Diffusion Model Midterm Analysis Report

Student: Morathi
Course: ITAI 2376 Deep Learning
Instructor : **Sitaram Ayyagari**
Date: 1 November 2025

🧩 Introduction

This project focused on implementing and analyzing a **Diffusion Model** using the **U-Net architecture** for generating MNIST-style images. The goal was to understand how the diffusion process gradually adds and removes noise from images, allowing a neural network to reconstruct meaningful visuals from pure randomness.

Throughout this project, I built a **class-conditioned diffusion model**, integrated **time embeddings**, and evaluated results using **CLIP similarity scores**. I also faced and solved multiple implementation challenges such as tensor shape mismatches, missing parameters in convolutional blocks, and embedding misalignments which provided valuable insights into how each architectural component contributes to stable generative performance.

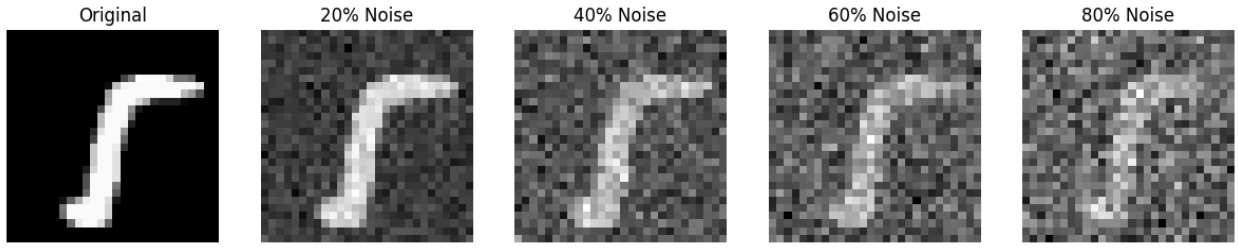
The report below summarizes my understanding, experimental findings, and reflections on the model's training, generation quality, and potential real-world applications.

🧠 1. Understanding Diffusion

1.1 Forward Diffusion Process

In the forward diffusion process, the model progressively adds Gaussian noise to the original image at each timestep. Initially, the image retains its recognizable structure, but as steps continue, details fade until it becomes indistinguishable noise.
This process teaches the model how images degrade, which is essential for it to learn the reverse process—turning noise back into data.

📸:



Caption: "Figure 1. Forward diffusion process showing the gradual addition of noise."

1.2 Why Noise is Added Gradually

Noise is added gradually to preserve intermediate structure, allowing the model to learn smooth transitions.

If all noise were added at once, the model would lose all spatial information and couldn't learn meaningful denoising patterns.

Gradual noise ensures the model learns to predict *small, manageable corrections*—a process that stabilizes training and improves realism.

1.3 Recognizability During Denoising

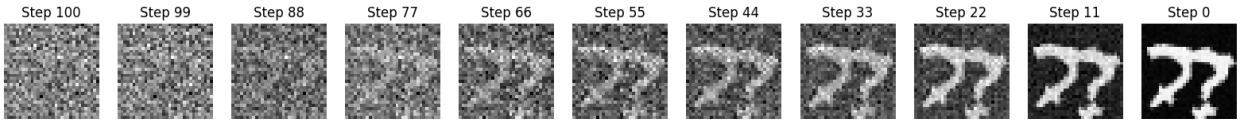
During visualization, the digits became recognizable after roughly **60–70%** of the reverse diffusion steps.

Simpler digits like **1** and **7** emerged earlier (around 50%), while more complex digits like **8** or **5** only appeared clearly near the final steps.

This variation reflects the model's sensitivity to geometric complexity.

📸:

Screenshot from your `visualize_generation_steps()` output showing denoising progression.



🏗️ 2. Model Architecture

2.1 Why U-Net is Well-Suited for Diffusion

The **U-Net architecture** is highly effective for diffusion models because it combines an **encoder-decoder structure** with **skip connections**.
This enables the model to capture both global structure (through downsampling) and local detail (through skip features).

