Medical Image Generation Using Diffusion Model

Important Libraries

```
!pip install -q --no-cache-dir lightning torchmetrics medmnist torch-
fidelity

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 13.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.6/731.6 kB 35.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.4/66.4 kB 255.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 70.7/70.7 kB 253.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.8/65.8 kB 161.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 596.7/596.7 kB 42.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.7/45.7 kB 245.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 69.8/69.8 kB 263.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.5/59.5 kB 160.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.9/129.9 kB 265.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 725.0/725.0 kB 51.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.3/88.3 kB 283.7 MB/s eta
0:00:00
etadata (setup.py) ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
67.0/67.0 kB 249.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.4/58.4 kB 141.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 242.9 MB/s eta
0:00:00

import lightning as L
from lightning.pytorch import Trainer, seed_everything
from lightning.pytorch.loggers import TensorBoardLogger, CSVLogger
from lightning.pytorch.callbacks.early_stopping import EarlyStopping
from lightning.pytorch.callbacks import LearningRateMonitor,
ModelCheckpoint
```

```
from torchmetrics.image.fid import FrechetInceptionDistance
from torchmetrics.image.kid import KernelInceptionDistance

from google.colab.patches import cv2_imshow

import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import torch.nn.functional as F

import torchvision.transforms as transforms
from torchvision.transforms import Compose, ToTensor, Lambda,
ToPILImage, Resize

from PIL import Image

import numpy as np
import matplotlib.pyplot as plt

from medmnist.dataset import MedMNIST
from medmnist.info import INFO
from medmnist.utils import montage2d

import os
import cv2
import random
import math

import warnings

warnings.filterwarnings("ignore")

%matplotlib inline
plt.rcParams['axes.facecolor'] = 'lightgray'
plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
```

Configuration

```
MAX_EPOCH = 100
BATCH_SIZE = 200
LR = 4.3e-02
CHECKPOINT_DIR = os.getcwd()
FLAG = "bloodmnist"
IMAGE_SIZE = 32
N_CHANNEL = INFO[FLAG]['n_channels']
```

Dataset

Configuration

```python
DATA_SEED = int(np.random.randint(2147483647))
print(f"Random seed: {DATA_SEED}")
```

```
Random seed: 871242078
```

Utils

```python
image_transform = Compose(
    [
        Resize(IMAGE_SIZE),
        ToTensor(),
        Lambda(lambda x: (x * 2) - 1),
    ]
)

class MedMNIST2D(MedMNIST):
    @property
    def __prune__(self):
        random.seed(DATA_SEED)
        random.shuffle(self.imgs)
        prune_idx = len(self.imgs) - (len(self.imgs) % BATCH_SIZE)
        if self.split == "train":
            self.imgs = self.imgs[:prune_idx]
        elif self.split == "test":
            self.imgs = self.imgs[:prune_idx]
        elif self.split == "val":
            self.imgs = self.imgs[:prune_idx]
        else:
            raise ValueError

    def __getitem__(self, index):
        """
        return: (without transform/target_transofrm)
            img: PIL.Image
        """
        img = self.imgs[index]
        img = Image.fromarray(img)

        if self.as_rgb:
            img = img.convert("RGB")

        if self.transform is not None:
            img = self.transform(img)

        return img
    def montage(self, length=20, replace=False, save_folder=None):
        n_sel = length * length
        sel = np.random.choice(self.__len__(), size=n_sel,
replace=replace)

        montage_img = montage2d(
            imgs=self.imgs, n_channels=self.info["n_channels"],
sel=sel
        )

        if save_folder is not None:
            if not os.path.exists(save_folder):
                os.makedirs(save_folder)
            montage_img.save(
                os.path.join(save_folder,
f"{self.flag}_{self.split}_montage.jpg")
            )

        return montage_img


class BiomedicalDataset(MedMNIST2D):
    flag = FLAG

TrainDataset = BiomedicalDataset(
    split="train",
    transform=image_transform,
    download=True,
)
TestDataset = BiomedicalDataset(
    split="test",
    transform=image_transform,
    download=True,
)
ValDataset = BiomedicalDataset(
    split="val",
    transform=image_transform,
)
```

```
Downloading https://zenodo.org/record/6496656/files/bloodmnist.npz?
download=1 to /root/.medmnist/bloodmnist.npz
```

```
100%|██████████| 35461855/35461855 [00:06<00:00, 5656003.44it/s]
```

```
Using downloaded and verified file: /root/.medmnist/bloodmnist.npz
```

