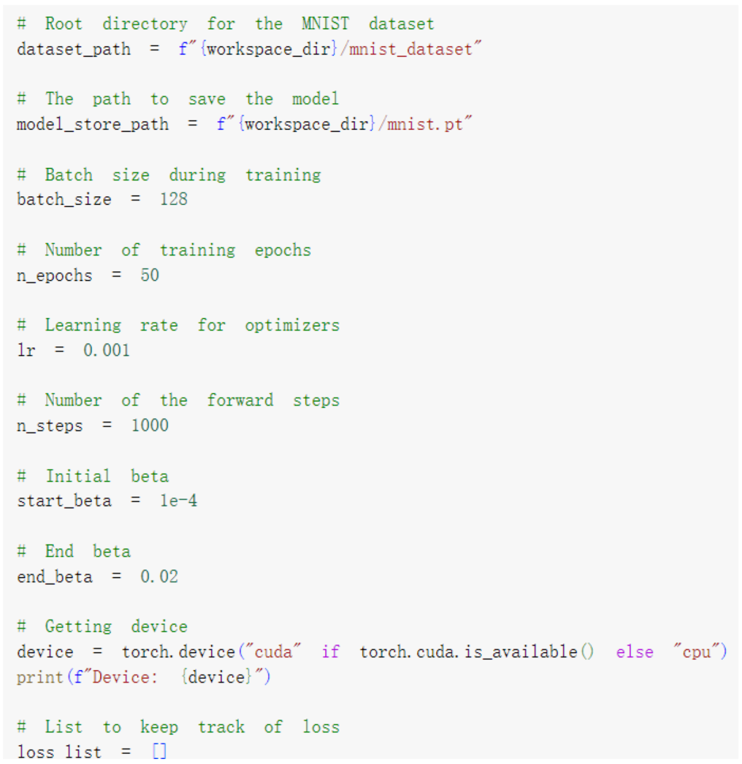
IMVFX HW2-2

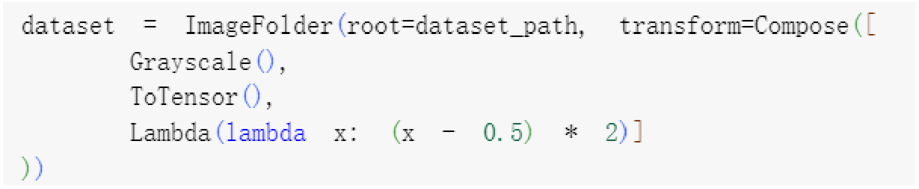
1. Please provide a brief introduction about your experiments on the MNIST and Anime Face dataset, including details such as setting of hyperparameter, data augmentation techniques used, network structure, etc

   (1) MNIST dataset



```
# Root directory for the MNIST dataset
dataset_path = f"{workspace_dir}/mnist_dataset"

# The path to save the model
model_store_path = f"{workspace_dir}/mnist.pt"

# Batch size during training
batch_size = 128

# Number of training epochs
n_epochs = 50

# Learning rate for optimizers
lr = 0.001

# Number of the forward steps
n_steps = 1000

# Initial beta
start_beta = 1e-4

# End beta
end_beta = 0.02

# Getting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Device: {device}")

# List to keep track of loss
loss_list = []
```