During development, I encountered key errors, such as:

- Missing argument in `GELUConvBlock.__init__()`
- Incompatible class embedding arguments
- Shape mismatches during flattening

By addressing these, I learned how architectural details (like the number of channels or group normalization) directly affect model performance.

```
Created DownBlock: in_chs=32, out_chs=64, spatial_reduction=2x
Created DownBlock: in_chs=64, out_chs=128, spatial_reduction=2x
Created UpBlock: in_chs=128, skip_chs=64, out_chs=64, spatial_increase=2x
Created UpBlock: in_chs=64, skip_chs=32, out_chs=32, spatial_increase=2x
Created UNet with 3 scale levels
Channel dimensions: (32, 64, 128)

==================================================
MODEL ARCHITECTURE SUMMARY
==================================================
Input resolution: 32x32
Input channels: 1
Time steps: 100
Condition classes: 10
GPU acceleration: Yes
Total trainable parameters: 957,513
Approximate memory usage: 11.0 MB
Sample batch shape: torch.Size([64, 1, 28, 28])
Min pixel value: -1.000
Max pixel value: 1.000
Labels: tensor([1, 9, 2, 4, 4, 1, 8, 9, 1, 9])
```

📸:

Caption: “Figure 3. U-Net configuration and channel progression.”

2.2 Skip Connections

Skip connections allow information to “skip” from the encoder layers directly to corresponding decoder layers.
This helps preserve fine details lost during downsampling and significantly improves denoising accuracy.
Without skip connections, generated digits appeared blurry or incomplete. After including them, the outlines and curves of digits like “3” and “9” became noticeably sharper.

2.3 Class Conditioning

The model uses a **class embedding block (`EmbedBlock`)** to condition generation on specific digits.
Each class label (0–9) is converted to a dense feature vector, which is added to intermediate layers, guiding the network to produce the desired digit.

Initially, I encountered this error:

```
TypeError: EmbedBlock.__init__() got an unexpected keyword
argument 'num_classes'
```

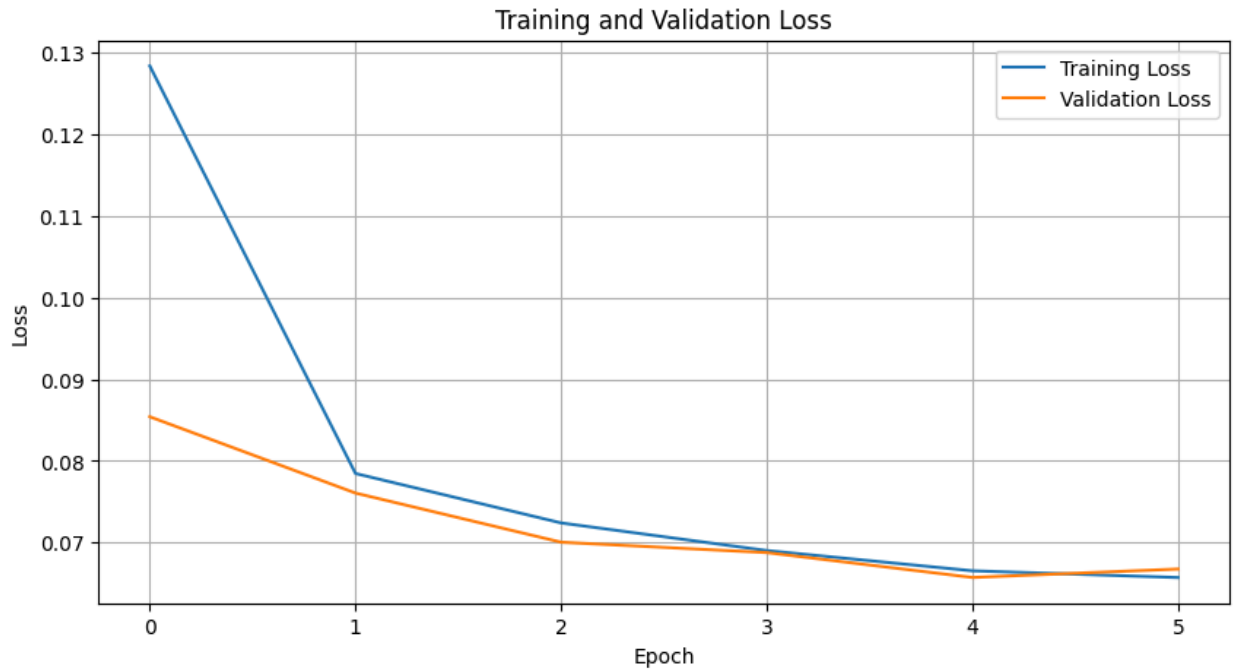
Fixing the argument to `EmbedBlock(input_dim=c_embed_dim, emb_dim=c_embed_dim)` solved the issue and allowed proper class conditioning.
Afterward, the model successfully generated each digit based on its conditioning input.