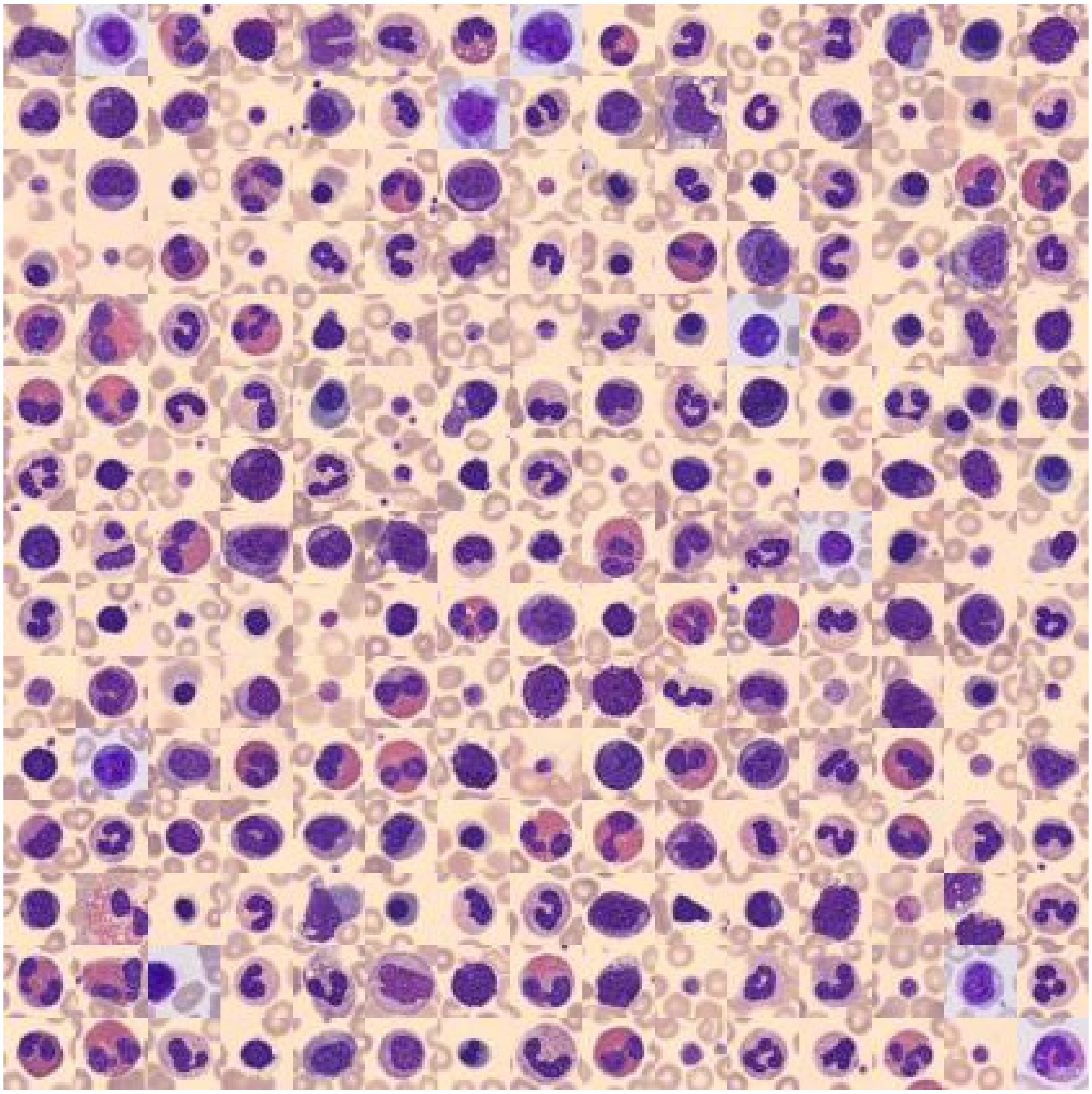
```
TrainDataset.__prune__
TestDataset.__prune__
ValDataset.__prune__
```

