DLCV HW2

NTU, FALL 2024

B10901151 林祐群
指導教授： 王鈺強
2024年10月28日

Problem 1: Diffusion Models

1. (5%) Describe your implementation details and the difficulties you encountered.

The implementation is a Denoising Diffusion Probabilistic Model (DDPM) for generating digit images across two datasets (MNIST-M and SVHN).

i. Architecture Components:

class ContextUnet(nn.Module):

A U-Net architecture with context embedding for conditional generation. I use ConvBlock modules with residual connections and implement time embedding and context embedding for conditioning. I also considered using GroupNorm and SiLU activations for better training stability

ii. DDPM Implementation:

class DDPM(nn.Module):

I use a linear noise schedule between β_1 and β_2 and implement forward diffusion process and reverse denoising. It supports classifier-free guidance for better conditional generation

iii. Dataset Handling:

class DigitDataset(Dataset):

A custom dataset class handling two different digit datasets (MNIST-M and SVHN), which combines both datasets with dataset-specific labels. I use a weighted sampler for balanced training with normalization and resizing.

iv. Training Process:

def train_model(model, train_loader, num_epochs=100, lr=1e-4):

I use AdamW optimizer with cosine learning rate scheduling and implement mixed precision training using torch.cuda.amp for faster training to settle down the problem of limited computing resource.

v. Sampling Process:

def sample_images(model, num_samples=50, guidance_scale=1.5, device='cuda'):

I implement classifier-free guidance for conditional generation with a guidance scale of 1.5 for better quality, including denormalization and proper image saving.

vi. Conditioning Strategy:

I use combined labels: digit_label + dataset_label * 10 and implement context masking for classifier-free guidance with the embeddings of both time steps and class conditions.

Difficulties I encountered are mainly stemming from the denoising process where improper implementation of just a few lines of code will lead to significant deviation from the ground truth. That is to say, I spent most of the time finding out the wrong implementation throughout my code for the problem.

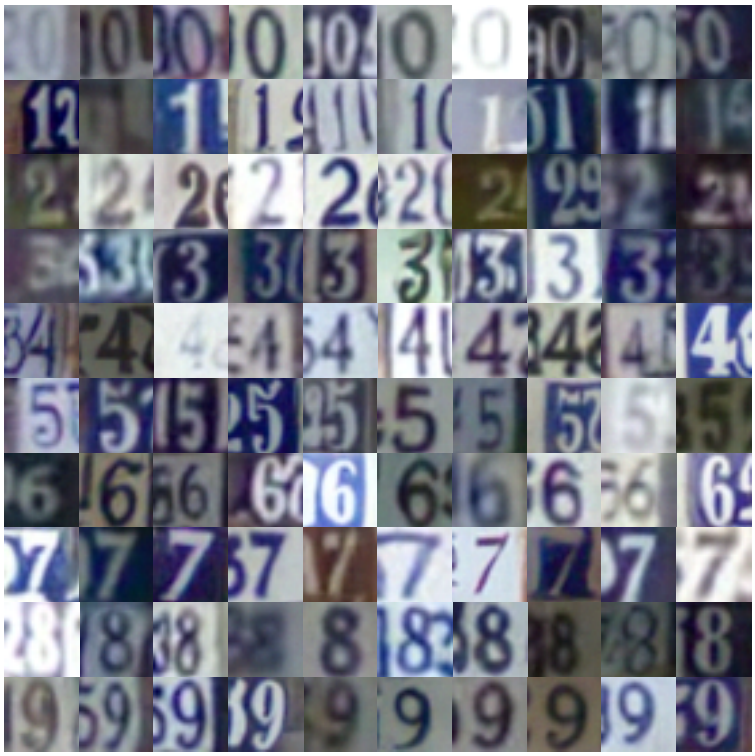
Also, the conditional strategy is not trivially implemented since lots of errors I encountered during the process of embeddings these conditions into the model. I've also considered using one-hot encodings, but failed to properly integrated into the code.