上圖是我在 MNIST dataset 的 diffision model 的超參數設定，除了 n_epoch 改為 50 之外，其他超參數都跟助教的 sample code[3]一樣沒有變動。



```
dataset = ImageFolder(root=dataset_path, transform=Compose([
        Grayscale(),
        ToTensor(),
        Lambda(lambda x: (x - 0.5) * 2)]
))
```

```
class UNet(nn.Module):
    def __init__(self, n_steps=1000, time_embedding_dim=256):
        super(UNet, self).__init__()

        # Time embedding
        self.time_step_embedding = nn.Embedding(n_steps, time_embedding_dim)
        self.time_step_embedding.weight.data = time_embedding(n_steps, time_embedding_dim)
        self.time_step_embedding.requires_grad_(False)

        # The first half
        self.time_step_encoder1 = nn.Sequential(
            nn.Linear(time_embedding_dim, 1),
            nn.SiLU(),
            nn.Linear(1, 1)
        )

        self.block1 = nn.Sequential(
            nn.LayerNorm((1, 28, 28)),
            nn.Conv2d(1, 8, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(8, 8, kernel_size=3, stride=1, padding=1),
            nn.LeakyReLU(0.2),
        )
        self.down1 = nn.Conv2d(8, 8, 4, 2, 1)

        self.time_step_encoder2 = nn.Sequential(
            nn.Linear(time_embedding_dim, 8),
            nn.SiLU(),
            nn.Linear(8, 8)
        )
```

```
        self.block2 = nn.Sequential(
            nn.LayerNorm((8, 14, 14)),
            nn.Conv2d(8, 16, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=1),
            nn.LeakyReLU(0.2),
        )
        self.down2 = nn.Conv2d(16, 16, 4, 2, 1)

        self.time_step_encoder3 = nn.Sequential(
            nn.Linear(time_embedding_dim, 16),
            nn.SiLU(),
            nn.Linear(16, 16)
        )

        self.block3 = nn.Sequential(
            nn.LayerNorm((16, 7, 7)),
            nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.LeakyReLU(0.2),
        )
        self.down3 = nn.Sequential(
            nn.Conv2d(32, 32, 2, 1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(32, 32, 4, 2, 1)
        )
```

```
# The bottleneck
self.time_step_encoder_mid = nn.Sequential(
        nn.Linear(time_embedding_dim, 32),
        nn.SiLU(),
        nn.Linear(32, 32)
)

self.block_mid = nn.Sequential(
        nn.LayerNorm((32, 3, 3)),
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
        nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
        nn.LeakyReLU(0.2),
)

# The second half
self.up1 = nn.Sequential(
        nn.ConvTranspose2d(32, 32, kernel_size=4, stride=2, padding=1),
        nn.LeakyReLU(0.2),
        nn.ConvTranspose2d(32, 32, 2, 1)
)

self.time_step_encoder4 = nn.Sequential(
        nn.Linear(time_embedding_dim, 64),
        nn.SiLU(),
        nn.Linear(64, 64)
)
```

```
self.block4 = nn.Sequential(
        nn.LayerNorm((64, 7, 7)),
        nn.Conv2d(64, 16, kernel_size=3, stride=1, padding=1),
        nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=1),
        nn.LeakyReLU(0.2),
)

self.up2 = nn.ConvTranspose2d(16, 16, 4, 2, 1)

self.time_step_encoder5 = nn.Sequential(
        nn.Linear(time_embedding_dim, 32),
        nn.SiLU(),
        nn.Linear(32, 32)
)

self.block5 = nn.Sequential(
        nn.LayerNorm((32, 14, 14)),
        nn.Conv2d(32, 8, kernel_size=3, stride=1, padding=1),
        nn.Conv2d(8, 8, kernel_size=3, stride=1, padding=1),
        nn.LeakyReLU(0.2),
)

self.up3 = nn.ConvTranspose2d(8, 8, 4, 2, 1)
```

```
self.time_step_encoder6 = nn.Sequential(
        nn.Linear(time_embedding_dim, 16),
        nn.SiLU(),
        nn.Linear(16, 16)
)
self.block6 = nn.Sequential(
        nn.LayerNorm((16, 28, 28)),
        nn.Conv2d(16, 8, kernel_size=3, stride=1, padding=1),
        nn.Conv2d(8, 8, kernel_size=3, stride=1, padding=1),
        nn.LeakyReLU(0.2),
        nn.LayerNorm((8, 28, 28)),
        nn.Conv2d(8, 8, kernel_size=3, stride=1, padding=1),
        nn.Conv2d(8, 8, kernel_size=3, stride=1, padding=1),
        nn.LeakyReLU(0.2),
)

self.final_layer = nn.Conv2d(8, 1, 3, 1, 1)
```

```
def forward(self, x, t):
        t = self.time_step_embedding(t)
        n = len(x)
        output1 = self.block1(x + self.time_step_encoder1(t).reshape(n, -1, 1, 1))
        output2 = self.block2(self.down1(output1) + self.time_step_encoder2(t).reshape(n, -1, 1, 1))
        output3 = self.block3(self.down2(output2) + self.time_step_encoder3(t).reshape(n, -1, 1, 1))
        output_mid = self.block_mid(self.down3(output3) + self.time_step_encoder_mid(t).reshape(n, -1, 1, 1))
        output4 = torch.cat((output3, self.up1(output_mid)), dim=1)
        output4 = self.block4(output4 + self.time_step_encoder4(t).reshape(n, -1, 1, 1))
        output5 = torch.cat((output2, self.up2(output4)), dim=1)
        output5 = self.block5(output5 + self.time_step_encoder5(t).reshape(n, -1, 1, 1))
        output6 = torch.cat((output1, self.up3(output5)), dim=1)
        output6 = self.block6(output6 + self.time_step_encoder6(t).reshape(n, -1, 1, 1))
        output = self.final_layer(output6)
        return output
```