Description

```
INFO[FLAG]['description']
```

```
{"type":"string"}
```

```
TrainDataset.montage(15)
```



Model

Utils

```
def get_index_from_list(vals, t, x_shape):
    """
    Returns a specific index t of a passed list of values vals
    while considering the batch dimension.
    """
    batch_size = t.shape[0]
    out = vals.gather(-1, t.cpu())
    return out.reshape(batch_size, *((1,) * (len(x_shape) -
1))).to(t.device)

class AvgMeter(object):
    def __init__(self, num=40):
        self.num = num
        self.reset()

    def reset(self):
        self.scores = []

    def update(self, val):
        self.scores.append(val)

    def show(self):
        out = torch.mean(
            torch.stack(
                self.scores[np.maximum(len(self.scores)-self.num, 0):]
            )
        )
        return out
```

Noise Scheduler

```
class NoiseScheduler(nn.Module):
    def __init__(self, T=500):
        super().__init__()

        self.T = T

        self.betas = torch.linspace(1e-04, 2e-02, self.T)
        alphas = 1.0 - self.betas
        alphas_cumprod = torch.cumprod(alphas, axis=0)
        alphas_cumprod_prev = F.pad(alphas_cumprod[:-1], (1, 0),
value=1.0)
        self.sqrt_recip_alphas = torch.sqrt(1.0 / alphas)
        self.sqrt_alphas_cumprod = torch.sqrt(alphas_cumprod)
        self.sqrt_one_minus_alphas_cumprod = torch.sqrt(1.0 -
alphas_cumprod)
        self.posterior_variance = (
            self.betas * (1.0 - alphas_cumprod_prev) / (1.0 -
alphas_cumprod)
        )

    def forward(self, x, t):
        return self._forward_diffusion_sample(x, t)
    def _forward_diffusion_sample(self, x_0, t):
        """
        Takes an image and a timestep as input and
        returns the noisy version of it
        """
        device = "cpu" if not torch.cuda.is_available() else "cuda"
        noise = torch.randn_like(x_0)
        sqrt_alphas_cumprod_t = get_index_from_list(
            self.sqrt_alphas_cumprod, t, x_0.shape
        )
        sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
            self.sqrt_one_minus_alphas_cumprod, t, x_0.shape
        )
        # mean + variance
        return sqrt_alphas_cumprod_t.to(device) * x_0.to(
            device
        ) + sqrt_one_minus_alphas_cumprod_t.to(device) *
noise.to(device), noise.to(
            device
        )

FORWARD = NoiseScheduler
```

