprocessing images Task 2: Preprocessing image

Preprocess the imput image so that it is normalized into the value between (-1, 1) *i.e.*, the original pixel value should be subtracted from 127.5 and then divided by 127.5, and then return the normalized image back.

Why? Because the last layer of generator in our GAN_GP model has the activation of *tanh*. We need to be consistent for both faked images and real images when they are fed into our critic model.

[7]:
```
# Preprocess the data below
def preprocess(img):
    pass
```

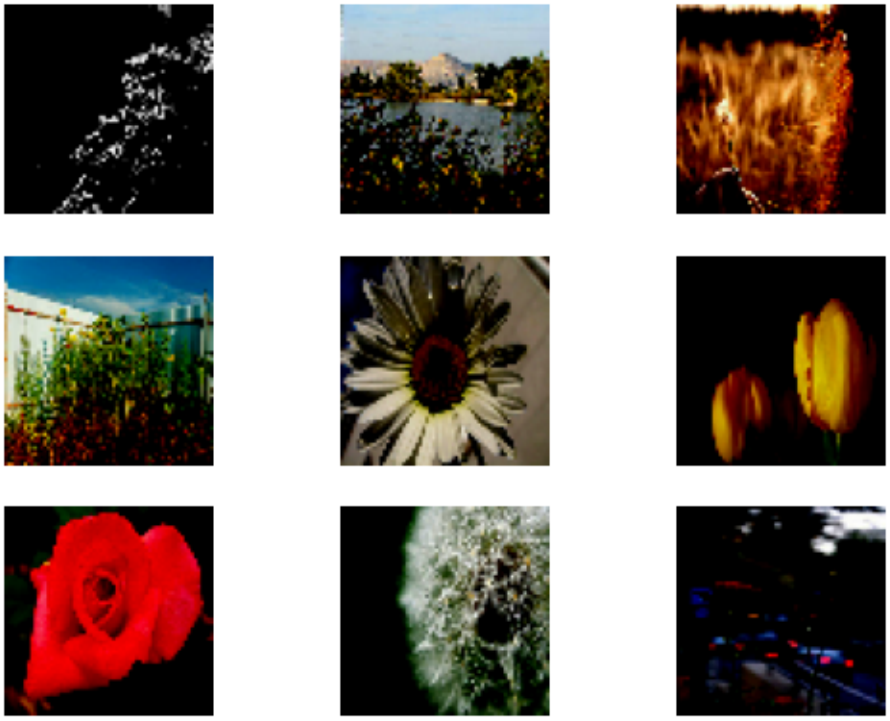
[9]:
```
# the training dataset has been processed for model training
train_data = train_data.map(lambda x: preprocess(x))
```

1.1.3 1.3 Checking train data

Note: If all the above steps are set correctly, you should be able to see the images read frm training data.

[10]:
```
import matplotlib.pyplot as plt

# Take one batch from the dataset
for batch in train_data.take(1):
    # Iterate through images in the batch
    for i in range(9):  # Show 9 images for preview
        plt.subplot(3, 3, i + 1)
        plt.imshow(batch[i].numpy().clip(0, 1))  # Clip values to [0, 1] range
        plt.axis('off')
    plt.show()
```



1.2 2. Define the WGAN_GP Model

1.2.1 2.1 The Generator

Task 3: Generator:

Define your generator **on your own choices**, but you may follow the common strategies:

- **The Generator Architecture**. The Generator Architecture is very similar to decoder in VAE, that is, using transposed convolutional layers (Conv2DTranspose).
- **Activation Functions**. The final layer should use an activation function that matches the data characteristics, i.e., *tanh* for pixel values in the range [-1, 1]. Intermediate layers can use ReLU or LeakyReLU.
- **Final Output Layer**: In general, you should have the same number of channels and spatial dimensions as the original input image i.e., **(64, 64 ,3)**. So you need cropping or padding if necessary.
- Suggested input Latent Space **dimension = 50** (don't make it too big as our training set size is only about 3000 vs celeb dataset size = 200,000
- print out the generator model summary

