Unconditional Image Generation on Latent Space

Part I: Training Unconditional Latent Diffusion Model on Face Dataset

In this part, I transformed an unconditional image generation model that originally worked in pixel space to one that functions in latent space. The primary goal was to reduce computational costs while maintaining or improving the quality of generated images.

The initial model was designed to generate and predict pixel values directly, which can be computationally expensive, especially for high-resolution images. To eliminate this, I modified the original script to work in latent space by integrating a Variational Autoencoder (VAE).

I started with the Pokemon and Flower datasets, which are better starting point since their size are smaller. Also, it is simpler to train a model to generate pokemon and flower than generate human faces since they are easier tasks. After ensuring the script was functioning correctly, I moved on to the FFHQ256 dataset.

	CelebA-HQ 256×256	FFHQ 256×256	LSUN-Churches 256×256	LSUN-Bedrooms 256×256
f	4	4	8	4
z-shape	$64 \times 64 \times 3$	$64 \times 64 \times 3$	-	$64 \times 64 \times 3$
$\|\mathcal{Z}\|$	8192	8192	-	8192
Diffusion steps	1000	1000	1000	1000
Noise Schedule	linear	linear	linear	linear
Nparams	274M	274M	294M	274M
Channels	224	224	192	224
Depth	2	2	2	2
Channel Multiplier	1,2,3,4	1,2,3,4	1,2,2,4,4	1,2,3,4
Attention resolutions	32, 16, 8	32, 16, 8	32, 16, 8, 4	32, 16, 8
Head Channels	32	32	24	32
Batch Size	48	42	96	48
Iterations*	410k	635k	500k	1.9M
Learning Rate	9.6e-5	8.4e-5	5.e-5	9.6e-5

Hyperparameters for unconditional LDMs from CompVis Paper.

	LDM-1	*LDM-2*	*LDM-4*	*LDM-8*	*LDM-16*	*LDM-32*
z-shape	$256 \times 256 \times 3$	$128 \times 128 \times 2$	$64 \times 64 \times 3$	$32 \times 32 \times 4$	$16 \times 16 \times 8$	$88 \times 8 \times 32$
$\|\mathcal{Z}\|$	-	2048	8192	16384	16384	16384
Diffusion steps	1000	1000	1000	1000	1000	1000
Noise Schedule	linear	linear	linear	linear	linear	linear
Model Size	270M	265M	274M	258M	260M	258M
Channels	192	192	224	256	256	256
Depth	2	2	2	2	2	2
Channel Multiplier	1,1,2,2,4,4	1,2,2,4,4	1,2,3,4	1,2,4	1,2,4	1,2,4
Attention resolutions	32, 16, 8	32, 16, 8	32, 16, 8	32, 16, 8	16, 8, 4	8, 4, 2
Head Channels	32	32	32	32	32	32
Batch Size	9	11	48	96	128	128
Iterations*	500k	500k	500k	500k	500k	500k
Learning Rate	9e-5	1.1e-4	9.6e-5	9.6e-5	1.3e-4	1.3e-4