U-Net

```
class Block(nn.Module):
    def __init__(self, in_ch, out_ch, time_emb_dim, up=False):
        super().__init__()
        self.time_mlp = nn.Linear(time_emb_dim, out_ch)
        if up:
            self.conv1 = nn.Conv2d(2 * in_ch, out_ch, 3, padding=1)
            self.transform = nn.ConvTranspose2d(out_ch, out_ch, 4, 2,
1)
        else:
            self.conv1 = nn.Conv2d(in_ch, out_ch, 3, padding=1)
            self.transform = nn.Conv2d(out_ch, out_ch, 4, 2, 1)
        self.conv2 = nn.Conv2d(out_ch, out_ch, 3, padding=1)
        self.bnorm1 = nn.BatchNorm2d(out_ch)
        self.bnorm2 = nn.BatchNorm2d(out_ch)
        self.relu = nn.ReLU()

    def forward(
        self,
        x,
        t,
    ):
        # First Conv
        h = self.bnorm1(self.relu(self.conv1(x)))
        # Time embedding
        time_emb = self.relu(self.time_mlp(t))
        # Extend last 2 dimensions
        time_emb = time_emb[(...,) + (None,) * 2]
        # Add time channel
        h = h + time_emb
        # Second Conv
        h = self.bnorm2(self.relu(self.conv2(h)))
        # Down or Upsample
        return self.transform(h)


class PositionalEmbedding(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim

    def forward(self, time):
        device = time.device
        half_dim = self.dim // 2
        embeddings = math.log(10000) / (half_dim - 1)
        embeddings = torch.exp(torch.arange(half_dim, device=device) *
-embeddings)
        embeddings = time[:, None] * embeddings[None, :]
        embeddings = torch.cat((embeddings.sin(), embeddings.cos()),
dim=-1)
        # TODO: Double check the ordering here
        return embeddings


class UNet(nn.Module):
    def __init__(self, image_channels=N_CHANNEL, scale=4):
        super().__init__()

        down_channels = (
            64 // scale,
            128 // scale,
            256 // scale,
            512 // scale,
            1024 // scale,
        )
        up_channels = (
            1024 // scale,
            512 // scale,
            256 // scale,
            128 // scale,
            64 // scale,
        )
        out_dim = 3
        time_emb_dim = 32

        # Time embedding
        self.time_mlp = nn.Sequential(
            PositionalEmbedding(time_emb_dim),
            nn.Linear(time_emb_dim, time_emb_dim),
            nn.ReLU(),
        )

        # Initial projection
        self.conv0 = nn.Conv2d(image_channels, down_channels[0], 3,
padding=1)

        # Downsample
        self.downs = nn.ModuleList(
            [
                Block(down_channels[i], down_channels[i + 1],
time_emb_dim)
                for i in range(len(down_channels) - 1)
            ]
        )
        # Upsample
        self.ups = nn.ModuleList(
            [
                Block(up_channels[i], up_channels[i + 1],
time_emb_dim, up=True)
                for i in range(len(up_channels) - 1)
            ]
        )

        # Edit: Corrected a bug found by Jakub C (see YouTube comment)
        self.output = nn.Conv2d(up_channels[-1], out_dim, 1)

    def forward(self, x, timestep):
        # Embedd time
        t = self.time_mlp(timestep)
        # Initial conv
        x = self.conv0(x)
        # Unet
        residual_inputs = []
        for down in self.downs:
            x = down(x, t)
            residual_inputs.append(x)
        for up in self.ups:
            residual_x = residual_inputs.pop()

            # Add residual x as additional channels
            x = torch.cat((x, residual_x), dim=1)
            x = up(x, t)
        return self.output(x)

BACKWARD = UNet
```

