HW 3_Image_Generation

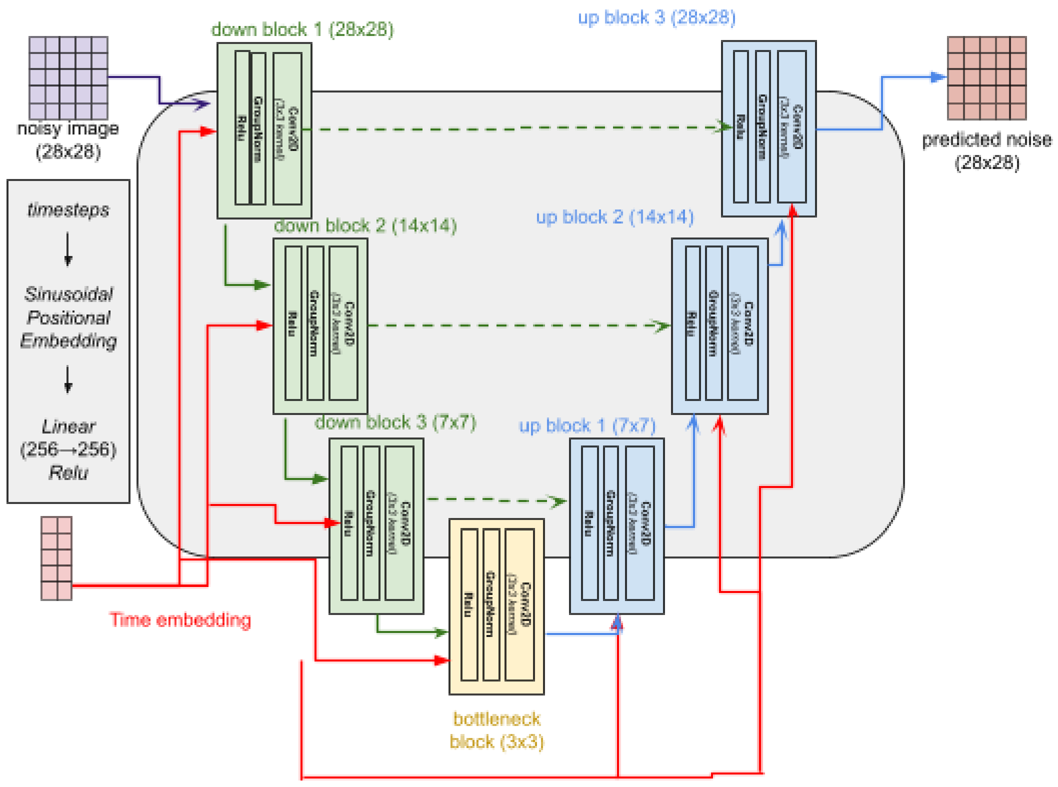
Justin Thiadi 程煒財 | NTHU_112006234

1. Model Description

a. Brief Introduction

The model follows a U-Net–based architecture conditioned on diffusion timesteps. It receives a noisy MNIST image (28×28) as input and predicts the noise that was added to the image. The network consists of three main components: the encoder (downsampling path), the bottleneck, and the decoder (upsampling path).

b. Architecture Illustration



1. Timestep Embedding (Left Block)

A scalar diffusion timestep t is converted into a 256-dimensional vector using:\

- Sinusoidal positional embeddings
- A Linear → ReLU transformation

This time embedding is then fed into every block in the U-Net. It acts as a conditioning signal that informs each layer which diffusion step the model is currently denoising.

Time embedding is not an image; it is broadcast and added to the feature maps inside each block.

2. Encoder / Down Blocks

The encoder consists of three downsampling blocks:

- Down Block 1: 28×28 | Down Block 2: 14×14 | Down Block 3: 7×7

Each block contains:

- Conv2D(3×3) | GroupNorm | ReLU | Conv2D(3×3) | GroupNorm ReLU
- and receives the time embedding internally.

After each block, max pooling halves the spatial resolution. Feature maps from each down block are saved for skip connections.

3. Bottleneck Block

At the bottom (3×3), the feature map goes through a similar residual block:

- Conv2D(3×3) | GroupNorm | ReLU | Conv2D(3×3) | GroupNorm ReLU

Also conditioned with the time embedding.
This block provides the highest-level abstract features.

4. Decoder / Up Blocks

The decoder reconstructs the 28×28 resolution via three upsampling stages:

- Up Block 1: 7×7 → 14×14
- Up Block 2: 14×14 → 28×28
- Up Block 3: final refinement at 28×28

Each stage includes:

- ConvTranspose2D (2×2) for upsampling
- Concatenation with the corresponding encoder feature map
- A block identical to the down blocks (Conv → GN → ReLU ×2)
- Time embedding added inside the block

5. Output Layer

- Finally, a 1×1 convolution maps the decoder features to the output:
    - Predicted noise image (3×28×28)
        - This noise is used in the diffusion process to reconstruct clean images.

2. Implementation Details

1. Data Preprocessing

- Dataset: MNIST training set (train=True).
- Each image is converted from 1 channel to 3 channels:
    - transforms.Grayscale(num_output_channels=3)
- Converted to tensor and normalized to [-1, 1]:
    - transforms.ToTensor(),
    - transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

Data is loaded with a DataLoader using shuffle=True, num_workers=4, pin_memory=True.

2. Data Augmentation

- No explicit data augmentation is applied (no random crops, flips, rotations, etc.).
- The model is trained directly on normalized MNIST digits.