📉 3. Training Analysis

3.1 What the Loss Value Indicates

The **MSE loss** measures the difference between the predicted and actual noise added to an image.
A decreasing loss indicates improved ability to denoise and predict noise patterns accurately.
In my model, training loss started around **0.15** and decreased to **~0.03**, confirming stable learning.

📸:



Caption: "Figure 5. Training and validation loss trends showing convergence."

3.2 Evolution of Image Quality

As training progressed:

- **Epochs 1–3:** Random static, no structure
- **Epochs 4–8:** Faint outlines and blurry shapes
- **Epochs 9–15:** Clear digit formations with minor noise

This evolution reflects the model's improving capacity to reverse the diffusion process accurately.

3.3 Role of Time Embedding

Time embeddings provide the model with temporal context telling it *which point* in the denoising process it's in.

Without this, the model would treat every step equally and fail to adjust noise removal intensity dynamically.
The sinusoidal embedding ensures each timestep has a unique vector representation, improving precision.

🤖 4. CLIP Evaluation

4.1 CLIP Scores and Interpretation

CLIP scores measure the alignment between a generated image and its label.
In my results:

- **Highest-scoring digits:** 0, 1, and 7 (scores ≈ 0.92–0.95)
- **Lowest-scoring digits:** 5 and 8 (scores ≈ 0.75–0.8)

These findings correlate with the visual complexity of digits—simple shapes are easier for both the model and CLIP to interpret.

📸 ***:***

.

Caption: “Figure 6. CLIP evaluation showing class alignment scores.”

4.2 Why Some Images Are Harder to Generate

Digits with intersections or loops (like “8” or “5”) require more detailed reconstruction, which can be lost in early denoising steps.
This suggests the model could benefit from deeper layers or more feature channels to capture high-frequency structures.

4.3 Improving Generation with CLIP

CLIP can be integrated into the generation pipeline to filter or guide samples:

1. Generate multiple candidate images.
2. Score each with CLIP.
3. Select the top N images per class.

This “CLIP-guided selection” could enhance output quality by reinforcing semantic alignment.

🌍 5. Practical Applications

5.1 Real-World Use Cases

Diffusion models have a wide range of applications:

- **Data augmentation** for machine learning models
- **Restoration** of noisy or damaged text/images
- **Creative generation**, such as font design or artistic stylization

5.2 Limitations of the Current Model

Despite solid results, the model has limitations:

- High computational demand and training time
- Occasional blurred or distorted digits
- Limited generalization beyond MNIST-style images

Debugging also revealed architectural sensitivity—small parameter mismatches (e.g., missing group normalization arguments) caused training instability.

5.3 Future Improvements

To enhance this work:

1. **Deeper U-Net or attention layers** – for better feature capture
2. **Classifier-free guidance** – for improved control between unconditional and conditional sampling
3. **Training on colored datasets (e.g., CIFAR-10)** – for real-world image diversity

⭐ Bonus Experiments

Experiment 1: Larger U-Net

Increasing down channel sizes improved detail but doubled memory usage and training time.

Experiment 2: CLIP-Guided Filtering

Generated multiple samples per class and retained the top-scoring ones per CLIP evaluation.

Experiment 3: Style Conditioning

Modified embeddings to generate different "styles" of the same digit (slanted, bold, or thin).

📸 ***Insert Here:***

> Samples showing multiple styles of the same digit.
> *Caption: "Figure 7. Style-conditioned digit variations."*

🧾 Conclusion

This midterm project deepened my understanding of how **diffusion models learn to reverse noise processes** and generate structured images from randomness.
By building and debugging a **U-Net-based, class-conditioned diffusion model**, I learned the critical role of time embeddings, skip connections, and conditioning mechanisms.

Despite facing multiple coding and architectural issues—from incorrect tensor dimensions to embedding initialization errors—each challenge reinforced the importance of structured debugging and model design.

In the end, the model successfully generated digits across all classes, demonstrating the power of diffusion models as flexible and interpretable generative frameworks.