Report of HW5

Huanyu Wang 522030910212

Task 1

The final best parameters of my model:

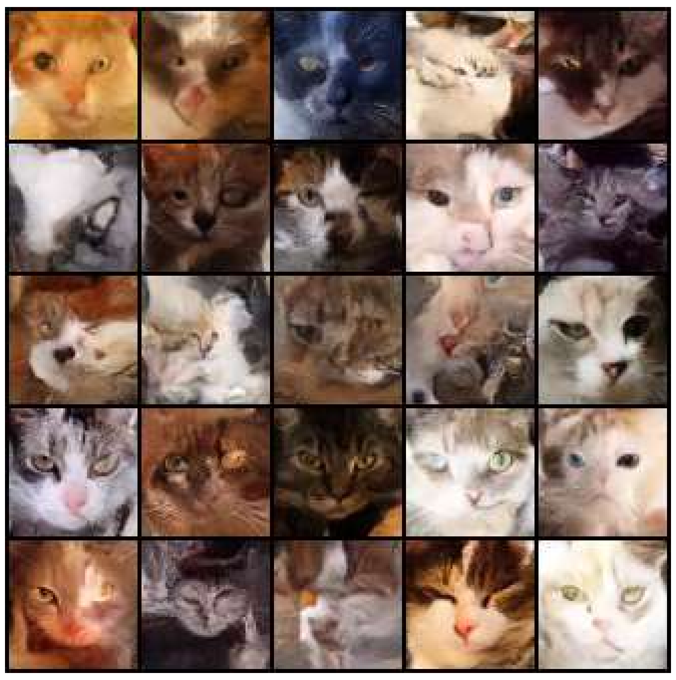
```
IMG_SIZE = 64
batch_size = 16
train_num_steps = 10000
lr = 7e-5
grad_steps = 2
ema_decay = 0.999
channels = 128
dim_mults = (1, 2, 4, 8, 16)
timesteps = 1500
beta_schedule = 'linear'
```

The main improvement includes increasing the channels and dim_mults, which enables the model to catch more details from the image, at the cost of more training and inferencing time.

Besides, learning rate is decreased and grad_steps is set to 2, which can equally increase the batch size for updating and also make the training process more delicate.

Also, more timesteps can add more noise and make the model generate better.

The sample of random generation with diffusion(1000 cat faces):



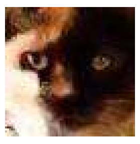
The FID score is **82.19857685440738**

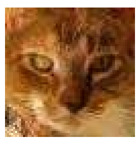
Task 2

Fusion generation with diffusion(50 pairs):

The parameter lambda is set to **0.5**, trying to get the exact average feature of the two cat faces.

Here are some quite satisfying results:





The FID score is **164.6714116636661**

Task 3

```python
def model_predictions(self, x, t, clip_x_start = False, rederive_pred_noise = False):
    """
    This function generates predicted noise and the starting image sample (x_start) based on the
    current input x and time step t. It first predicts the noise (pred_noise) through the model
    and then estimates the original image (x_start) from the predicted noise.

    Process Overview:
    1. Use the model to generate the predicted noise (`pred_noise`).
    2. Estimate the starting image (`x_start`) based on the predicted noise using `predict_start_from_noise`.

    Arguments:
    - x: The current input image.
    - t: The current time step, indicating how many diffusion steps have been applied to the image.
    - clip_x_start (bool): If True, clips the `x_start` values to be within the range [-1, 1].
    - rederive_pred_noise (bool): If True and `clip_x_start` is also True, it rederives the predicted noise
    from the clamped `x_start` values.

    Returns:
    - pred_noise: The noise predicted by the model, i.e., the noise estimated from the input x and time step t.
    - x_start: The original image sample (x_start) predicted from the noise, used for further denoising steps.
    """


def p_sample(self, x, t: int):
    """
    This function performs a single step of sampling in the reverse diffusion process. It generates
    a denoised image prediction from the noisy input `x` at a given time step `t`.
    It first computes the mean and variance of the model's prediction at the current time step using
    the `p_mean_variance` function. Then, it adds random noise to the model's predicted mean
    based on the current variance to obtain the denoised image.

    Process Overview:
    1. The function first computes the model's predicted mean and log variance at the given time
    step `t` using the `p_mean_variance` method.
    2. The denoised image (`pred_img`) is generated by adding the scaled noise to the predicted mean,
    based on the model's predicted variance.

    Arguments:
    - x: The current noisy input image at time step `t`.
    - t: The current time step in the reverse diffusion process. At `t == 0`, no noise is added.

    Returns:
    - pred_img: The denoised image predicted by the model at time step `t`. This is the output of
    the reverse diffusion process for this step.
    - x_start: The estimated original image sample (x_start), which can be used for further processing
    or reference in the model's denoising steps.
    """
```

```python
def p_sample_loop(self, shape, return_all_timesteps = False):
    """
    This function performs the full sampling loop for generating an image using the reverse diffusion process.
    It iteratively applies the `p_sample` function over all time steps, starting from random noise and progressively
    denoising it through the reverse diffusion steps.
    The loop starts with a random tensor and applies the reverse diffusion process (denoising) step by step,
    producing an image. Optionally, it can return the intermediate images at each time step.

    Process Overview:
    1. Initialize the input image as random noise of the specified shape.
    2. Iterate through the time steps in reverse order, starting from the final diffusion step (t = num_timesteps) and
    progressively applying the reverse denoising process (using `p_sample`).
    3. For each time step, `p_sample` is called to obtain a denoised image, and the intermediate images are stored.

    Arguments:
    - shape: The shape of the input image tensor.
    - return_all_timesteps (bool): If True, the function will return all intermediate images at each timestep.

    Returns:
    - ret: The generated image after the full reverse diffusion process.
    """


def p_sample_loop2(self, shape, img1, img2, lamda,return_all_timesteps = False):
    """
    This function performs the reverse diffusion sampling loop to generate an image by combining
    two input images (`img1` and `img2`) through a weighted interpolation, and then applying the
    reverse diffusion process iteratively.
    It starts by generating noisy versions of `img1` and `img2` at random time steps, and then
    interpolates between these noisy images based on the weight `lamda`. The loop then applies
    reverse denoising steps (`p_sample`) to progressively denoise the image over multiple timesteps.

    Arguments:
    - shape: The shape of the output image tensor.
    - img1: The first input image.
    - img2: The second input image.
    - lamda: A weight parameter for interpolating between `img1_start` and `img2_start`.
    A value of `0` means the output will be based entirely on `img1_start`, and a value of `1`
    means it will be based entirely on `img2_start`.
    - return_all_timesteps (bool): If True, the function will return all intermediate images generated
    at each timestep.

    Returns:
    - ret: The generated image after the reverse diffusion process.

    Process Overview:
    1. Two noisy images are generated from `img1` and `img2` at random time steps using the `q_sample` function.
    2. The two noisy images are then interpolated using `lamda` to form the initial image (`img`).
    3. The image is normalized before starting the reverse diffusion loop.
    4. The reverse diffusion process is applied iteratively using the `p_sample` function, and intermediate images
    are collected.
    5. Optionally, return all intermediate images (if `return_all_timesteps` is True), or just the final generated image.
    6. After the loop, the generated image is unnormalized (if necessary) and returned.
    """
```

```python
def q_sample(self, x_start, t, noise=None):
    """
    This function generates a noisy version of the input image (`x_start`) at a given time step `t`
    by adding noise to it. The amount of noise added depends on the model's parameters at that time step.
    It uses the cumulative product of the diffusion steps (through `sqrt_alphas_cumprod` and `sqrt_one_minus_alphas_cumprod`)
    to control the strength of the noise added based on the current time step `t`. If no custom noise is provided,
    standard Gaussian noise is generated and added to the input.

    Process Overview:
    1. If no custom noise is provided, Gaussian noise is generated with the same shape as `x_start`.
    2. The noise is scaled and added to the original image based on the model's parameters (`sqrt_alphas_cumprod`
    and `sqrt_one_minus_alphas_cumprod`) at the given time step `t`.
    3. The function returns the noisy image, which is a weighted combination of the original image and the noise.

    Arguments:
    - x_start: The original image or signal, which will be perturbed by noise.
    - t: The current time step, representing how much noise should be added to `x_start` based on the diffusion process.
    - noise: (Optional) Custom noise to add to the input. If not provided, random Gaussian noise is generated.

    Returns:
    - A noisy version of the input `x_start` at time step `t`.
    """


def p_losses(self, x_start, t, noise = None):
    """
    This function computes the loss for a single timestep in the reverse diffusion process.
    The loss is based on the difference between the predicted noise and the true noise added to the original image
    (`x_start`) during the forward diffusion process.
    The function first generates a noisy version of the input image (`x_start`) at time step `t` using the `q_sample`
    function. It then computes the model's output by passing the noisy image through the model. The loss is calculated
    by comparing the model's output to the true noise and applying a weighting factor that depends on the current time
    step (`t`).

    Process Overview:
    1. A noisy version of the original image (`x_start`) is generated using the `q_sample` function at the specified
    time step `t` by adding Gaussian noise.
    2. The noisy image is passed through the model to predict the noise.
    3. The loss between the predicted noise and the true noise is computed using the model's output.
    4. The loss is weighted by a factor (`loss_weight`) that may vary with the time step `t`.

    Arguments:
    - x_start: The original image.
    - t: The current time step at which the loss should be computed. It controls the amount of noise in the image.
    - noise: (Optional) The true noise that was added to `x_start` to obtain the noisy image. If not provided, random
    Gaussian noise is generated.

    Returns:
    - The average loss for this timestep, which measures how well the model predicted the noise added to `x_start`
    at time step `t`.
    """
```

