

# Deep Learning for Computer Vision HW2 Report

## *Problem 1 – Conditional Diffusion Models*

1. Describe your implementation details and the difficulties you encountered.

The model learns to predict the true noise  $\epsilon$  at any diffusion step  $t$ , conditioned on both the **digit label (0–9)** and the **domain type** (MNIST-M or SVHN).

- **Architecture:**

The network follows a small U-Net structure with two down-sampling and two up-sampling stages connected by skip links.

Each residual block receives a concatenated conditional vector composed of

- sinusoidal time embedding (128 dims),
- digit embedding (64 dims), and
- domain embedding (16 dims).

A mid-level self-attention block enhances global consistency.

The final output predicts the 3-channel RGB noise  $\epsilon$ .

- **Condition Encoding and Training Setup:**

The model jointly learns digit and domain conditions using **classifier-free guidance**. During training, 15 % of samples randomly drop their condition to encourage both conditional and unconditional generation.

We use a **cosine  $\beta$  schedule** with  $T = 1000$  diffusion steps, and train with the standard **MSE( $\hat{\epsilon}, \epsilon$ )** objective.

The optimizer is **AdamW** ( $\text{lr} = 1\text{e-}4$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ).

Training is stabilized by **EMA (Exponential Moving Average)** weight updates and **mixed-precision (AMP)** training.

Batch size = 128, trained for 50 epochs.

- **Data Processing and Conditional Mapping:**

MNIST-M and SVHN images are converted to RGB, resized to  $28 \times 28$ , and normalized to  $[-1, 1]$ .

Even digits are assigned to the **MNIST-M** domain and odd digits to the **SVHN** domain, enabling cross-domain conditional generation within a single model.

- **Sampling and Reverse Process Visualization:**

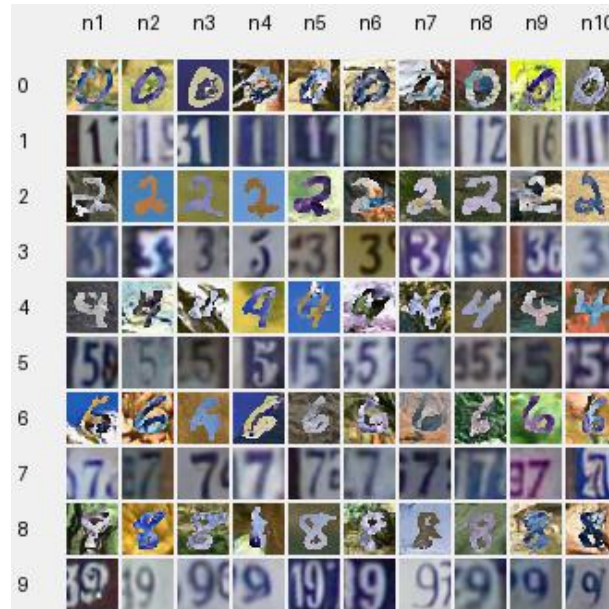
Both **DDPM** and **DDIM ( $\eta = 0$ )** sampling are implemented for inference.

For each digit 0–9, 50 images are generated with classifier-free guidance weight = 2.0.

A tracing mode saves six intermediate denoising steps ( $t = 1000 \rightarrow 0$ ) to visualize the reverse diffusion process for selected digits

- Please show 10 generated images for each digit (even digits for Mnist-M, odd digits for SVHN) in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits

**Figure 1. Ten generated samples for each digit (0–9).** Each row corresponds to a digit class from 0 to 9, and each column (n1–n10) corresponds to a different random noise input used as the starting point of the reverse diffusion process. These random noises determine the variations in style, texture, and color among samples of the same digit.

