University of Washington

Classical and Deep Methods for Computer Vision
EE P 596

Report: Generating face images using Diffusion models

Authors: Arnab Karmakar, Stan Birchfield

December 17, 2023

Introduction

Diffusion models has been one of the most popuar algorithms for image generation in recent times. However, unconditional diffusion models do not provide precise control over the generated images. To effectively utilise a diffusion models for the desired type of output, conditioning over image attributes are required. In this project, we attempt to develop a conditional diffusion model, to utilize the influence of image attributes — ranging from skin tone to hair color, age, gender etc. — for the purpose of nuanced image generation.

The motivation behind this project is rooted in achieving precise control over the generated images, particularly in terms of specific object poses and attributes within the image. The initial scope of the project focused on developing a diffusion model for hand object interaction image generation, however, the project pivoted to implementing a baseline conditional diffusion model for facial image generation that can later be adapted to more downstream tasks such as GenAI based image editing. We first develop an unconditional diffusion model for high quality random face image generation, and make modifications to incorporate face attribute vectors to generate controlled images. We demonstrate our experiments in MNIST and CelebA dataset, and report our results.

A survey of famous diffusion models reveals that popular models such as stable diffusion, Dall-E or Midjourney conditions image generation on text prompts. However, the intricacies inherent in the context of nuanced image generation necessitate a framework that conditions not only on text but also on object attributes, thereby enhancing control over the generation process. In contrast to previous models that rely on complex architectures and supplementary attributes such as image segmentation map and pose detection, our approach aims to streamline the process, effecting one-step image generation based on the image attributes of a given facial image.

The key contribution of this work are the following:

- We develop a diffusion model from scratch, conditioned on image attributes
- We evaluate our model on MNIST and CelebA data set, demonstrating smooth interpolation through editing image attributes
- We compare the performance of the unconditional model with respect to conditional model, attaining a 4% increase in FID

Method

We implement the Denoising Diffusion Probabilistic Models (DDPM) [HJA20] paper and implement the classifier free guidance [HS22] technique to include the attribute information throughout the image generation process. The base architecture is of a U-Net that utilises ResNet blocks and self attention modules. Figure 1 depicts the basic framework of the model architecture. The details of the DDPM model is further explained in Figure 2

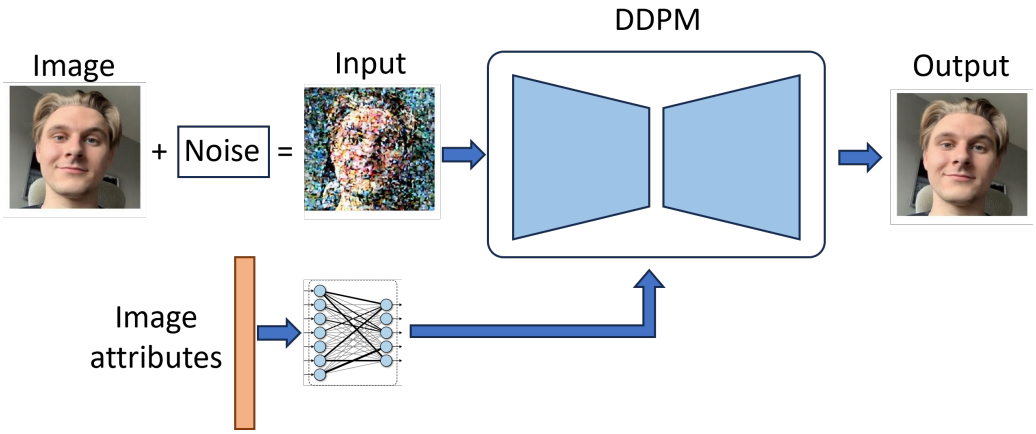

Figure 1: The overall architecture of our model. The DDPM model takes the noisy image as an input and iteratively denoises by predicting the noise contained in the image. During testing, only pure noise is fed to the DDPM model along with the transformed face attributes and it generates face images.