data augmentation 跟 U-NET 的 network structure 也一樣都跟助教的 sample code[3]一樣。

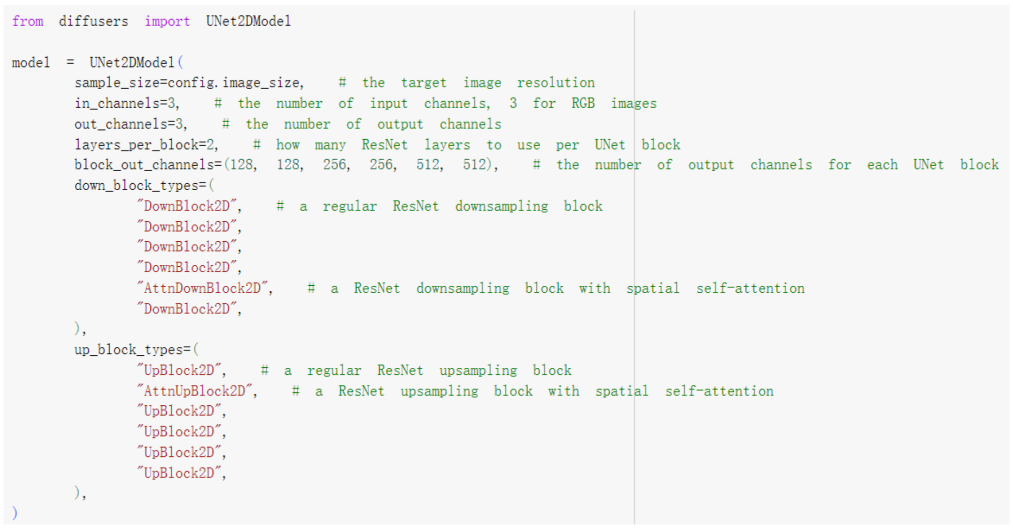
(2) Anime Face

```
from dataclasses import dataclass


@dataclass
class TrainingConfig:
        image_size = 64  # the generated image resolution
        train_batch_size = 64
        eval_batch_size = 16  # how many images to sample during evaluation
        num_epochs = 50
        gradient_accumulation_steps = 1
        learning_rate = 1e-4
        lr_warmup_steps = 500
        save_image_epochs = 10
        save_model_epochs = 10
        mixed_precision = "fp16"  # `no` for float32, `fp16` for automatic mixed precision
        output_dir = "anime_face"  # the model name locally and on the HF Hub
        push_to_hub = False
        hub_private_repo = False
        overwrite_output_dir = True  # overwrite the old model when re-running the notebook
        seed = 0


config = TrainingConfig()
```

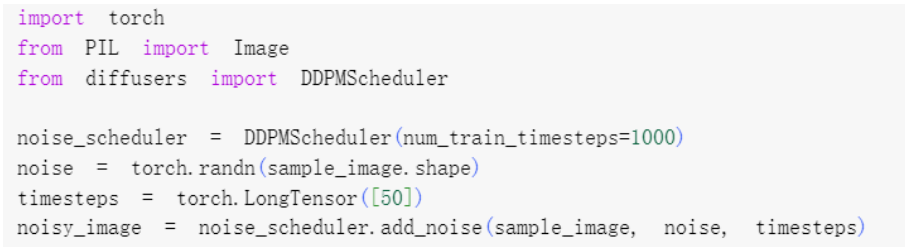
上圖是我在 Anime Face 的 diffision model 的超參數設定，這個 diffusion model 我是按照 hugging face[1]的教學建構的，我將 image size 設成 anime face data set 的 64*64 尺寸，training 的 batch size 設成 64，總共訓練 50 個 epoch。

```
from torchvision import transforms

preprocess = transforms.Compose(
        [
                transforms.Resize((config.image_size, config.image_size)),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize([0.5], [0.5]),
        ]
)
```

在 data augmentation 的部分使用跟 DCGAN 作業一樣的 RandomHorizontalFlip，這部分是希望能夠在最後兩個 model 比較結果時比較公平，resize 是確保輸入圖片尺寸皆是 64*64。



```
from  diffusers  import  UNet2DModel

model  =  UNet2DModel(
        sample_size=config.image_size,    #  the  target  image  resolution
        in_channels=3,    #  the  number  of  input  channels,  3  for  RGB  images
        out_channels=3,    #  the  number  of  output  channels
        layers_per_block=2,    #  how  many  ResNet  layers  to  use  per  UNet  block
        block_out_channels=(128,  128,  256,  256,  512,  512),    #  the  number  of  output  channels  for  each  UNet  block
        down_block_types=(
                "DownBlock2D",    #  a  regular  ResNet  downsampling  block
                "DownBlock2D",
                "DownBlock2D",
                "DownBlock2D",
                "AttnDownBlock2D",    #  a  ResNet  downsampling  block  with  spatial  self-attention
                "DownBlock2D",
        ),
        up_block_types=(
                "UpBlock2D",    #  a  regular  ResNet  upsampling  block
                "AttnUpBlock2D",    #  a  ResNet  upsampling  block  with  spatial  self-attention
                "UpBlock2D",
                "UpBlock2D",
                "UpBlock2D",
                "UpBlock2D",
        ),
)
```