Note: You may refer to the generator model in *10_5_WGAN_GP_CeleA_Faces.ipynb*

[ ]:

[12]:
```
# Define the Generator below

# Generator
generator = None
```

[11]:
```
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 50)	0
reshape (Reshape)	(None, 1, 1, 50)	0
conv2d_transpose (Conv2DTranspose)	(None, 4, 4, 512)	409,600
batch_normalization (BatchNormalization)	(None, 4, 4, 512)	2,048
leaky_re_lu (LeakyReLU)	(None, 4, 4, 512)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 8, 8, 256)	2,097,152
batch_normalization_1 (BatchNormalization)	(None, 8, 8, 256)	1,024
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 256)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 16, 16, 128)	524,288
batch_normalization_2	(None, 16, 16, 128)	512

```
(BatchNormalization)                                                    ␣
↪
leaky_re_lu_2 (LeakyReLU)               (None, 16, 16, 128)             ␣
↪  0
conv2d_transpose_3 (Conv2DTranspose)    (None, 32, 32, 64)              ␣
↪131,072
batch_normalization_3                   (None, 32, 32, 64)              ␣
↪256
(BatchNormalization)                                                    ␣
↪
leaky_re_lu_3 (LeakyReLU)               (None, 32, 32, 64)              ␣
↪  0
conv2d_transpose_4 (Conv2DTranspose)    (None, 64, 64, 3)               ␣
↪3,075


Total params: 3,169,027 (12.09 MB)

Trainable params: 3,167,107 (12.08 MB)

Non-trainable params: 1,920 (7.50 KB)
```

1.3 2.2 The Critic

Task 4: Critic:

Define your critic **on your own choices**, but you may follow the common strategies:

- **Fully Convolutional Design**: The critic uses convolutional layers to downsample the image gradually.
- **Final Output Layer**: The last layer should only have one node, and we don't have an activation layer since we are implementing WGAN, not GAN.
- **Leaky ReLU Activation**: Helps with gradient flow for low-activation inputs.
- **Gradient Penalty**: Regularizes the critic's gradient by addding gradient penalty to improve convergence.
- Print out your critic's model summary.

Note: You may refer to the generator model in *10_5_WGAN_GP_CeleA_Faces.ipynb*

[19]:
```
# Define the Critic
```

```
critic = None
```

[13]:

```
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 64, 64, 3)	0
conv2d (Conv2D)	(None, 32, 32, 64)	3,136
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	131,200
leaky_re_lu_5 (LeakyReLU)	(None, 16, 16, 128)	0
dropout (Dropout)	(None, 16, 16, 128)	0
conv2d_2 (Conv2D)	(None, 8, 8, 256)	524,544
leaky_re_lu_6 (LeakyReLU)	(None, 8, 8, 256)	0
dropout_1 (Dropout)	(None, 8, 8, 256)	0
conv2d_3 (Conv2D)	(None, 4, 4, 512)	2,097,664
leaky_re_lu_7 (LeakyReLU)	(None, 4, 4, 512)	0
dropout_2 (Dropout)	(None, 4, 4, 512)	0

```
conv2d_4 (Conv2D)                    (None, 1, 1, 1)                  ␣
↪8,193

flatten (Flatten)                    (None, 1)                        ␣
↪  0
```

Total params: 2,764,737 (10.55 MB)

Trainable params: 2,764,737 (10.55 MB)

Non-trainable params: 0 (0.00 B)

[13]:

1.4 2.3 WGAN_GP

Task 5: WGAN_GP:

You will define the WGAN_GP model class inherits *model.Model*. You may copy most of WGAN_GP model implemenation from *10_5_WGAN_GP_CeleA_Faces.ipynb*, but you need to understand how the following methods are implemented: - *wasserstein_loss*: Measures the Wasserstein distance between real and fake data distributions to guide training. - *gradient_penalty*: Regularizes the critic's gradients to enforce Lipschitz continuity for stable training. - *train_step*: Alternates between updating the critic (multiple steps) and generator (single step) to improve both models.