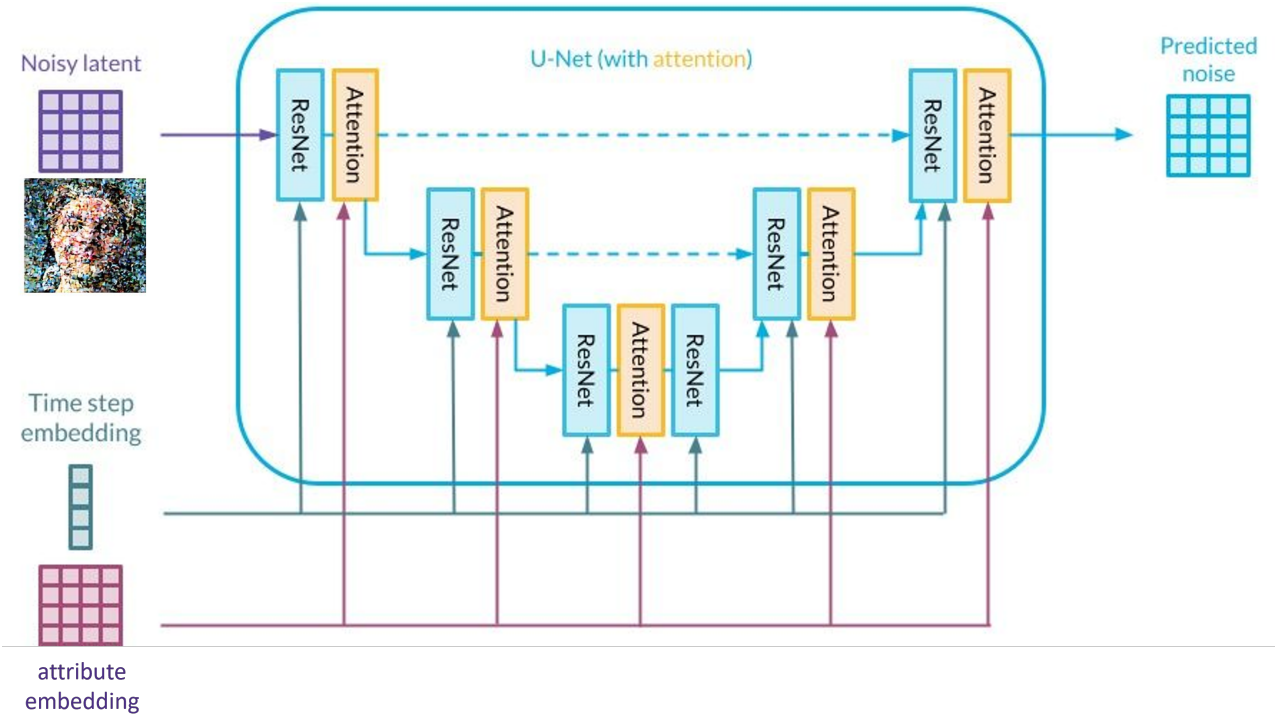

Figure 2: Our model architecture uses a U-Net model with ResNet Blocks and self attention layers. The timestep and learned attribute embedding is fed to the network at every iteration.

Basic Architecture of Denoising Diffusion Probabilistic Models (DDPM)

Denosing Diffusion Probabilistic Models (DDPM) is a type of generative model that focuses on the diffusion process. The key idea is to model the data distribution by iteratively applying a series of diffusion steps to a simple base distribution (e.g., a Gaussian distribution) until it reaches the desired data distribution.

Diffusion Process

At each diffusion step, noise is added to the data. The diffusion process is typically modeled as a series of conditional transformations, where the current data is conditioned on the previous data and noise.

Likelihood Estimation

The goal is to estimate the likelihood of the data under the diffusion process. The model is trained to generate data samples and is optimized to maximize the likelihood of the observed data.

Invertible Neural Networks

Invertible neural networks are often used to model the conditional transformations in the diffusion process. In our case, this is the U-Net as described in Figure 2. These networks allow for efficient computation of both forward and inverse transformations.

Classifier-Free Guidance Technique:

Incorporating a classifier-free guidance technique involves leveraging the latent space of the model to guide the generation process without relying on explicit classifiers. This can be achieved by using a mechanism that encourages certain features or characteristics in the latent space during the training process. In our case, the attribute vector of the CelebA images are mapped to a latent embedding of vector size 5 through a fully connected neural network.

0.0.1 Baseline

In this particular project, we did not have a baseline to compare our results with, since we have trained our model from scratch. However, in the results section, we present a comparison of image generation results when compared between the unconditional model and the conditional model. The basic code was inspired from the works of [HJA20] and [HS22].

Training

Dataset

We use two datasets: MNIST and CalebA. We performe class-conditioned image generation in the MNIST dataset and facial attribute conditioned image generation in the CelebA dataset. The CelebA dataset contains 200k images split between test, train and valid folder. We only use the training set for our image generation.

Training details

- We use the cosine learning rate schedule for the DDPM training.
- The total number of parameters in the U-Net model is 1.8M.
- We used MSE loss for the error predictions and Adam optimizer for gradient descent.
- We trained our model in Google Colab using the MNIST dataset and the train set of CelebA. Each epoch took around 30 minutes. We have trained our model for 12 epochs.

Results

We present our results in this section. The training curve for the CelebA dataset is shown in Figure3.

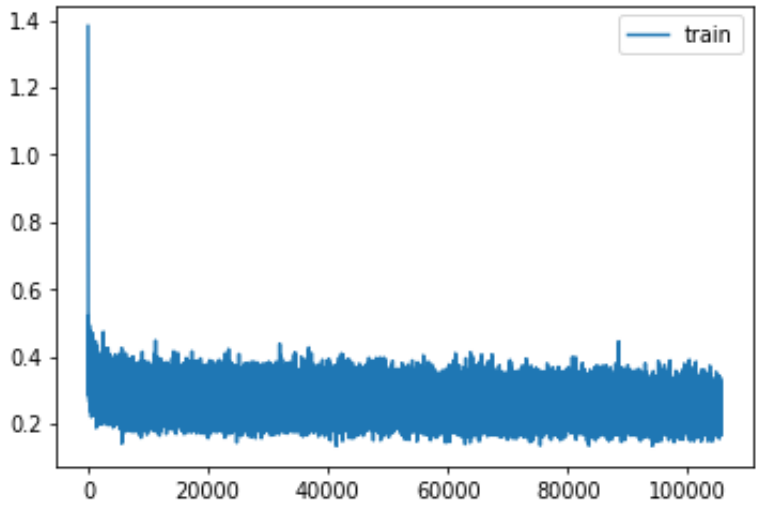

Figure 3: The training loss curve for the DDPM model using the CelebA dataset

We used the Fréchet Inception Distance (FID) and Inception score (IS) on a subset of generated images to evaluate the image generation quality of our DDPM model. The results are shown in Table 1.

The higher FID in the conditional model suggests better quality of images, whereas a slightly lower IS indicative of lesser diversity, mist likely due to the subset of images that were taken into consideration.

Given a larger training dataset and more evaluation time, the FID and IS values are expected to improve.

The generated images for the MNIST datset is shown in Figure 4. The class condition images from zero to nine shows a good generation cpaability of our model.

Model	FID	IS
Baseline (unconditional)	5.268	3.425
Conditional	4.356	3.409

Table 1: Comparison of FID and IS for different models.

Figure 4: Generated images of DDPm when trained on MNIST dataset, conditioned on class 0-9

Subsequently, we show the generated facial images when trained on the CelebA data set in Figure 5. In this particular data set, the image generation results are impressive and we show the attribute interpolation in Figure 6