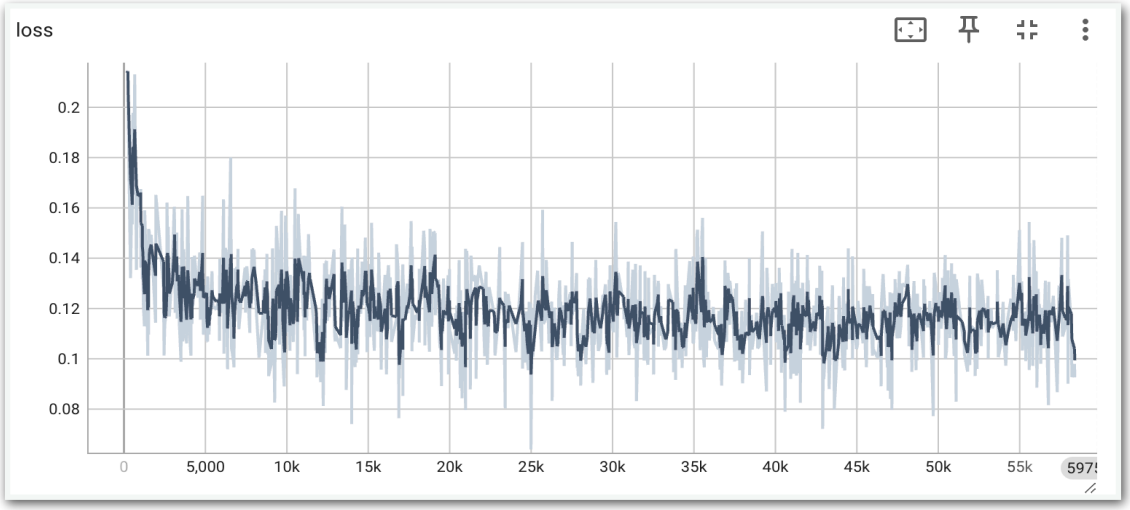
Hyperparameters for the unconditional LDM's trained on CelebA dataset

One of the most challenging aspects was adjusting the channels, pipeline script, and hyperparameters. I used diffusers repo from GitHub, HuggingFace, the original CompVis paper, and various forums to guide these adjustments. The scheduler and Unet hyperparameters were particularly tricky since they depend heavily on the input image resolution, the downsampling factor of the Autoencoder, architecture, and the complexity of the task. I obtained these setting from the tables above.

The pipeline was built using the DiffusionPipeline class from the diffusers library, with significant modifications to the output generation process.

Images generated by base unconditional model

After 70 epochs, results were mostly quite realistic and diverse. Some samples have minor artifacts, but it's important to note that the training could have been extended, and it was done within limited time and resources. All samples in these report were generated using the same seed to ensure more accurate comparison.



loss curve of unconditional image generation(ffhq-70 epochs)

After the first few thousand steps, the loss decreases to a certain level and then slows down. This type of loss curve is typically observed in while training diffusion models. Model learns early patterns easily, however, the more complex features like face details take more time to learn. Loss doesn't always directly reflect the sample quality.

Part II: Fine Tuning the Model on a Subject

I tried different people's faces as instance images. When instance images with similar backgrounds and clothes were used, model overfit these features quickly and generated faces with same background and clothes. Therefore, instance dataset should be diverse in terms of lighting, clothing, posing, angles, and background.

I used images of Heath Ledger and aligned all images with FFHQ dataset with alignment algorithm for this dataset.

Instance Images of Heath Ledger

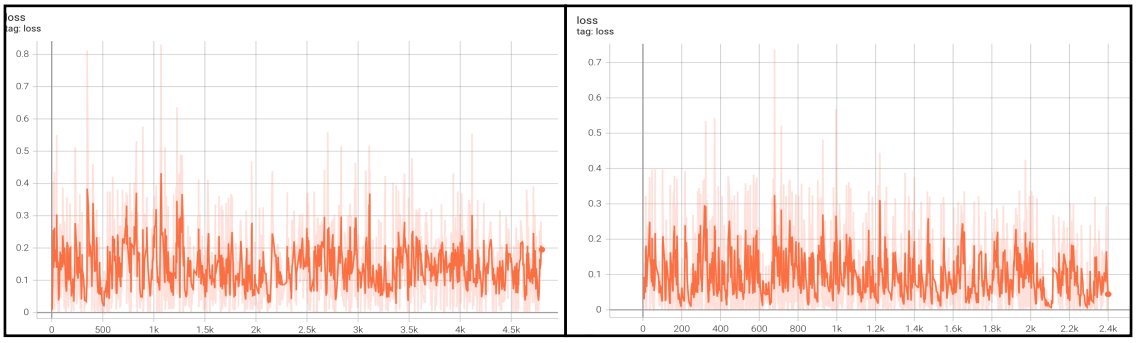
Since our fine tuning method is unconditional, our fine tuned model doesn't always generate perfect samples. Especially, when using prior preservation, the prior part of the loss can cause model to sometimes generate random images that don't look like our instances. When trying to adjust hyperparameters to make all outputs look like instances, the model may overfit easily.