In addition, be consistent with your default parameters, such as: - Input Latent Space **dimension = 50** - Batch size = 128

[13]:
```
# Define the WGAN_GP model below

class WGAN_GP(Model):
    def __init__(self, generator, critic, clip_value=0.01,
                 n_critic=5, latent_dim=50, batch_size=128):
         pass


    def wasserstein_loss(self, y_true, y_pred):
        pass

    def compile(self, **kwargs):
        pass

    def train_step(self, real_imgs):
        pass
```

```
    def gradient_penalty(self, real_imgs, fake_imgs):
        pass
```

[15]:
```
# Initialize and compile WGAN
wgan_gp = WGAN_GP(generator=generator, critic=critic)
wgan_gp.compile()
```

1.5 3. Model training

[16]:
```
def plot_results(images, n_cols=None):
    '''visualizes fake images'''
    display.clear_output(wait=False)

    n_cols = n_cols or len(images)
    n_rows = (len(images) - 1) // n_cols + 1

    if images.shape[-1] == 1:
        images = np.squeeze(images, axis=-1)

    plt.figure(figsize=(n_cols, n_rows))

    for index, image in enumerate(images):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(np.clip(image, 0, 1))
        plt.axis("off")
```

Task 6: Train the wgan_gp model and generate the faked images:

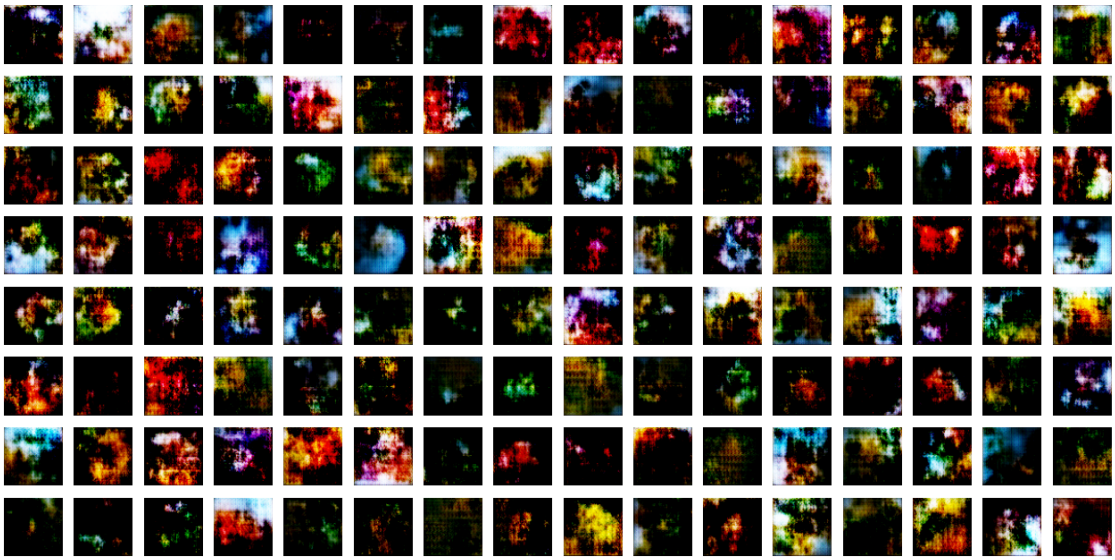
You may refer from *10_4_wgan_gp_mnist.ipynb* to see how to train the model. After each epoch, you will also generate 128 faked images and then do ploting using *plot_results* method implemented above.

Important: - It is computing intensive so the final results may not look good enough if epochs number is small - Suggested epochs = 100 for google colab - You may try large epochs such as 500 if you have GPUs installed on your local machine.

[10]:
```
# training your wgan_gp model and display 128 faked images
n_epochs = 100
```

[23]:
```

```



[17]: