Prog5_VAE_Flower_Colab

November 7, 2024

1 CPSC320: Program 5 - Variational Autoencoder (VAE) on Flower Dataset (Run on Google Colab)

In this programming assignment, you will build a VAE 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 *08_4_VAEs_celeba_colab.ipynb* and get most of he code from there. You need to make some adjustments wherever necessary.

[ ]:
```
# Import TensorFlow and Keras
import tensorflow as tf
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 import backend as K
import numpy as np
import matplotlib.pyplot as plt
```

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

```
Num GPUs Available:  1
Tue Oct 29 13:03:09 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                 9W /  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 4.

1.1.1 1.1 Upload Dataset to Google Colab

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

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

[ ]:
```python
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!
```

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.

[ ]:
```
# 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))
```

```
Found 3456 files.
<class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>
```

[ ]:
```

```

1.2.2 Preprocessing images Task 2: Preprocessing image

Preprocess the imput image so that it is normalized into the value between (0, 1) *i.e.*, divided by 255.0, and then return the normalized image back.

Important:, we not going to return a typical tuple (img, img), (img, label) as used in our autoencoder or classification model.

Why? If you adopt the VAE model from *08_4_VAEs_celeba_colab.ipynb train_step* method assumes data only conains imaage, not a tuple of (image,label), or (image,image)

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

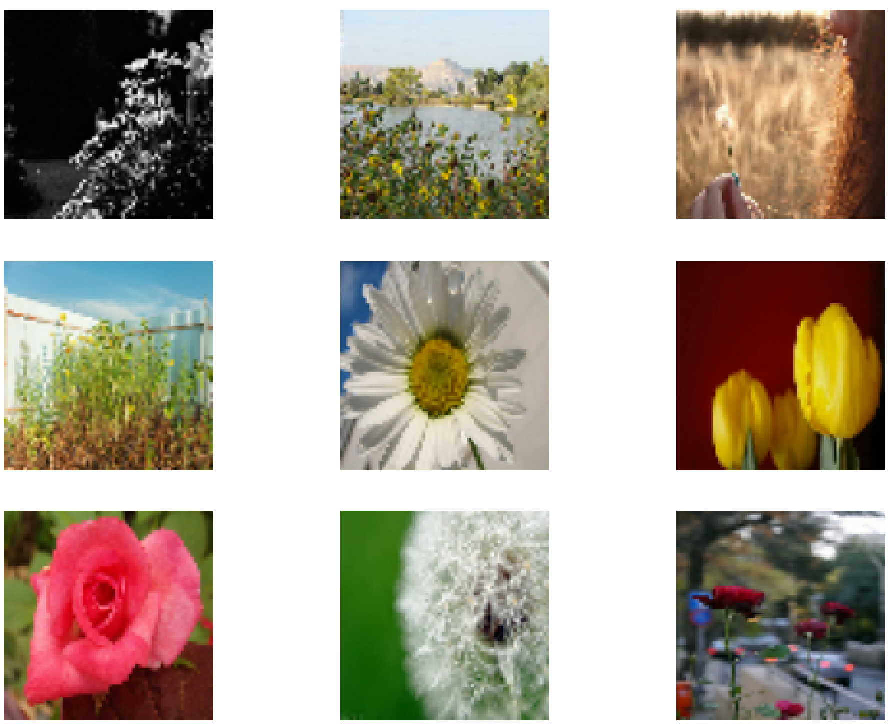
[ ]:
```
# the training dataset has been processed for model training
train_data = train_data.map(lambda x: preprocess(x))
```

[ ]:
```

```

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.

[ ]:
```
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 VAE Model

1.2.1 2.1 Sampling

Task 3: Define sampling class

First, you will build the `Sampling` class. This will be a custom Keras layer that will provide the Gaussian noise input along with the mean (mu) and standard deviation (sigma) of the encoder's output. In practice, the output of this layer is given by the equation:

$$z = \mu + e^{0.5\sigma} * \epsilon$$

where μ = mean, σ = logrithm of variance, and ϵ = random sample

[ ]:
```
class Sampling(tf.keras.layers.Layer):
  def call(self, inputs):
    """Generates a random sample and combines with the encoder output

    Args:
      inputs -- output tensor from the encoder

    Returns:
      tensor combined with a random sample
    """

    pass
```

[ ]:
```
```

1.2.2 2.2 The encoder

Task 4: Encoder:

Define your encoder (**on your own choices**) that compresses the input image to a lower-dimensional latent representation. - It uses Convolutional layers to reduce the spatial dimensions and a Flatten layer to create a latent vector. - The outputs are the mean and log variance vectors of the latent space. - Suggested 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 encoder model summary

Note: You may refer encoder model in *08_4_VAEs_celeba_colab.ipynb*

[ ]:
```
"""
## Step 4: Define the Encoder
The encoder compresses the input image to a lower-dimensional latent␣
↪representation.
It uses Convolutional layers to reduce the spatial dimensions and a Flatten␣
↪layer to create a latent vector.
The outputs are the mean and log variance vectors of the latent space.
"""
```

```python
# Encoder implementation here....
pass

encoder = None
```

[ ]: '\n## Step 4: Define the Encoder\nThe encoder compresses the input image to a lower-dimensional latent representation.\nIt uses Convolutional layers to reduce the spatial dimensions and a Flatten layer to create a latent vector.\nThe outputs are the mean and log variance vectors of the latent space.\n'

[ ]:
```python

```

1.3 2.2 The decoder

Task 5: Decoder:

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

- **Mirror the Encoder Architecture**. If the encoder uses convolutional layers, the decoder generally uses transposed convolutional layers (Conv2DTranspose).
- **Activation Functions**. The final layer should use an activation function that matches the data characteristics, i.e., Sigmoid for pixel values in the range [0, 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.

Print out your decoder's model summary.

[ ]:
```python
# Cell 5: Define the Decoder
"""
## Step 5: Define the Decoder
The decoder reconstructs the original image from the latent representation.
It uses Dense and Conv2DTranspose layers to upsample and reshape the latent␣
↪vector back to the original image shape.
"""

# Decoder

pass

decoder = None
```

```
Model: "functional"


 Layer (type)                          Output Shape                          ␣
↪Param #
```

```
decoder_input (InputLayer)          (None, 50)                                  ␣
↪  0

dense (Dense)                       (None, 8192)                              ␣
↪417,792

batch_normalization_3               (None, 8192)                              ␣
↪32,768
(BatchNormalization)                                                            ␣
↪

leaky_re_lu_3 (LeakyReLU)           (None, 8192)                                ␣
↪  0

reshape (Reshape)                   (None, 8, 8, 128)                           ␣
↪  0

conv2d_transpose (Conv2DTranspose)  (None, 16, 16, 64)                        ␣
↪73,792

batch_normalization_4               (None, 16, 16, 64)                          ␣
↪256
(BatchNormalization)                                                            ␣
↪

leaky_re_lu_4 (LeakyReLU)           (None, 16, 16, 64)                          ␣
↪  0

conv2d_transpose_1 (Conv2DTranspose)  (None, 32, 32, 32)                      ␣
↪18,464

batch_normalization_5               (None, 32, 32, 32)                          ␣
↪128
(BatchNormalization)                                                            ␣
↪

leaky_re_lu_5 (LeakyReLU)           (None, 32, 32, 32)                          ␣
↪  0

conv2d_transpose_2 (Conv2DTranspose)  (None, 64, 64, 16)                       ␣
↪4,624

batch_normalization_6               (None, 64, 64, 16)                          ␣
↪ 64
(BatchNormalization)                                                            ␣
↪
```

```
leaky_re_lu_6 (LeakyReLU)            (None, 64, 64, 16)
↪  0

conv2d_transpose_3 (Conv2DTranspose)   (None, 64, 64, 3)
↪435
```

```
Total params: 548,323 (2.09 MB)

Trainable params: 531,715 (2.03 MB)

Non-trainable params: 16,608 (64.88 KB)
```

[ ]:

```

```

1.4 2.3 Autoencoder

Task 6: VAE:

You will define VAE model class inherits model.Model. You may copy most of VAE model implemenation from *08_4_VAEs_celeba_colab.ipynb*, but you pay particular attention to the followings: - *train_step*: - For the reconstruction_loss: you may try multiply **10,000 or 15,000** by losses.MeanSquaredError()(data, reconstruction), because we calculate mean square error for 64X64X3=12288 pixels, so the total error is rougly 10,000 or 15,000. - *test_step*: you don't need it, so just remove it from the VAE class

[ ]:

```
"""
## Step 6: Define the VAE Class
The VAE class integrates the encoder, decoder, and sampling function.
The custom `train_step()` method handles forward pass and loss calculation.
"""
class VAE(models.Model):
    def __init__(self, encoder, decoder, **kwargs):
        pass

    @property
    def metrics(self):
        pass

    def call(self, inputs):
        """Call the model on a particular input."""
       pass

    def train_step(self, data):
        """Step run during training."""
```

```
        pass
```

[ ]:

```

```

1.5 3. Compile and Train the model

Task 7: Create a VAE model instance and Compile:

- Create an instance of the VAE class using the encoder and decoder models defined earlier.
- Compile it using the Adam optimizer.

[ ]:

```
# Create a VAE model instance and Compile

pass
```

[ ]:

```

```

Task 8: Model fit:

Do model fitting on your **train_data** from Section 1.2.3, set epochs to **500**.

[ ]:

```
"""
## Step 8: Train the VAE
We'll train the VAE using the dataset defined earlier.
"""
pass
```

[ ]:

```

```

1.6 4. Reconstruct using the variational autoencoder

[ ]:

```
def display(
    images, n=10, size=(20, 3), cmap="gray_r", as_type="float32", save_to=None
):
    """
    Displays n random images from each one of the supplied arrays.
    """
    if images.max() > 1.0:
        images = images / 255.0
    elif images.min() < 0.0:
        images = (images + 1.0) / 2.0

    plt.figure(figsize=size)
    for i in range(n):
        _ = plt.subplot(1, n, i + 1)
        plt.imshow(images[i].astype(as_type), cmap=cmap)
        plt.axis("off")

    if save_to:
```

```
        plt.savefig(save_to)
        print(f"\nSaved to {save_to}")

    plt.show()
```

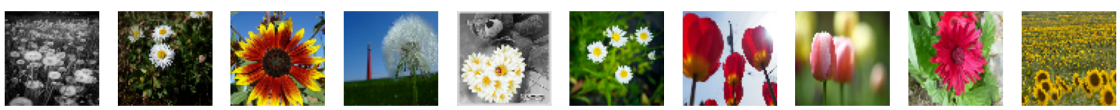
[ ]:

Task 9: Reconstruct images: - Select the first subset of the training set (done for you) - Create autoencoder predictions and display

[ ]:
```
# Select a subset of the training set
batches_to_predict = 1
example_images = np.array(
    list(train_data.take(batches_to_predict).get_single_element())
)

# Create autoencoder predictions and display
```

[ ]:

```
4/4              1s 3ms/step
Example real faces
```



```
Reconstructions
```

[ ]:

1.7 4. Generating New Images

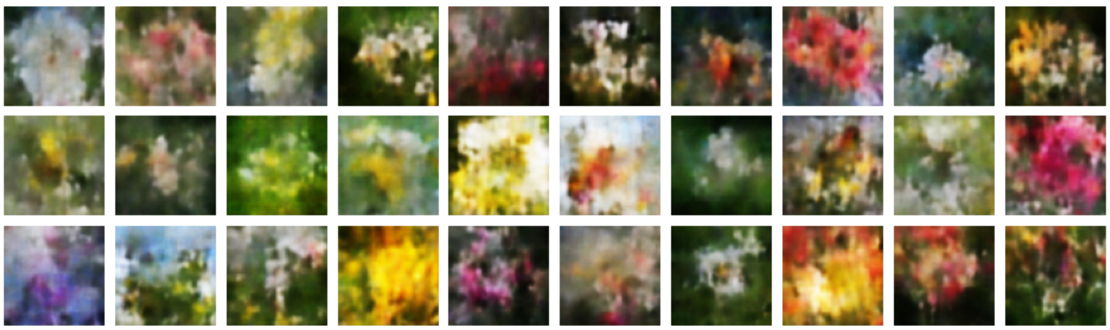
Task 10: Generating new images - Generate z_samples using *np.random.normal*, with sample size of grid_width * grid_height and **latent_dim = 50** (if defined 50 in your encoder model) - Decoder the sampled points and save them in reconstructions for image display.

[ ]:
```
# Sample some points in the latent space, from the standard normal distribution
grid_width, grid_height = (10, 3)
z_sample = None
```

```python
# Decode the sampled points
reconstructions = None
```

[ ]:
```python

```

[ ]:
```python
# Draw a plot of decoded images
fig = plt.figure(figsize=(12, 3.5))
fig.subplots_adjust(hspace=0.1, wspace=0.1)

# Output the grid of faces
for i in range(grid_width * grid_height):
    ax = fig.add_subplot(grid_height, grid_width, i + 1)
    ax.axis("off")
    ax.imshow(reconstructions[i, :, :])
```



[ ]:
```python

```