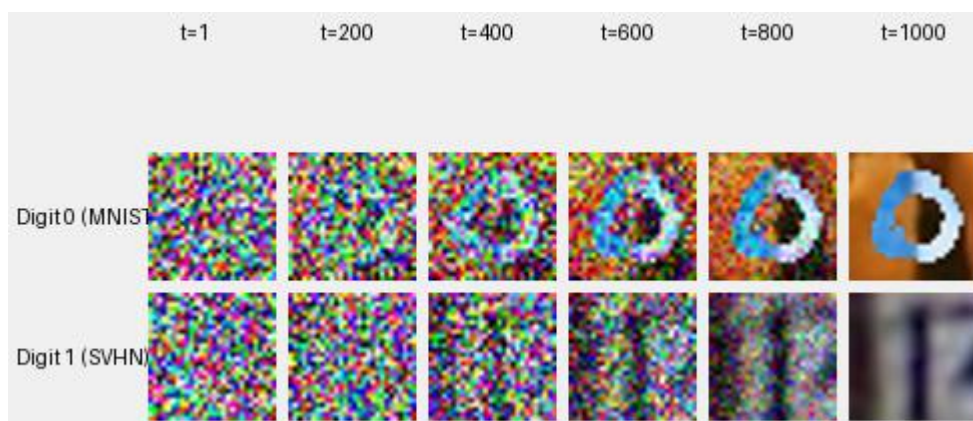


**Figure 1.** Ten generated samples for each digit (0–9).

To quantitatively evaluate generation quality, a pre-trained digit classifier was used to predict the generated results. The MNIST-M subset achieved accuracy = 0.920 (230 / 250) while the SVHN subset achieved accuracy = 0.988 (247 / 250). Overall accuracy was 0.954, indicating that the conditional diffusion model successfully learned the mapping between digit labels and domain styles.

- Visualize a total of six images in the reverse process of the first “0” and first “1” in your outputs in (2) and with different time steps.

**Figure 2.** Reverse diffusion visualization for digit 0 (MNIST-M) and digit 1 (SVHN). Each row shows a digit class and each column a timestep ( $t = 1 \rightarrow 1000$ ).



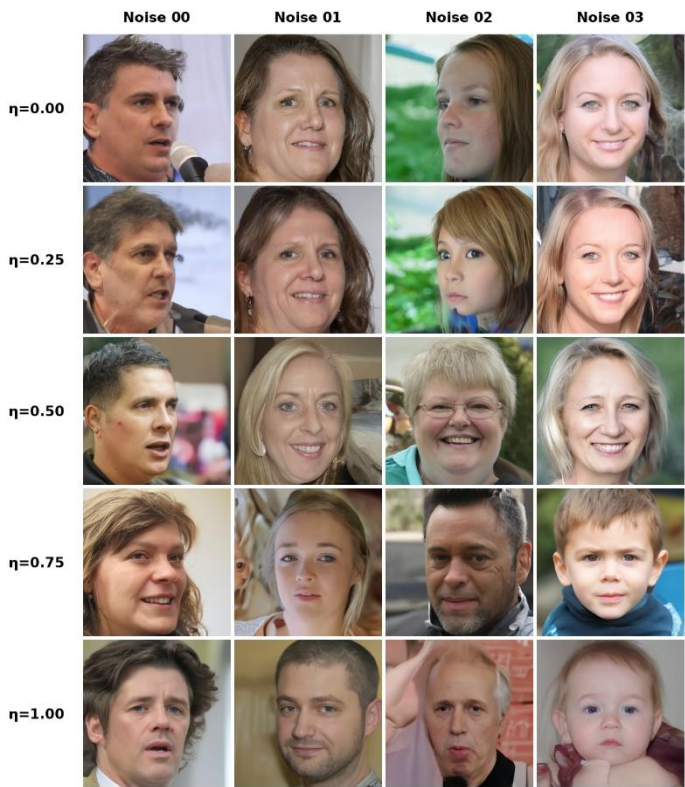
The model gradually removes Gaussian noise, revealing clear digit shapes as  $t$  increases. MNIST-M digits become visible around  $t \approx 600$ , while SVHN digits emerge later with realistic backgrounds, demonstrating correct denoising behavior.