There is a tradeoff between ensuring that a high percentage of the samples looks like our desired subject and maintaining diversity while preserving more learned features from the original model. Also, I tried to use only male faces as class images and it didn't give better results.

Images generated with fine tuned model(without preservation loss)

As the model is fine tuned, variation decreases and results are getting closer to the instances.As the model is fine-tuned, the variation in the generated outputs gradually reduces, leading to results that more closely resemble the specific instances provided for training. This refinement process enhances the model's ability to capture and replicate the unique characteristics of the instances, but it may also limit the diversity of the outputs.

Images generated with fine tuned model(with preservation loss)



Loss curve of Dreambooth with Prior Preservation Loss (weight=0.5, 4800 steps)

Loss curve of Dreambooth without Prior Preservation Loss (2400 steps)

Loss curves has similar shape but without prior preservation, the loss drops to lower levels at half the number of steps as expected.

Experiments:

1: First of all, I used default configurations of Unet2DModel from huggingface. The only change were size of in_channels, out_channels and sample_size. Then, saved the results and continued experimenting the configurations in the paper for a better model.

2: I used the

```
!accelerate launch train_unconditional.py \
--train_data_dir "data" \
--resolution=256 \
--output_dir="outputs" \
--train_batch_size=96 \
--eval_batch_size=8 \
--num_epochs=80 \
--gradient_accumulation_steps=1 \
--checkpointing_steps=8000 \
--use_ema \
--save_images_epochs=5 \
--save_model_epochs=5 \
--learning_rate=5e-5 \
--lr_warmup_steps=500 \
--mixed_precision="no"
```

```
if args.model_config_name_or_path is None:
    model = UNet2DModel(
        sample_size=32,
        in_channels=4,
        out_channels=4,
        layers_per_block=2,
        block_out_channels=(192,384,384,768,768),
        down_block_types=(
            "AttnDownBlock2D",
            "AttnDownBlock2D",
            "AttnDownBlock2D",
            "AttnDownBlock2D",
            "DownBlock2D",
        ),
        up_block_types=(
            "UpBlock2D",
            "AttnUpBlock2D",
            "AttnUpBlock2D",
            "AttnUpBlock2D",
            "AttnUpBlock2D",
        ),
        attention_head_dim=24,
    )
```

I tried these as it was the actual configurations in the paper for Unconditional model trained on LSUN Church dataset. However, the results were not the best.

3: Parameters & hyperparameters for final version. These settings are derived from the article.

```
!accelerate launch train_unconditional.py \
--train_data_dir "data" \
--resolution=256 \
--output_dir="outputs" \
--train_batch_size=84 \
--eval_batch_size=8 \
--num_epochs=70 \
--gradient_accumulation_steps=1 \
--checkpointing_steps=8000 \
--use_ema \
--save_images_epochs=5 \
--save_model_epochs=5 \
--learning_rate=8.4e-5 \
--lr_warmup_steps=500 \
--mixed_precision="no"
```

```
# Initialize the model
if args.model_config_name_or_path is None:
    model = UNet2DModel(
        sample_size=32,
        in_channels=4,
        out_channels=4,
        layers_per_block=2,
        block_out_channels=(256,512,1024),
        down_block_types=(
            "AttnDownBlock2D",
            "AttnDownBlock2D",
            "AttnDownBlock2D",
        ),
        up_block_types=(
            "AttnUpBlock2D",
            "AttnUpBlock2D",
            "AttnUpBlock2D",
        ),
        attention_head_dim=32,
    )
```