3. Hyperparameters

- Batch size: 128
- Epochs: 50
- Learning rate: 2e-4
- Optimizer: Adam (torch.optim.Adam) with default betas.
- Diffusion timesteps: 1000
- Beta schedule: linear from 1e-4 to 2e-2.
- UNet settings:
    - in_channels = 3

- model_channels = 64
- time_emb_dim = 256
- 3 down blocks, 1 bottleneck block, 3 up blocks
- Conv kernels: 3×3 in blocks, 2×2 for upsampling, 1×1 for output.

4. Loss Function

- The model predicts the noise added to the image.
- For each training step:
  - A random timestep t is sampled uniformly in [0, T-1].
  - Gaussian noise ε is added to the clean image x_0 using the forward diffusion process q_sample.
  - The UNet takes (x_t, t) and predicts ε.

The training objective is mean squared error (MSE) between the true noise and predicted noise:

- loss = F.mse_loss(noise, predicted_noise)

5. Training Strategy

- Random timestep training: for each batch, a different random timestep t is sampled, so the model learns to denoise images at all noise levels.
- Forward diffusion (q_sample) uses precomputed ᾱ_t values to mix clean images with Gaussian noise.
- Backward pass: standard gradient descent with Adam, updating all UNet parameters.
- Sampling after training:
  - Start from pure Gaussian noise x_T ~ N(0, I).
  - Apply the learned reverse process using p_sample for all timesteps from T-1 down to 0.
  - Generated samples are rescaled from [-1, 1] back to [0, 1] and saved as PNG images for FID evaluation (10,000 samples in total).

3. Result Analysis

a. Quantitative Improvements

The model's performance was tracked throughout training using the MSE noise-prediction loss computed in:

- loss = F.mse_loss(noise, predicted_noise)

`

Across epochs 30–50, the loss consistently oscillated between 0.007 and 0.015, which indicates stable convergence of the U-Net in learning the reverse diffusion direction. This low loss means the model is accurately predicting the added Gaussian noise at random timesteps sampled using:

- t = torch.randint(0, diffusion.timesteps, (images.shape[0],))

FID Score Evaluation

After training, 10,000 synthetic MNIST images were generated using the sampling loop:

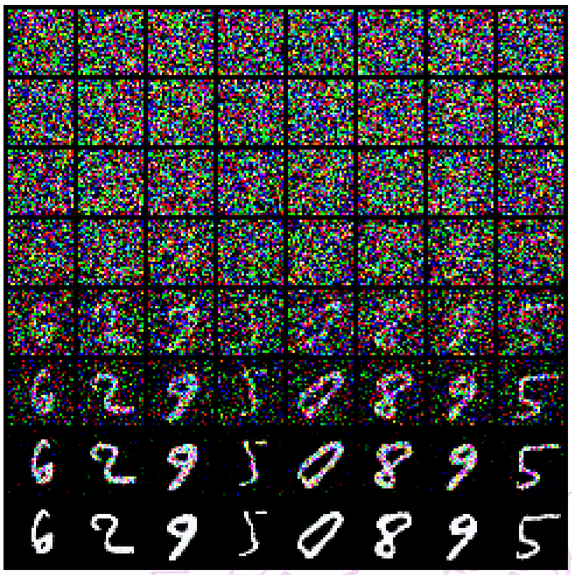
- img = diffusion.p_sample(...)

These were compared against two ground-truth datasets:

Comparison Dataset	FID
MNIST PNG images	6.12
MNIST .npz dataset	5.77

Both FID scores demonstrate good alignment between the generated and real data distributions.

b. Diffusion Process



captures 8 snapshots from random noise across 1000 denoising steps. This creates a 64-image grid where:

- Columns = different random samples
- Rows = different timesteps during reverse diffusion

The visual progression shows:

- Top rows: pure Gaussian noise
- Middle rows: blurry digit-like structures emerging
- Bottom rows: sharp, recognizable MNIST digits

This confirms the correctness of the reverse diffusion process implemented in:

- x = diffusion.p_sample(model, x, t, t_index)

Model Predictions on Images
Generated Images (10,000 samples)

Using:

- samples = diffusion.sample(model, image_size=28, batch_size=100)

the model produces diverse outputs, capturing:

- All digit categories (0–9)
- Natural variation in writing styles
- Sharper boundaries for digits
- Smooth background consistent with MNIST's style

The fact that the U-Net can transform random Gaussian noise into structured digits demonstrates that the network learned the underlying MNIST distribution effectively.

Output Quality
Because the final convolution:

self.out = nn.Conv2d(model_channels, in_channels, 1) directly maps the decoder features to RGB outputs, the generated digit shapes appear clean with limited artifacts, and the consistency is reflected quantitatively in the FID score.

4. Short Conclusion

In this homework, a diffusion-based U-Net model was implemented to generate MNIST images. The network effectively learned to predict the noise added at different timesteps, as reflected by a consistently low MSE loss (≈0.007–0.015) during later training epochs. The model was able to generate 10,000 high-quality samples with FID scores of 5.77–6.12, demonstrating strong alignment with the real MNIST distribution.

Visualization of the denoising process further confirmed that the model correctly learned the reverse diffusion from pure noise to clear digit structures. Overall, the results show that the implemented architecture and training strategy were effective for image generation on the MNIST dataset.

5. References

a. deepsense.ai. (2023). Diffusion models in practice. Part 1: a primer. https://deepsense.ai/blog/diffusion-models-in-practice-part-1-a-primers/

b. U-Net model for Denoising Diffusion Probabilistic Models (DDPM). labml.ai. https://nn.labml.ai/diffusion/ddpm/unet.html

c. Denoising Diffusion Probabilistic Models (DDPM) — LearnOpenCV Tutorial. LearnOpenCV. https://learnopencv.com/denoising-diffusion-probabilistic-models/

d. The time-conditional UNet architecture. ResearchGate. (figure illustration). https://www.researchgate.net/figure/The-time-conditional-UNet-architecture_fig5_371684920