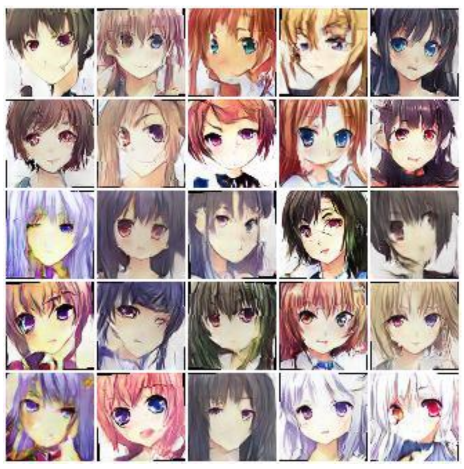
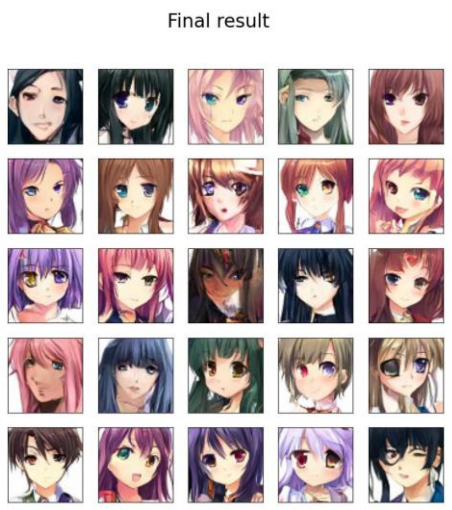


```
import  torch
from  PIL  import  Image
from  diffusers  import  DDPMScheduler

noise_scheduler  =  DDPMScheduler(num_train_timesteps=1000)
noise  =  torch.randn(sample_image.shape)
timesteps  =  torch.LongTensor([50])
noisy_image  =  noise_scheduler.add_noise(sample_image,  noise,  timesteps)
```

上圖是 UNet 架構跟 DDPMScheduler，一樣是按照[1]的教學建構的，num_inference_steps 意思就是 number of denoising steps，這部分我設 1000，至於 DDPMScheduler 上面沒有顯示的 beta_schedule，我是用預設的 linear，因為之前用過其他 dataset 訓練 ddpm 時，發現選 linear 的訓練速度比選用 squaredcos_cap_v2 快。

2. Comparing the generation quality between DCGAN and DDPM: Compare the resolution, level of detail, and diversity of generated images. You can assess them using metrics such as FID, IS, or subjective evaluations. Encourage writing more about the experiment you want to discuss.

DCGAN	DDPM

	
FID	FID
FID: 45.21080927104859	FID: 19.929999768139993

我使用了 pytorch-fid[2]來評估 DCGAN、DDPM 模型生成圖像的質量。FID 是一個衡量生成圖像與真實圖像相似度的指標，基於兩者在特徵空間上的距離，所以較低的 FID 分數意味著更高的圖像質量。在計算 FID 的實驗中我使用了整個 anime face dataset 來評估 DCGAN 模型，並生成了相同數量的圖片進行比較。但是對於 DDPM 模型，由於計算資源的限制，我只能使用 dataset 的前 10000 張圖片，同時 DDPM 也只生成了 10000 張圖片進行比較，另外我在 DDPM 模型的設置中將 num_inference_steps 設定改為 100，而不是上面的 final result 圖原本的 1000(因為要生成 10000 張圖片，再加上因為 colab 運算資源有限，沒辦法設成 1000)，且 DDPM 只訓練了 50 個 epoch（而 DCGAN 則訓練了 200 個 epoch），但從 FID 分數來看，以及從最終結果圖片的解析度、多樣性和細節（特別是眼睛和頭髮的細節）來看，DDPM 的表現顯著優於 DCGAN。這結果表示 DDPM 在捕捉細節和生成高質量圖像方面可能比 DCGAN 更有優勢，尤其是在有限的訓練情況下。

Reference

[1] https://huggingface.co/docs/diffusers/tutorials/basic_training

[2] https://github.com/mseitzer/pytorch-fid

[3] Sample Code : IMVFX_HW2_DDPM.ipynb - Colaboratory (google.com)