Task 4

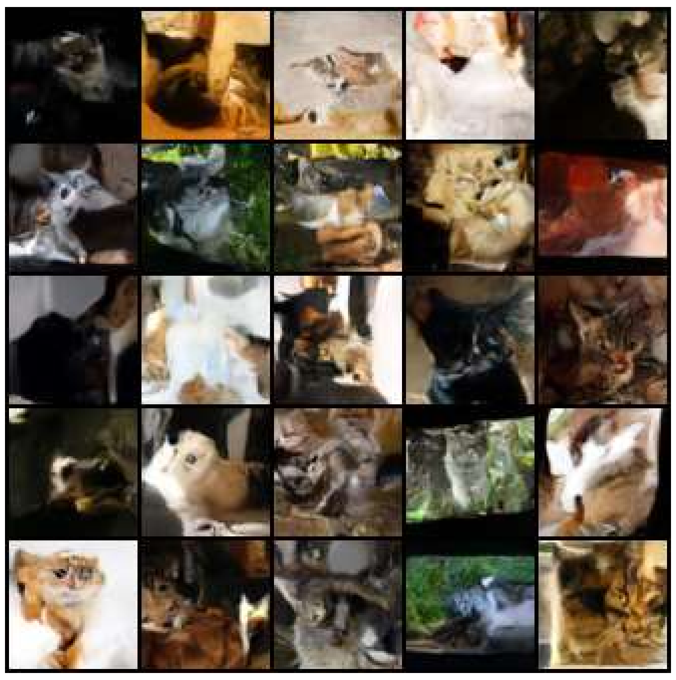
Train and test the diffusion model:

The final best parameters of my model:

```
IMG_SIZE = 64
batch_size = 16
train_num_steps = 20000
lr = 1e-4
grad_steps = 2
ema_decay = 0.999
channels = 64
dim_mults = (1, 2, 4, 8)
timesteps = 1500
beta_schedule = 'linear'
```

The IMG_SIZE is set to 64 for efficient training although nearly all images are of higher resolution.

The sample of random generation with this model(1000 cat faces):



However, using pytorch-fid requires the imagse to be of the same size, so I have to resize all the images to size 64.

```
import os
from PIL import Image
source_imgs_dir='./Data/cats_crop/'
target_imgs_dir='./Data/resized/'
count = 0
for file in os.listdir(source_imgs_dir):
    im = Image.open(source_imgs_dir + file)
    out = im.resize((64, 64), Image.LANCZOS)
    out.save(target_imgs_dir + file)
    count += 1
    if count % 500 == 0:
        print(count, 'images resized')
```

