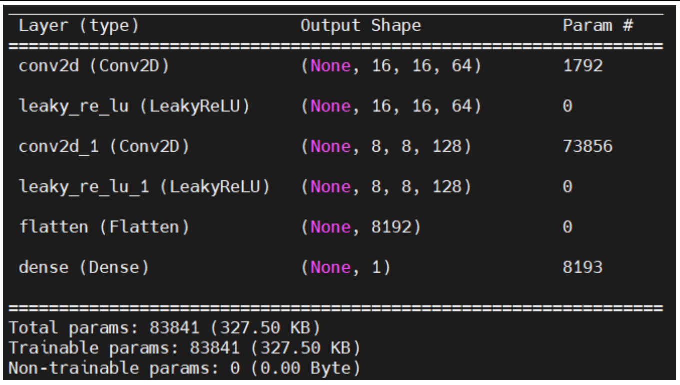
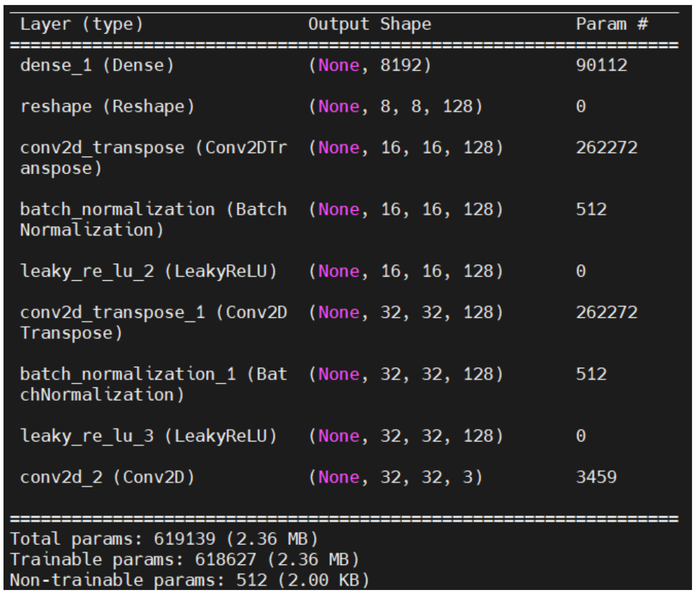
Principle of Artificial Intelligence HW3:

0811562 何祁恩 0810761 羅雯瀠 110511291 高可和 110511197 劉致廷

Part 1 GAN

1. Model Architecture

Model	Model Architecture
Discriminator	
Generator	

```
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 16, 16, 64)        1792

leaky_re_lu (LeakyReLU)     (None, 16, 16, 64)        0

conv2d_1 (Conv2D)           (None, 8, 8, 128)         73856

leaky_re_lu_1 (LeakyReLU)   (None, 8, 8, 128)         0

flatten (Flatten)           (None, 8192)              0

dense (Dense)               (None, 1)                 8193

=================================================================
Total params: 83841 (327.50 KB)
Trainable params: 83841 (327.50 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Layer (type)                Output Shape              Param #
=================================================================
dense_1 (Dense)             (None, 8192)              90112

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

conv2d_transpose (Conv2DTr  (None, 16, 16, 128)       262272
anspose)

batch_normalization (Batch  (None, 16, 16, 128)       512
Normalization)

leaky_re_lu_2 (LeakyReLU)   (None, 16, 16, 128)       0

conv2d_transpose_1 (Conv2D  (None, 32, 32, 128)       262272
Transpose)

batch_normalization_1 (Bat  (None, 32, 32, 128)       512
chNormalization)

leaky_re_lu_3 (LeakyReLU)   (None, 32, 32, 128)       0

conv2d_2 (Conv2D)           (None, 32, 32, 3)         3459

=================================================================
Total params: 619139 (2.36 MB)
Trainable params: 618627 (2.36 MB)
Non-trainable params: 512 (2.00 KB)
```

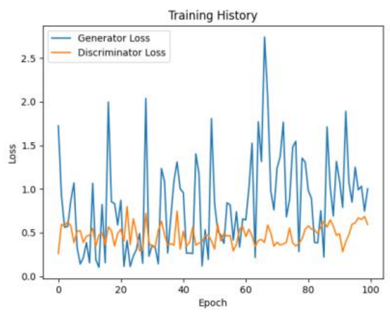
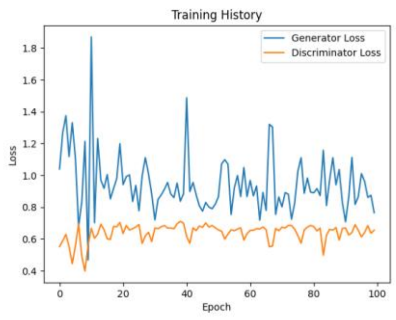
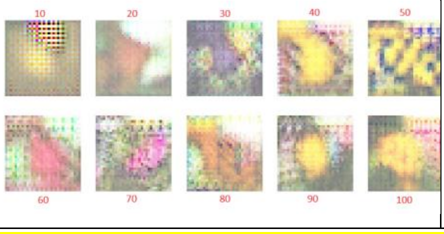
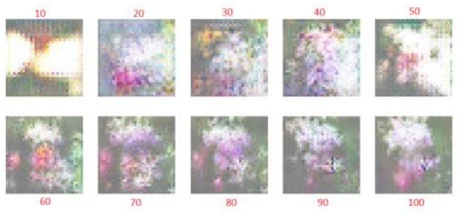
For the first part of the model, I am training a Generative Adversarial Network (GAN) to generate 32x32 colored flower images. Initially, I employ the TensorFlow data loader to load 16 types of flower images and down-sample them. There are two ways to utilize the data loader. Firstly, for enhanced training of the generator model with finer details, I use 256x256 images as training data, generating 32x32 fake flower data for output, and down-sampling the real flower data to train the discriminator. However, due to limited computational resources, I opt to down-sample directly to the desired output shape to save on training time.

2. Model Parameter

$$\begin{cases} iteration\ T = 100 \\ learning\ rate\ h = 0.0002 \\ beta\ \beta = 0.5 \\ loss\ \ function\ Loss = Binary\ Crossentropy \\ optimizer = \ Adam \\ latent\ dim = 10\ /\ 256 \end{cases}$$

3. Training Loss & History

- (20 pts) With **latent_dim** = **10**, **epochs** = **100** show the training history (loss of generator and discriminator) and the generated figure every 10 epoch. The input should always be `tf.random.normal(shape = (1, latent_dim), seed = 1)`
- (10 pts) Change the **latent_dim** = **256**, and repeat the above process.

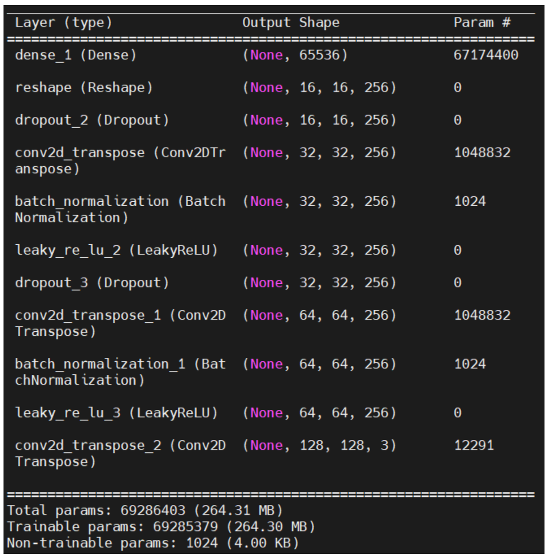
	Latent Dim = 10	Latent Dim = 256
Training Loss		
Training History		

其實結果遠看也蠻像花的啦 XD 至少背景綠綠的 花蠻多顏色蠻漂亮的

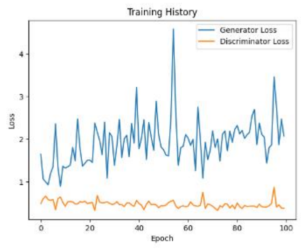
Based on the training loss above, it can be observed that our generator is not trained very effectively. In contrast, the discriminator appears to be smoother. The specific reason for this is that there are too many types of training data, and upon reviewing internal images, the quality of the photos is uneven. Therefore, the quality of the data is one of the factors that can impact the results of the generator. Additionally, we encountered the issue of discriminator winning mentioned in class. No matter how hard the generator is trained, the discriminator always manages to recognize it as a fake image rather than a real image. Moving on to the training history, it can be observed that some contours are quickly learned at the very beginning. In the case of a larger latent dimension, contours resembling flowers can be generated more quickly.

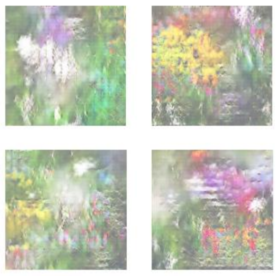
4. High Resolution Training Loss & History

- (10 pts) Try to generated higher resolution pictures (128, 128, 3), and show 4 figures at the end. The hyperparameters are up to you.



```
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 65536)             67174400

 reshape (Reshape)           (None, 16, 16, 256)       0

 dropout_2 (Dropout)         (None, 16, 16, 256)       0

 conv2d_transpose (Conv2DTr  (None, 32, 32, 256)       1048832
 anspose)

 batch_normalization (Batch  (None, 32, 32, 256)       1024
 Normalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 256)       0

 dropout_3 (Dropout)         (None, 32, 32, 256)       0

 conv2d_transpose_1 (Conv2D  (None, 64, 64, 256)       1048832
 Transpose)

 batch_normalization_1 (Bat  (None, 64, 64, 256)       1024
 chNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 64, 64, 256)       0

 conv2d_transpose_2 (Conv2D  (None, 128, 128, 3)       12291
 Transpose)

=================================================================
Total params: 69286403 (264.31 MB)
Trainable params: 69285379 (264.30 MB)
Non-trainable params: 1024 (4.00 KB)
```





Basically, it's the same as the previous steps, but with dropout applied. Without dropout, the model struggles to train, and it helps produce higher resolution photos, reducing the grid-like discontinuity in the output.

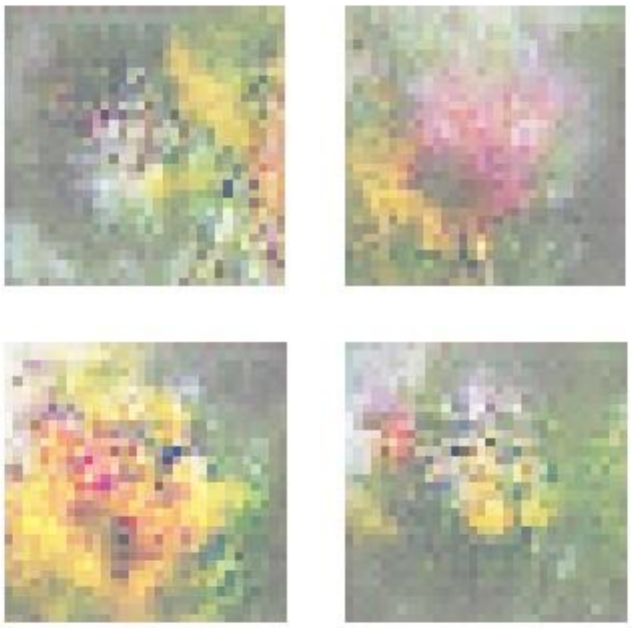
➢ Optimized Model Structure

Originally, I used the same model structure except for the input layer; however, the model couldn't be trained effectively. The output images showed some strange edges and artifacts. By monitoring the generator and discriminator losses, I found that the generator loss was very high compared to the discriminator loss. Therefore, I suspected encountering an issue of discriminator winning or overfitting. To address this, I tuned the learning rate and added some dropout layers. Additionally, to resolve the vanishing gradient issue, I introduced batch normalization.

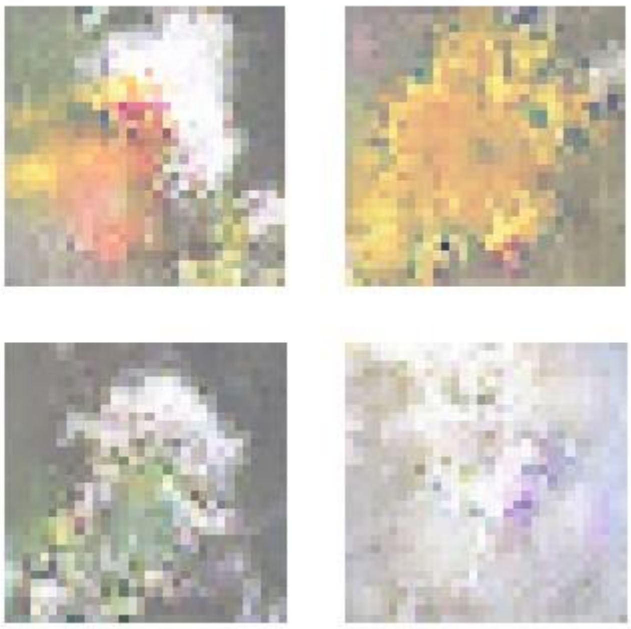
➢ Discriminator Winning Solution

To tackle the discriminator winning problem, instead of training the discriminator with real and fake image batches separately, I merged the real and fake images together. This approach makes it harder for the discriminator to distinguish between real and fake images. Furthermore, as the discriminator model is essentially a regression or classification model, I labeled the real images as 0.9 instead of 1. By doing so, I hoped that the discriminator would not be overly confident in identifying real images, further addressing the discriminator winning issue.

Another Output Image for GAN Latent Dim = 10

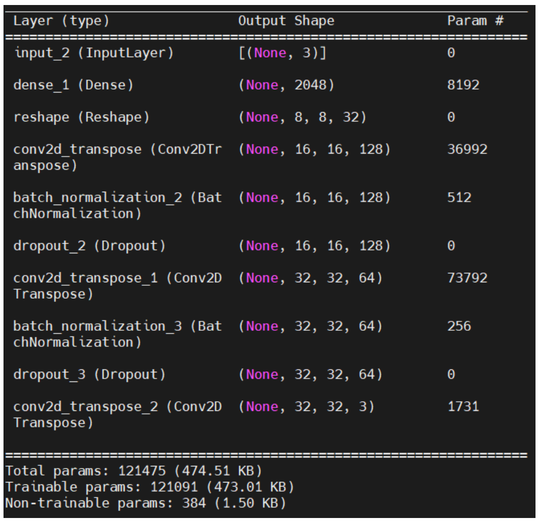
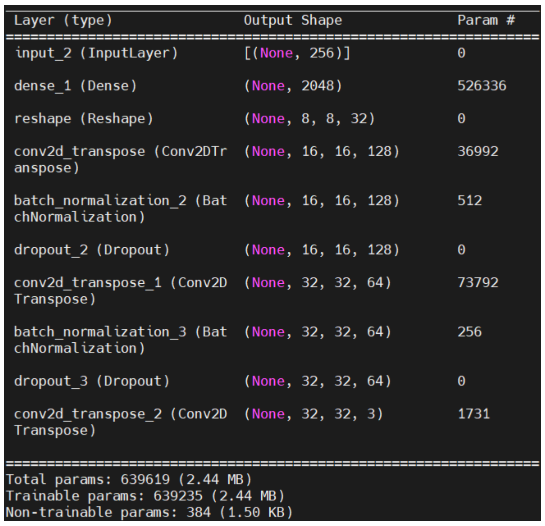


Another Output Image for GAN Latent Dim = 256



Part 2 VAE

1. Model Architecture

Latent Dim = 3	Latent Dim = 256
	

```
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 3)]               0

dense_1 (Dense)             (None, 2048)              8192

reshape (Reshape)           (None, 8, 8, 32)          0

conv2d_transpose (Conv2DTr  (None, 16, 16, 128)       36992
anspose)

batch_normalization_2 (Bat  (None, 16, 16, 128)       512
chNormalization)

dropout_2 (Dropout)         (None, 16, 16, 128)       0

conv2d_transpose_1 (Conv2D  (None, 32, 32, 64)        73792
Transpose)

batch_normalization_3 (Bat  (None, 32, 32, 64)        256
chNormalization)

dropout_3 (Dropout)         (None, 32, 32, 64)        0

conv2d_transpose_2 (Conv2D  (None, 32, 32, 3)         1731
Transpose)

=================================================================
Total params: 121475 (474.51 KB)
Trainable params: 121091 (473.01 KB)
Non-trainable params: 384 (1.50 KB)
```

```
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 256)]             0

dense_1 (Dense)             (None, 2048)              526336

reshape (Reshape)           (None, 8, 8, 32)          0

conv2d_transpose (Conv2DTr  (None, 16, 16, 128)       36992
anspose)

batch_normalization_2 (Bat  (None, 16, 16, 128)       512
chNormalization)

dropout_2 (Dropout)         (None, 16, 16, 128)       0

conv2d_transpose_1 (Conv2D  (None, 32, 32, 64)        73792
Transpose)

batch_normalization_3 (Bat  (None, 32, 32, 64)        256
chNormalization)

dropout_3 (Dropout)         (None, 32, 32, 64)        0

conv2d_transpose_2 (Conv2D  (None, 32, 32, 3)         1731
Transpose)

=================================================================
Total params: 639619 (2.44 MB)
Trainable params: 639235 (2.44 MB)
Non-trainable params: 384 (1.50 KB)
```

Variational Auto Encoder (VAE) is harder to train compared to the GAN. We attempted to create a simple model with batch normalization to prevent vanishing gradients and included a dropout layer to prevent overfitting. Due to the simplicity of our model, we were able to train it effectively in just 100 epochs.

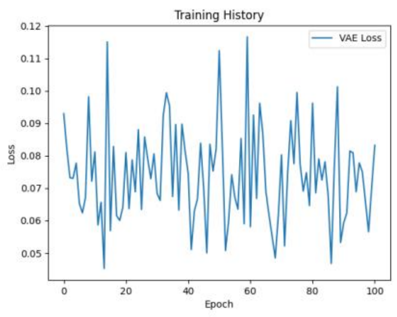
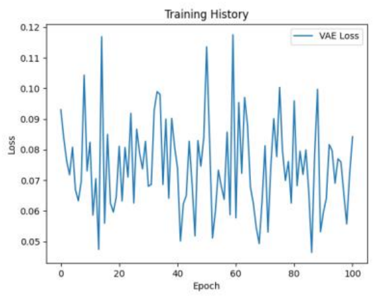
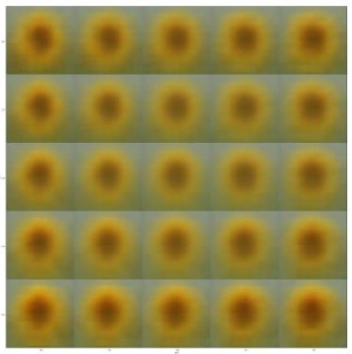
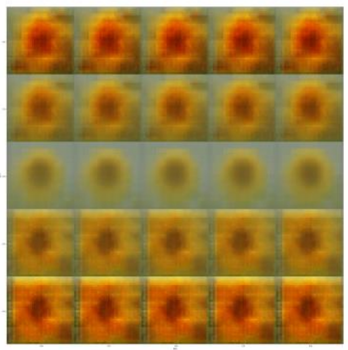
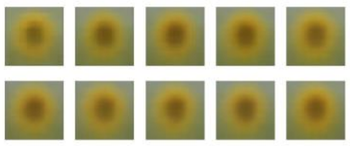
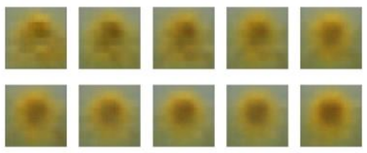
2. Model Parameter

$$\begin{cases} iteration\ T = 100 \\ learning\ rate\ \eta = 0.001 \\ loss\ \ function\ Loss = KL + Reconstruction \\ optimizer = \ Adam \\ latent\ dim = 3\ /\ 256 \end{cases}$$

Adjusting the learning rate is crucial in the VAE model training process. Since the TA instructed us to train the VAE model in only 100 iterations, a too-small learning rate might hinder the successful learning of image features. Conversely, a too-large learning rate may result in overfit images. Therefore, I carefully adjusted the learning rate to find the best fit for my VAE model. We selected the sunflower, with its straightforward flower structure, as our dataset. We hope to achieve excellent results!

3. Training Loss & History

- (20 pts) With **latent_dim** = **3**, **epochs** = **100** show the training history (loss of KL and reconstruction) and the generated figure every 10 epochs. The input should always be the same as Part 1. Plot the figure like Figure 1 with the code in file, and plot the specific class of flower (one figure is enough) like Figure 2.
- (10 pts) Now let the **latent_dim** = **256**, and repeat the above process.

	Latent Dim = 3	Latent Dim = 256
Training Loss		
Output Result		
Different Latent Result		
Training History		

Note that the VAE loss = KL loss + Reconstruction Loss.

- (10 pts) Explain the reparameterization operation at the bottleneck and how it can replace the original sampling and do the backpropgation? Derive the gradient $\frac{\partial \mu}{\partial L}, \frac{\partial \sigma}{\partial L} =?$

The reparameterization trick is a technique used in training probabilistic models, particularly those involving variational autoencoders (VAEs). It addresses the challenge of backpropagation through the sampling operation, which is a non-differentiable operation. This trick introduces a differentiable transformation during the sampling process, allowing gradients to flow through the model during training.

Consider a generic probabilistic model where we have a random variable z with a probability distribution parameterized by some learnable parameter ϕ. The sampling operation is typically denoted as $z = g(z \mid \phi)$, where g is a probability distribution.

In the context of VAE, the typical bottleneck layer involves sampling from Gaussian distribution with a mean μ and standard deviation σ. The reparameterization trick replaces the sampling operation with a differentiable transformation.

➢ Original Sampling

$$z \sim N(\mu, \sigma^2)$$

➢ Reparameterization Trick

$$z = \mu + \sigma \odot \varepsilon$$

where

$$\varepsilon \sim N(0,1)$$

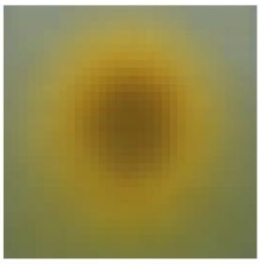
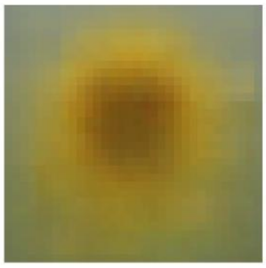
In this way, z is now a deterministic function of $\mu, \sigma \,\&\, \varepsilon$. And therefore, the reparameterization trick helps during backpropagation because the gradient of the sampled variable z is straightforward to compute. Denote the loss as L, the partial derivative an be computed by chain rules as follows:

$$\begin{cases} \dfrac{\partial L}{\partial \mu} = \dfrac{\partial L}{\partial z} \cdot \dfrac{\partial z}{\partial \mu} = \dfrac{\partial L}{\partial z} \\ \dfrac{\partial L}{\partial \sigma} = \dfrac{\partial L}{\partial z} \cdot \dfrac{\partial z}{\partial \sigma} = \dfrac{\partial L}{\partial z} \cdot \varepsilon \end{cases}$$

Therefore, the gradient can flow through the parameterization trick, making it possible to perform backpropagation and optimize the model parameters through gradient descent.

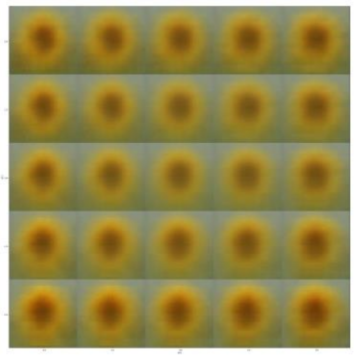
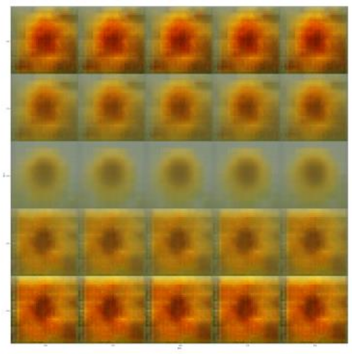
4. Result Analysis

➢ How different latent dim affect the model?

	Latent Dim = 3	Latent Dim = 256
Output Result		

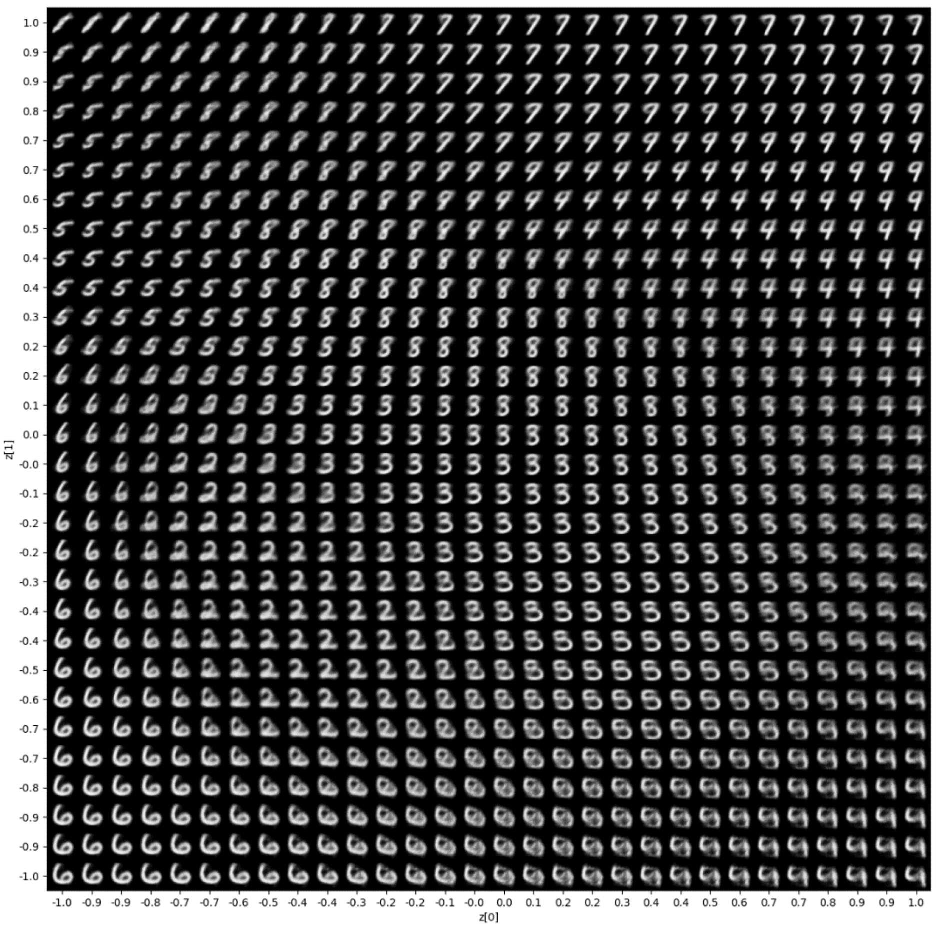
The latent variables in a VAE model represent a compressed and continuous representation of the input data. Each dimension of the latent space can be thought of as capturing a different aspect or feature of the data. As you can see, the higher latent space (256) can provide more flexibility. However, in the training process, the higher latent dim can also lead to overfitting if not properly regularized. Besides, the sunflower in the right-hand side, has deeper and express more detail in the middle of the flower. Furthermore, by sampling different points in the latent space, the VAE can generate diverse and realistic samples.

➢ Latent Space Analysis

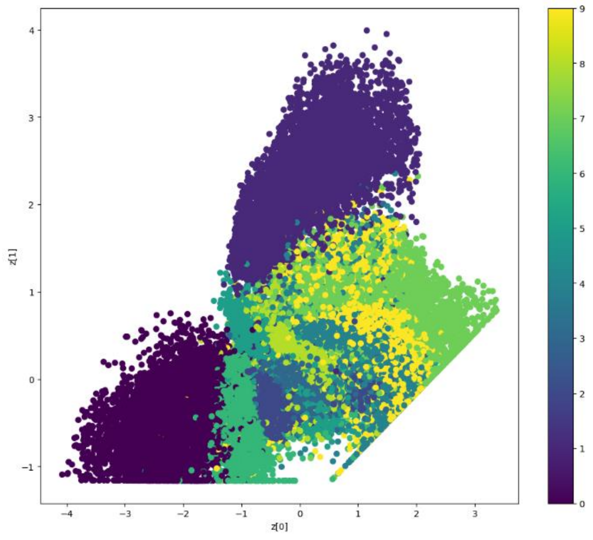
	Latent Dim = 3	Latent Dim = 256
Output Result		

Since we only choose 1 type to train the VAE model, rather than many types of flower. Therefore, for the different latent space, the result will not have big difference. And therefore, rather than direct sample in standard normal distribution, I also scale it to perform the higher difference in the latent vector. However, if the model is well-trained, the model might be more sensitivity to the latent vector.

In Keras document of VAE, the latent space of the MINST dataset are as follow:



As you can see, given that there are multiple types of datasets, providing distinct continuous latent vectors as input is likely to yield a spectrum of results. This allows us to observe how the input latent vectors affect the VAE model."



Also, we can observe how the latent space clusters different type of class.

Reference: [https://keras.io/examples/generative/vae/]

Part 3 Combination

- (20 pts) Each method above has its pros and cons. Please google methods that combine both models' advantages. Choose one and briefly explain how it works.

GAN and VAE each have their own advantages and disadvantages, and the choice between using GAN or VAE depends on the user's needs. If the focus is on generating high-quality, realistically strong samples and there is no need to model the probability of the samples, then GAN may be a better choice.

Compared to GAN, VAE has more opportunities to create novel samples. VAE has the advantage of autoencoding, preserving essential features of the data, resulting in generated samples with similar structures. Therefore, if there is a need to model uncertainty or higher requirements in representation learning, VAE may be more suitable.

Many models have incorporated the characteristics of both GAN and VAE, combining their strengths. Examples include AAE (Adversarial Autoencoder), CVAE-GAN (Conditional Variational Autoencoder GAN), and VAE-GAN (Variational Autoencoder GAN).

CVAE-GAN (Conditional Variational Autoencoder Generative Adversarial Network) introduces the concept of conditional generation compared to VAE-GAN. This means that during the generation process, additional conditional information is introduced. This helps the model consider specific attributes or conditions of the input data during training and generation, aiding in the generation of samples with specific properties.

CVAE-GAN is a hybrid model that combines elements of CVAE and GAN, capable of generating diverse and conditionally high-quality samples. The CVAE part involves the encoder converting the input data and conditional information into latent representations. The decoder then receives the latent representations and decodes them to generate reconstructed data, minimizing the loss caused by the reconstructed data to preserve the input data's appearance.

The GAN part involves the generator receiving the latent representations and conditional information, generating data corresponding to samples in the latent space. The generator aims to generate samples with specific attributes, and the discriminator distinguishes between real and generated data. Through the competitive process between the two, losses are minimized to ensure that the generator continually improves, making it challenging for the discriminator to differentiate between real and generated samples.

In CVAE-GAN, in addition to the original input data, there is also additional conditional information, meaning that the process of generating samples is influenced by the conditional information. The generated samples not only contain the structure of the latent representations but also exhibit attributes specific to certain conditions.

Part 4 Suggestion

Some suggestion that I think can make this homework better in the future.

- **Reduce the dataset type in part1**

  In the GAN part, as TA has instructed us to use the entire dataset for training the model, it becomes challenging for us to visually determine the reasonableness of the results and conduct a thorough analysis on the output images. By reducing the number of types to just 5 or less, students may observe combinations such as sunflowers, roses, and tulips. This would make it more reasonable for discussing the results. Just imagine trying to mentally combine 16 different types of flowers—I have no idea what the outcome would be, and I believe the model faces the same challenge.

- **Don’t fix the parameter in epochs**

  For 2 part, as the TA has instructed us to limit the training to only 100 epochs, I am concerned that a denser model with dropout might result in underfitting. Consequently, I need to simplify my model to facilitate more effective training.

- **Open discussion in E3**

  To minimize duplicated discussions, it might be more clear and efficient to open a dedicated E3 discussion for each homework, allowing groups to share information more effectively."

Part 5 Summary

We implement a Generative Adversarial Network (GAN) and a Variational Autoencoder (VAE) for generating 32x32 colored flower images. Challenges in GAN training are observed, and VAE is explored with considerations for latent dimensions and the reparameterization trick.