**Figure 2.** Reverse diffusion visualization for digit 0 (MNIST-M) and digit 1 (SVHN).  
*Problem 2 – DDIM*

In this experiment, I implemented the DDIM (Denoising Diffusion Implicit Model) sampling process to generate face images . The program uses a pretrained UNet model as the denoiser and applies a deterministic DDIM sampling procedure with  $\eta = 0$ . During inference, 1000 diffusion steps are defined with a linear beta schedule (linear\_start = 1e-4, linear\_end = 2e-2), and a subset of 50 timesteps is selected uniformly for efficiency. Each sampled image is iteratively denoised from the initial Gaussian noise using the UNet-predicted noise term.After the final denoising step, a post-scaling normalization is applied to linearly rescale pixel values back into the range  $[-1, 1]$ , ensuring consistent brightness and contrast among generated outputs.

1.Please generate face images of noise 00.pt ~ 03.pt with different eta in one grid. Report and explain your observation in this experiment.

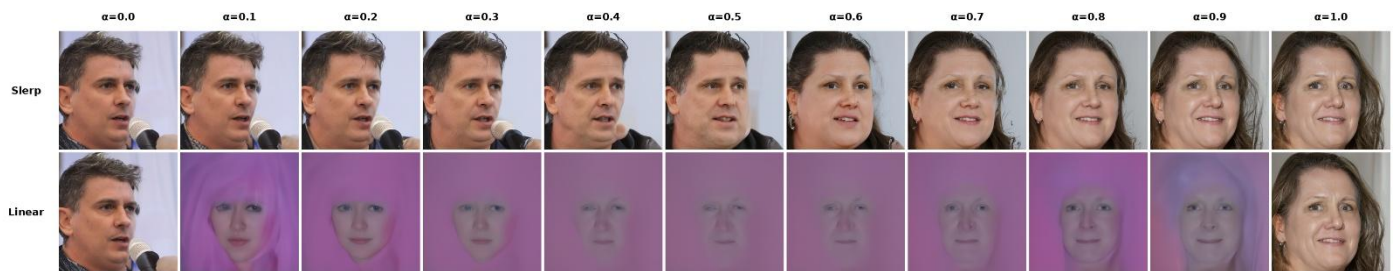
As shown in **Figure 3**, when  $\eta = 0.00$ , the generated images are the most stable and clear, with natural facial details and well-defined structures. As  $\eta$  gradually increases (e.g., 0.25–0.50), slight variations in lighting and expression appear, indicating enhanced diversity in the generated results. When  $\eta = 0.75$  and 1.00, the amount of noise increases noticeably, and some samples become blurry or distorted.



**Figure 3.** DDIM-generated faces with varying  $\eta$  values (rows) and noise inputs (columns).

2. Please generate the face images of the interpolation of noise 00.pt ~ 01.pt. The interpolation formula is spherical linear interpolation, which is also known as slerp.

From Figure 4, it can be observed that when using linear interpolation, the generated images in the middle range ( $\alpha \approx 0.3\text{--}0.8$ ) become blurry and color-distorted, and some faces appear deformed or unrealistic. This happens because linear interpolation changes the magnitude of the noise vectors, causing them to deviate from the Gaussian sphere distribution assumed during training, which leads to invalid intermediate samples. In contrast, spherical linear interpolation (Slerp) preserves the vector norm, keeping the interpolated points on the Gaussian sphere, thus producing smooth and natural transitions between two faces with consistent structure and realistic appearance.



**Figure 4.** DDIM-generated faces with varying  $\eta$  values (rows) and noise inputs (columns).

### *Problem 3 – ControlNet*

#### 1. Describe your implementation details and the difficulties you encountered.

This task was implemented based on the Stable Diffusion architecture, integrated with a customized ControlNet to achieve conditional image generation control. The overall pipeline consisted of a training phase and an inference phase. During training, the main components of Stable Diffusion — including the VAE encoder and UNet — were completely frozen, while only the ControlNet was optimized. This design ensured that the pretrained semantic capability of Stable Diffusion was preserved, allowing the ControlNet to learn structural alignment from the input control (hint) images. To enable multi-scale conditioning, forward hooks were registered to all twelve encoder blocks and the middle block of the UNet. These hooks injected the control features generated by the ControlNet into the corresponding UNet layers during the forward pass. Before injection, each control feature map was dynamically resized to match the spatial dimensions of the UNet feature maps and channel-aligned or padded when necessary. The training objective was the mean squared error between the predicted and ground-truth noise, with gradient clipping applied at each update step to maintain stability.