The FID score: **101.43956755943265**

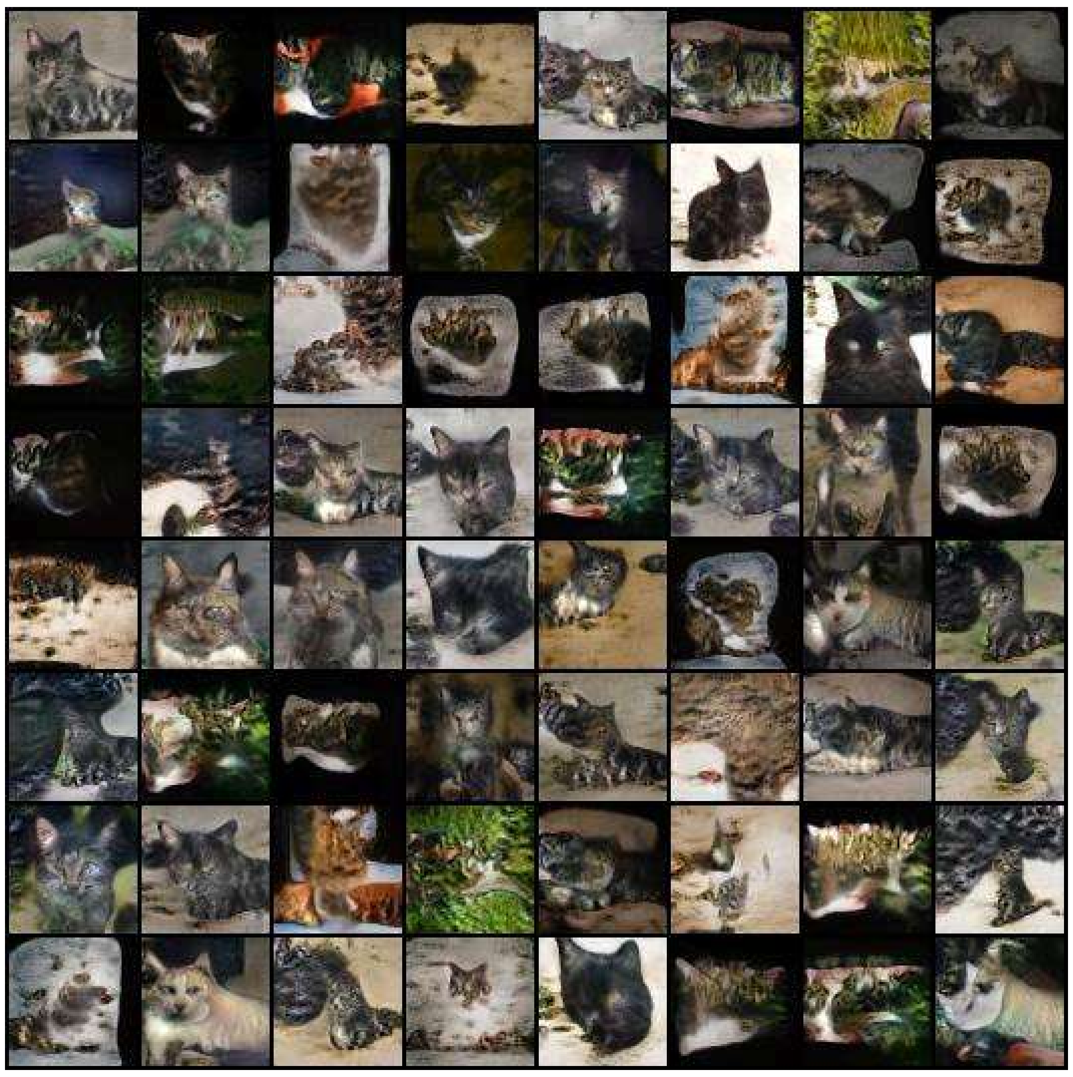
Train and test the styleGAN:

Here I use the cats_crop images that have been reshaped to square. Besides, the image-size is set to 64 to match the image size when training my own diffusion model above.

The final checkpoint is trained after 15000 steps, which takes more than three and a half hours on one 4090.

```
stylegan2_pytorch --data ./Data/cats_crop --image-size 64
```

The sample of random generation with this model(1000 cat faces):



Analyzing the differences

The generation of the sytleGAN model is much better than those of mine diffusion model.
This is because of some advanced technologies and methods that styleGAN uses when training the model:

1. Normalize the mean and variance of each feature map separately, which solves the droplet problem completely.
2. Move the operations such as appling bias and noise outside the style block.
3. Base normalization on the expected statistics of the incoming feature maps, but without explicit forcing.
4. Compute the regularization terms less frequently than the main loss function.
5. Remove progressive growing structure, but using a skip generator and a residual discriminator.

The above several methods imporve the quality of the model generation.