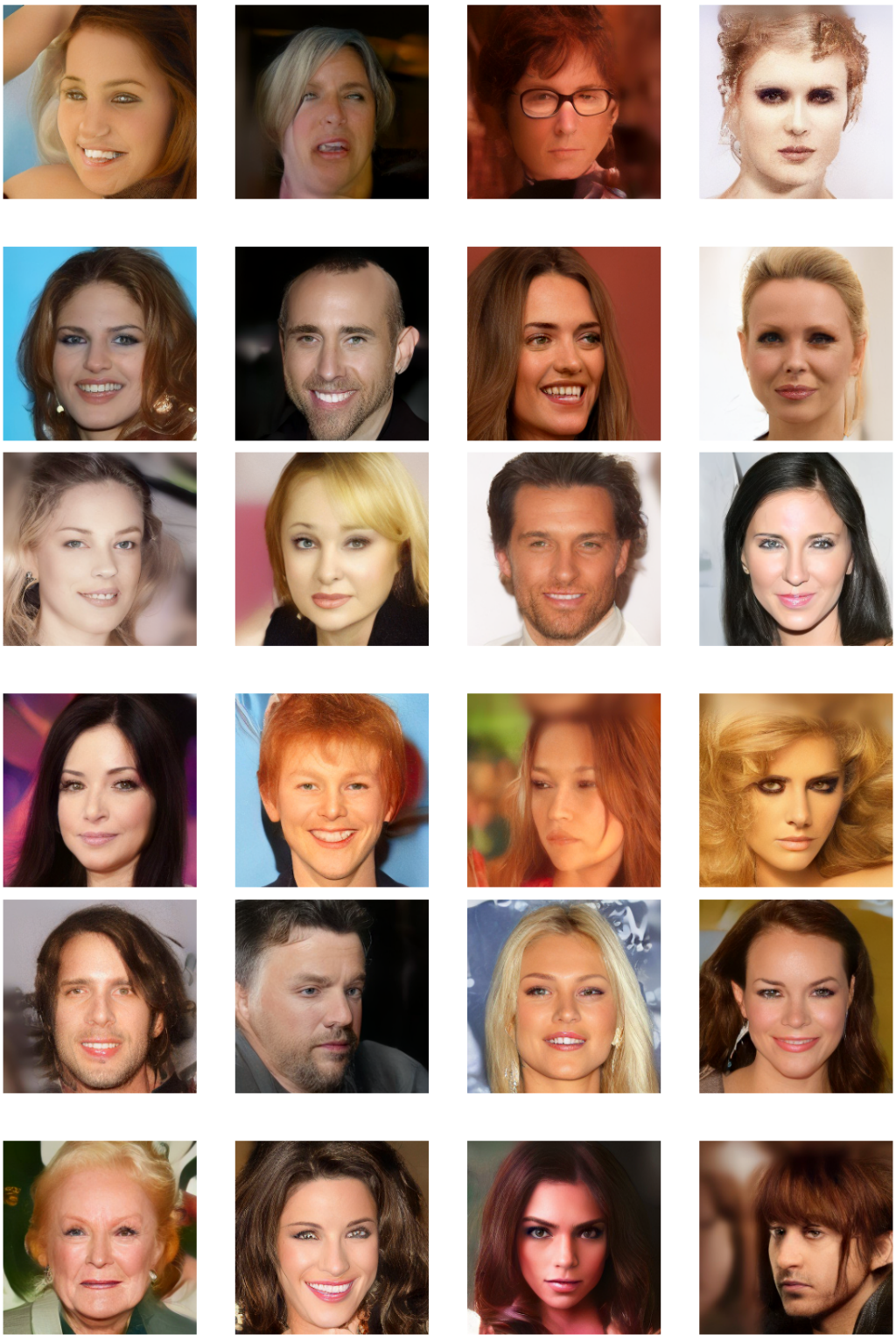

Figure 5: Generated images of DDPM when trained on CelebA dataset, conditioned on 40 facial attributes

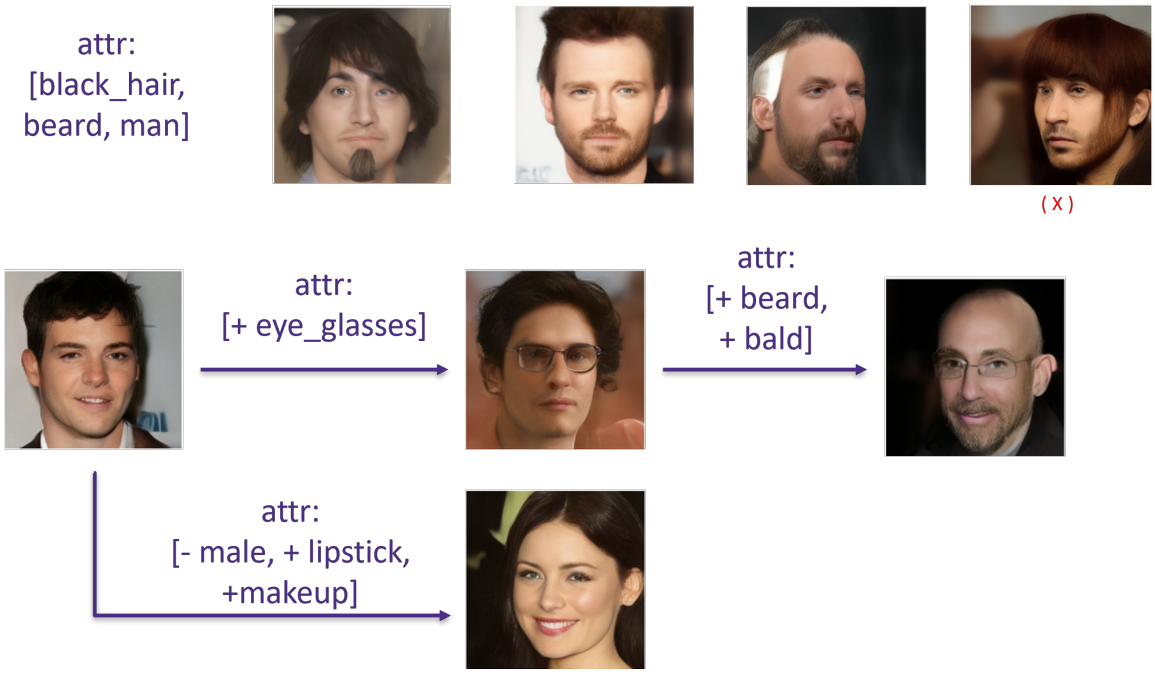

Figure 6: Face attribute interpolation results in CelebA

The conditional image generation results show us that the model is successfully able to associate different face attributes in its image generation process. For example, in this particular interpolation, we can see that adding facial attributes such as eyeglasses, beard and inverting male-female etc. results in successful guided image generation.

Conclusion

This particular project in Computer Vision brought together a plethora of emotions. What initially started as an HOI image generation project, went more into facial image editing. The primary challenge in this project was training the model from scratch. We realized that training is expensive (time and resources), that increases exponentially with model parameter increase. Also, specialized applications require a unique model architecture, so pre-trained weights are not always available, especially when attempting HOI synthesis.

The primary contributions/learning from this project are the following:

- Implemented and explored the core diffusion model architecture and experimented with different possibilities of achieving guided image generation
- Developed a complete conditional image generation pipeline from scratch. The generated face images for CelebA are of good quality
- Demostrated smooth interpolation between image attributes – an improvement of this model can lead to various downstream application in GenAI, including image editing

Armed with the knowledge from this project and the Computer Vision course in general, we would like to attempt the HOI synthesis project in future. We would like to explore our initial hypothesis that "A simple diffusion model with object pose embeddings as the conditional data can be the solution to complex model architectures and will generalize better in HOI synthesis tasks".

Appendix A

Setp and Data exploration:

```
import torch
import torchvision
from torch import nn
from torch.nn import functional as F
from torch.utils.data import DataLoader
from diffusers import DDPMScheduler, UNet2DModel
from matplotlib import pyplot as plt
from tqdm.auto import tqdm

import os

device = 'mps' if torch.backends.mps.is_available() else 'cuda' if torch.cuda.is_available
print(f'Using device: {device}')

# EXPLORING THE DATASET
import torchvision.transforms as T

img_size = 64
transforms = T.Compose([
        T.Resize((img_size, img_size)),
        T.ToTensor(),
        T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

# Load the dataset
# Using the "attr" (attribute) labels here for more control in image generation
dataset = torchvision.datasets.CelebA(root="celeba/", split="Train", target_type="attr", d

# Feed it into a dataloader (batch size 8 here just for demo)
train_dataloader = DataLoader(dataset, batch_size=8, shuffle=True)
# View some examples
x, y = next(iter(train_dataloader))
print('Input shape:', x.shape)
print('Sample Label for the first datapoint:', y[0])
plt.imshow(torchvision.utils.make_grid(x).numpy().transpose(1,2,0))
```