Lastly, the resizing and normalization of images and possible noise is of importance that I was not aware of at first and lead to poor results. This is quite crucial to generate a visually-decent picture, and should be consciously monitored through out the code to maintain consistency for proper implementation.

2. (5%) Please show 10 generated images **for each digit (0-9) from both MNIST-M & SVHN dataset** in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits. [see the below MNIST-M example, you should visualize **BOTH** MNIST-M & SVHN]

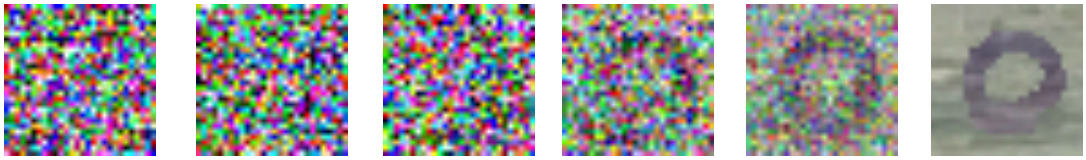
1. MNIST-M:

2. SVHN:



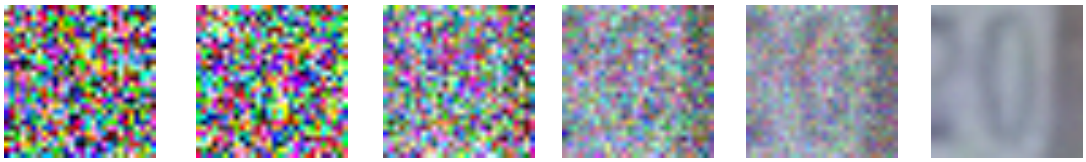
3. (5%) Visualize a total of six images from **both MNIST-M & SVHN datasets** in the reverse process of the **first "0"** in your outputs in (2) and with **different time steps**. [see the MNIST-M example below, but you need to visualize **BOTH** MNIST-M & SVHN]

1. MNIST-M:



t = 0 t = 200 t = 500 t = 800 t = 900 t = 1000

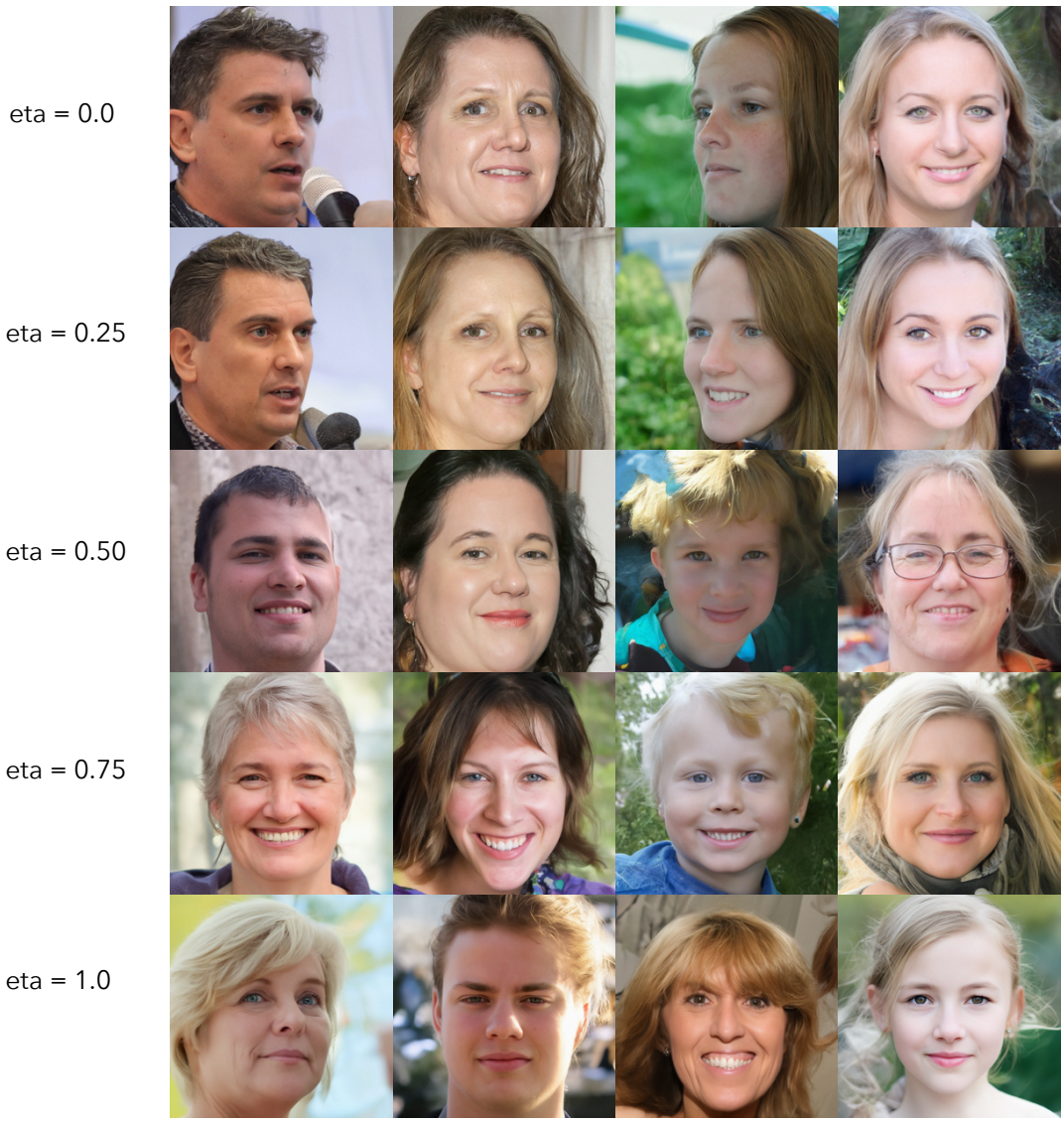
2. SVHN:



t = 0 t = 200 t = 500 t = 800 t = 900 t = 1000

Problem 2: DDIM

1. (7.5%) Please generate face images of noise **00.pt ~ 03.pt with different eta** in one grid. Report and explain your observation in this experiment.



Input Noise Files:

The experiment uses four noise files: `noise_00.pt`, `noise_01.pt`, `noise_02.pt`, and `noise_03.pt`, containing randomly generated noise sampled from the latent space, which serves as the starting point for the image generation process.

Different Eta Values:

Various eta values are set for the experiment, 0.0, 0.25, 0.5, 0.75 and 1.0. The eta parameter controls the level of noise added during the generation process, affecting how closely the generated images resemble the original samples.

Observation:

I. Visual Quality:

As eta increases, the generated images tend to have more diversity and variation in texture and detail. Lower eta values (e.g., 0.0) produce images that are closer to the original noise, resulting in smoother and potentially more coherent outputs.

II. Noise Effects:

When using higher eta values, the images may exhibit more pronounced noise patterns, leading to less recognizable or sharper features. This suggests that the balance between the original noise and the introduced randomness can significantly impact the quality of the generated images.

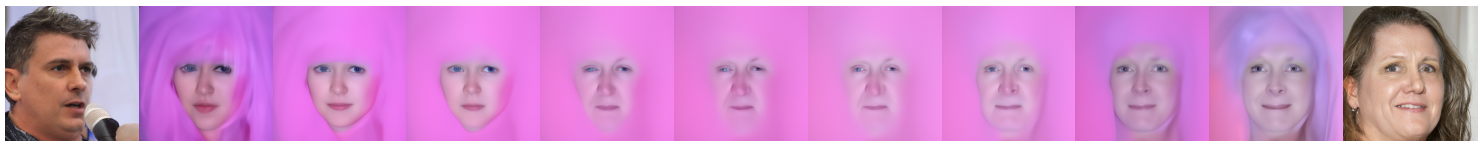
2. (7.5%) Please generate the face images of the interpolation of noise **00.pt ~ 01.pt.** The interpolation formula is spherical linear interpolation, which is also known as slerp.

in this case, $\alpha = \{0.0, 0.1, 0.2, \ldots, 1.0\}$.

What will happen if we simply use linear interpolation? Explain and report your observation. (There should be **two images** in your report, one for spherical linear and the other for linear)

I. Spherical Linear Interpolation:

II. Linear Interpolation:



Spherical Linear Interpolation (SLERP):

The images generated using slerp show a smooth transition between the two noise vectors, capturing the nuanced changes in features and details. The progression appears natural, with the images evolving in a coherent manner as α increases. This results in outputs that maintain high visual quality and structural integrity.

Linear Interpolation (LERP):

The images generated using linear interpolation exhibit more abrupt transitions and can appear less coherent. The changes in features and details may seem disjointed or unnatural. Linear interpolation implemented in this experiment fails to capture the true geometry of the latent space, leading to

images that look distorted and less realistic, especially at intermediate values of α, while only images at α equals to 0 or 1 can the images be visually recognizable.

Problem 3: Personalization

1. (7.5%) Conduct the CLIP-based zero shot classification on the hw2_data/clip_zeroshot/val, explain how CLIP do this, report the accuracy and 5 successful/failed cases.

- Use the prompt "A photo of {object}."
- The naming rules of Images are {class_id}_{image_id}.png.
- The id-to-label mapping is provided in hw2_data/clip_zeroshot/id2label.json
- You should be able to import clip after installing the package from environment.yaml.(You can follow the sample code in CLIP repo.)

Zero-shot classification overall accuracy: 56.64%

5 Successful Cases:

File: 5_495.png, True: clock, Predicted: clock

File: 8_470.png, True: porcupine, Predicted: porcupine

File: 33_488.png, True: mouse, Predicted: mouse

File: 21_483.png, True: wardrobe, Predicted: wardrobe

File: 32_495.png, True: bus, Predicted: bus

5 Failed Cases:

File: 24_492.png, True: house, Predicted: palm_tree

File: 34_461.png, True: castle, Predicted: willow_tree

File: 24_475.png, True: house, Predicted: willow_tree

File: 4_455.png, True: lamp, Predicted: willow_tree

File: 33_495.png, True: mouse, Predicted: willow_tree

How CLIP Works

CLIP (Contrastive Language–Image Pretraining) uses a neural network that was trained on a vast dataset of image-caption pairs. It can perform zero-shot

classification by comparing the similarity between image features and text features.

1. **Image Encoding**: CLIP encodes an input image into a high-dimensional feature vector.
2. **Text Encoding**: A set of candidate labels (e.g., `class_labels`) is transformed into descriptive prompts (e.g., `"A photo of {object}."`), which are then encoded as text feature vectors.
3. **Similarity Calculation**: The cosine similarity between the encoded image features and the encoded text features is calculated.
4. **Prediction**: The label with the highest similarity score is chosen as the predicted class.

2. (7.5%) What will happen if you simply generate an image containing multiple concepts (e.g., a <new1> next to a <new2>)? You can use your own objects or the provided cat images in the dataset. Share your findings and survey a related paper that works on multiple concepts personalization, and share their method.

I. Generating an image containing multiple concepts (like a `<new1>` next to a `<new2>`) using a model trained on textual inversion may yield:

1. The generated image will combine the characteristics of both concepts, potentially leading to creative or unexpected interactions between them.

2. If one concept is underrepresented or poorly learned, it might dominate the image or be rendered inaccurately.

3. There could be challenges with spatial relationships (e.g., positioning of the two concepts) or ensuring that both concepts are distinct rather than merging into a single, unclear representation.

That is, we need to consider Concept Prioritization and Spatial Relationship.

II. Paper: "Concept Weaver: Enabling Multi-Concept Fusion in Text-to-Image Models":

It introduces an innovative framework for generating images that effectively combine multiple personalized concepts using text-to-image models.

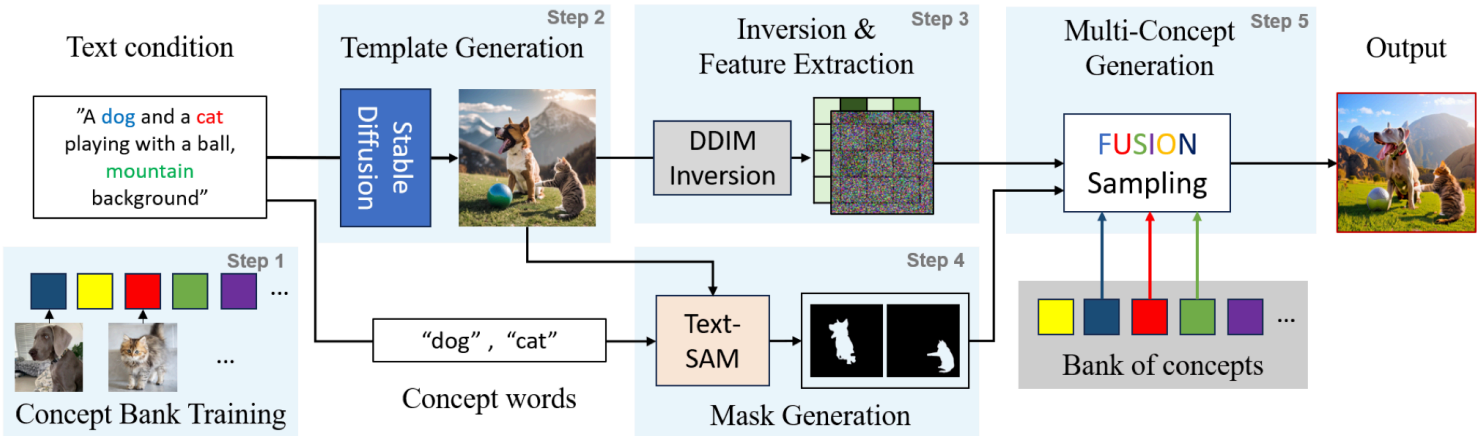
The key processes can be illustrated as the follows:

1. Feature Extraction: Concept Weaver captures U-Net model features through an inversion process. By utilizing latent features from different layers, such as residual layers and self-attention maps, the model retains structural information for each personalized concept. This structural information helps ensure each concept is distinctly represented in the final image.

2. Mask Generation and Concept-Aware Masking: To manage overlapping regions, a segmentation model is used to create concept-wise masks. These masks are then dilated (expanded) to avoid deformations in complex images, and overlapping areas are specially managed to maintain individual concept integrity.

3. Concept-Aware Fusion: During the generation phase, the system combines features specific to each concept with a unified background, applying a distinct text prompt for each concept to prevent "concept leakage." This unique fusion technique enables clear and individualized concept representations, even in scenes with multiple elements like "a cat and a dog in a mountain background."

Concept Weaver's multi-layered fusion and masking approach ensures high fidelity and reduces the common issue of blending between distinct concepts, which is challenging when using traditional compositional methods in diffusion models



Reference: https://arxiv.org/pdf/2404.03913

References

1. Generating images with DDPMs: A PyTorch Implementation:
https://medium.com/@brianpulfer/enerating-images-with-ddpms-a-pytorch-implementation-cef5a2ba8cb1
2. Dataset resize issue:
https://stackoverflow.com/questions/52463018/mnist-and-svhn-resize
3. DDIM:
https://arxiv.org/pdf/2010.02502
4. DDIM Sampling:
https://nn.labml.ai/diffusion/stable_diffusion/sampler/ddim.html
5. Slerp:
https://medium.com/akatsuki-taiwan-technology/spherical-linear-interpolation-38df7a7a0d38
6. Clip-repo:
https://github.com/openai/CLIP
7. Textual inversion repo:
https://github.com/rinongal/textual_inversion
8. Concept Weaver:
https://arxiv.org/pdf/2404.03913
9. ChatGPT
10. Claude
11. Gemeni