Several technical challenges were encountered during implementation. Initially, a “tensor does not require grad” error occurred because the hooks did not properly preserve gradient flow through the injected layers. This was resolved by redesigning the forward hooks to remain differentiable rather than detaching tensors. Another difficulty arose from the extremely small or unstable loss values — the Fill50k dataset was relatively simple, causing the model to converge too quickly ( $\text{loss} < 0.01$ ). To mitigate this, the learning rate and control strength parameters were tuned, and gradient clipping was applied to ensure consistent updates. A further issue came from mismatched feature dimensions between ControlNet and the UNet, as their spatial resolutions and channel counts differed across scales. This was handled by dynamically resizing and zero-padding the control feature maps before merging them into the UNet. Additionally, the gradient checkpointing mechanism in Stable Diffusion caused backpropagation conflicts when the UNet was frozen. This was resolved by explicitly disabling all gradient checkpointing in both the Stable Diffusion and PyTorch modules.

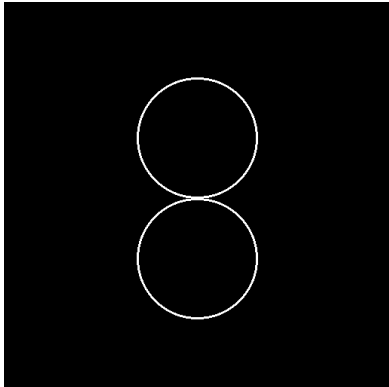
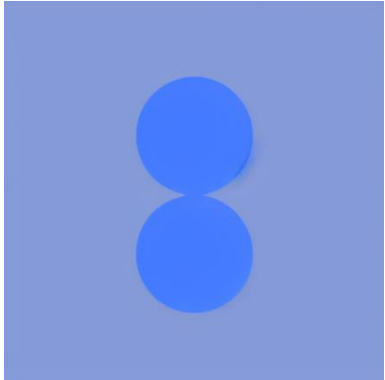
Regarding image resolution, early experiments used  $256 \times 256$  inputs, but the results showed that the ControlNet and Stable Diffusion decoder required consistent spatial resolution to align properly. Therefore, both training and inference were standardized to  $512 \times 512$  resolution.



2. Generate images using ControlNet with a control image containing two circles  
 You may freely define the color prompts for the circles.  
 You may also freely define the positions and sizes of the circles in the control image.  
 Provide two sets of generated results in the report and discuss whether ControlNet can be effectively applied under these conditions. If it fails or shows weak control, briefly explain the possible reasons.

Based on the table below, ControlNet demonstrates a reasonable ability to follow structural guidance from the control images. In the first case (*“red circle and blue circle on white background”*), the model successfully generates two circular shapes positioned vertically, consistent with the control image layout. However, the model fails to accurately reproduce the intended red and blue colors, resulting instead in blue-toned objects with limited color variation. In the second case (*“large yellow circle and small green circle on purple background”*), the generated output correctly reflects the relative size and spatial arrangement of the circles, but again shows noticeable color deviation and blending artifacts.

Overall, ControlNet performs well in capturing geometry and position, but shows weak color control under these conditions. The failure is likely caused by the limited training data diversity, insufficient fine-tuning on color-specific prompts, and potential imbalance between the text and control signals during inference. Increasing control strength or improving prompt-conditioning consistency could enhance the accuracy of color adherence in future runs.

Prompt	Control image	Target image
red circle and blue circle on white background		
large yellow circle and small green circle on purple background	