The Diffusion model architecture:

```
class ClassConditionedUnet(nn.Module):
  def __init__(self, class_attr=40, class_emb_size=5):
    super().__init__()

    # The linear layer will map the class attributes to a vector of size class_emb_size
    self.class_emb = nn.Linear(class_attr, class_emb_size)

    # Self.model is an unconditional UNet with extra input channels to accept the condition
```

```
        self.model = UNet2DModel(
            sample_size=64,           # the target image resolution
            in_channels=3 + class_emb_size, # Additional input channels for class cond.
            out_channels=3,           # the number of output channels
            layers_per_block=2,       # how many ResNet layers to use per UNet block
            block_out_channels=(32, 32, 32, 64),
            down_block_types=(
                "DownBlock2D",        # a regular ResNet downsampling block
                "AttnDownBlock2D",    # a ResNet downsampling block with spatial self-attentio
                "AttnDownBlock2D",
                "AttnDownBlock2D",
            ),
            up_block_types=(
                "AttnUpBlock2D",
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "AttnUpBlock2D",      # a ResNet upsampling block with spatial self-attention
                "UpBlock2D",          # a regular ResNet upsampling block
              ),
        )

      # Our forward method now takes the class labels as an additional argument
      def forward(self, x, t, class_attr):
        # Shape of x:
        bs, ch, w, h = x.shape
        # print("class_attr shape = ", class_attr.shape, class_attr.type)
        # class conditioning in right shape to add as additional input channels
        class_cond = self.class_emb(class_attr.float().to(device)) # Map to embedding dimensio
        # print("class_cond shape =", class_cond.shape)

        class_cond = class_cond.view(bs, class_cond.shape[1], 1, 1).expand(bs, class_cond.shap
        # x is shape (bs, 3, 64, 64) and class_cond is now (bs, 5, 64, 64)

        # Net input is now x and class cond concatenated together along dimension 1
        net_input = torch.cat((x, class_cond), 1) # (bs, 8, 64, 64)

        # Feed this to the UNet alongside the timestep and return the prediction
        return self.model(net_input, t).sample # (bs, 8, 64, 64)
```

Training and Sampling:

```
# Redefining the dataloader to set the batch size higher than the demo of 8
train_dataloader = DataLoader(dataset, batch_size=128, shuffle=True)
train_dataloader = accelerator.prepare(train_dataloader)
# How many runs through the data should we do?
n_epochs = 10

# Our network
net = ClassConditionedUnet().to(device)

trainable_params = sum(
        p.numel() for p in net.parameters() if p.requires_grad
)

print("total no.of parameetrs in unet model = ", trainable_params)
```

Model Training

```
# Create a scheduler
noise_scheduler = DDPMScheduler(num_train_timesteps=1000, beta_schedule='squaredcos_cap_v2
```

```
# if you want to strat training from a previous checkpoint then run this cell
# otherwise skip
epoch_no = 10
path = "drive/MyDrive/Colab Notebooks/"+str(epoch_no)+"_ckpt.pt"
net.load_state_dict(torch.load(path, map_location=torch.device('cpu')))

# Our loss function
loss_fn = nn.MSELoss()

# The optimizer
opt = torch.optim.Adam(net.parameters(), lr=1e-3)

# Keeping a record of the losses for later viewing
losses = []

# The training loop
for epoch in range(n_epochs):
    for x, y in tqdm(train_dataloader):

        # Get some data and prepare the corrupted version
        x = x.to(device) * 2 - 1 # Data on the GPU (mapped to (-1, 1))
        y = y.to(device)
        noise = torch.randn_like(x)
        timesteps = torch.randint(0, 999, (x.shape[0],)).long().to(device)
        noisy_x = noise_scheduler.add_noise(x, noise, timesteps)

        # Get the model prediction
        pred = net(noisy_x, timesteps, y) # Note that we pass in the labels y

        # Calculate the loss
        loss = loss_fn(pred, noise) # How close is the output to the noise

        # Backprop and update the params:
        opt.zero_grad()
        accelerator.backward(loss)
        opt.step()

        # Store the loss for later
        losses.append(loss.item())

    # Print out the average of the last 100 loss values to get an idea of progress:
    avg_loss = sum(losses[-100:])/100
    print(f'Finished epoch {epoch}. Average of the last 100 loss values: {avg_loss:05f}')
    torch.save(net.state_dict(), os.path.join("./drive/MyDrive/Colab Notebooks/", str(epoch

# View the loss curve
plt.plot(losses)
```

Bibliography

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[HS22] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance". In: *arXiv preprint arXiv:2207.12598* (2022).