Wrapper

```
class DiffusionModel(L.LightningModule):
    def __init__(
        self,
        forward_model,
        backward_model,
        batch_size,
        lr,
        max_epoch,
    ):
        super().__init__()

        self.forward_model = forward_model
        self.backward_model = backward_model

        self.batch_size = batch_size
        self.lr = lr
        self.max_epoch = max_epoch

        self.automatic_optimization = False

        self.model_loss = []
        self.val_fid = []
        self.val_mean_kid = []
        self.val_std_kid = []

        self.model_loss_recorder = AvgMeter()
        self.val_fid_recorder = AvgMeter()
        self.val_mean_kid_recorder = AvgMeter()
        self.val_std_kid_recorder = AvgMeter()

        self._device = "cuda" if torch.cuda.is_available() else "cpu"
        self._T = self.forward_model.T

        self.fid = FrechetInceptionDistance(feature=64,
normalize=True)
        self.kid = KernelInceptionDistance(
            feature=64,
            subset_size=50,
            normalize=True,
        )

    def forward(self, x, t):
        if self.training:
            x_noisy, noise = self.forward_model(x, t)
            noise_pred = self.backward_model(x_noisy, t)
            return F.l1_loss(noise, noise_pred)
        else:
            return self.backward_model(x, t)
    def on_train_epoch_start(self):
        self.fid.reset()
        self.kid.reset()

    def training_step(self, batch, batch_nb):
        x = batch
        self.fid.update((x + 1.0) / 2.0, real=True)
        self.kid.update((x + 1.0) / 2.0, real=True)
        t = torch.randint(0, self._T, (self.batch_size,),
device=self._device).long()
        loss = self(x, t)
        opt = self.optimizers()
        opt.zero_grad()
        self.manual_backward(loss)
        opt.step()
        self.log("model_loss", loss, prog_bar=True)
        self.model_loss_recorder.update(loss.data)

    def on_train_epoch_end(self):
        sch = self.lr_schedulers()
        sch.step()


self.model_loss.append(self.model_loss_recorder.show().data.cpu().nump
y())
        self.model_loss_recorder = AvgMeter()

    def validation_step(self, batch, batch_nb):
        x = batch
        self.fid.update((x + 1.0) / 2.0, real=True)
        self.kid.update((x + 1.0) / 2.0, real=True)

        _x = torch.randn(x.shape, device=self._device)
        for _t in range(0, self._T)[::-1]:
            t = torch.full((x.shape[0],), _t, device=self._device,
dtype=torch.long)
            betas_t = get_index_from_list(self.forward_model.betas, t,
_x.shape)
            sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
                self.forward_model.sqrt_one_minus_alphas_cumprod, t,
_x.shape
            )
            sqrt_recip_alphas_t = get_index_from_list(
                self.forward_model.sqrt_recip_alphas, t, _x.shape
            )
            model_mean = sqrt_recip_alphas_t * (
                _x - betas_t * self(_x, t) /
sqrt_one_minus_alphas_cumprod_t
            )
            posterior_variance_t = get_index_from_list(
                self.forward_model.posterior_variance, t, _x.shape
            )
            if torch.sum(t) == 0:
                _x = model_mean
            else:
                noise = torch.randn_like(_x)
                _x = model_mean + torch.sqrt(posterior_variance_t) *
noise.to(
                    self._device
                )
            _x = torch.clamp(_x, -1.0, 1.0)

        self.fid.update((_x + 1.0) / 2.0, real=False)
        self.kid.update((_x + 1.0) / 2.0, real=False)

        fid = self.fid.compute().data.cpu()
        self.log("val_fid", fid, prog_bar=True)
        self.val_fid_recorder.update(fid)

        mu_kid, std_kid = self.kid.compute()
        self.log("val_mean_kid", mu_kid.data.cpu(), prog_bar=True)
        self.log("val_std_kid", std_kid.data.cpu(), prog_bar=True)
        self.val_mean_kid_recorder.update(mu_kid.data.cpu())
        self.val_std_kid_recorder.update(std_kid.data.cpu())

    def on_validation_epoch_end(self):

self.val_fid.append(self.val_fid_recorder.show().data.cpu().numpy())
        self.val_fid_recorder = AvgMeter()


self.val_mean_kid.append(self.val_mean_kid_recorder.show().data.cpu().
numpy())
        self.val_mean_kid_recorder = AvgMeter()


self.val_std_kid.append(self.val_std_kid_recorder.show().data.cpu().nu
mpy())
        self.val_std_kid_recorder = AvgMeter()

    def test_step(self, batch, batch_nb):
        x = batch
        self.fid.update((x + 1.0) / 2.0, real=True)
        self.kid.update((x + 1.0) / 2.0, real=True)

        _x = torch.randn(x.shape, device=self._device)
        for _t in range(0, self._T)[::-1]:
            t = torch.full((x.shape[0],), _t, device=self._device,
dtype=torch.long)
            betas_t = get_index_from_list(self.forward_model.betas, t,
_x.shape)
            sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
                self.forward_model.sqrt_one_minus_alphas_cumprod, t,
_x.shape
            )
            sqrt_recip_alphas_t = get_index_from_list(
                self.forward_model.sqrt_recip_alphas, t, _x.shape
            )
            model_mean = sqrt_recip_alphas_t * (
                _x - betas_t * self(_x, t) /
sqrt_one_minus_alphas_cumprod_t
            )
            posterior_variance_t = get_index_from_list(
                self.forward_model.posterior_variance, t, _x.shape
            )
            if torch.sum(t) == 0:
                _x = model_mean
            else:
                noise = torch.randn_like(_x)
                _x = model_mean + torch.sqrt(posterior_variance_t) *
noise.to(
                    self._device
                )
            _x = torch.clamp(_x, -1.0, 1.0)
        self.fid.update((_x + 1.0) / 2.0, real=False)
        self.kid.update((_x + 1.0) / 2.0, real=False)

    def on_test_epoch_end(self):
        fid = self.fid.compute().data.cpu()
        self.log("test_fid", fid, prog_bar=False, logger=True)

        mu_kid, std_kid = self.kid.compute()
        self.log("mu_kid", mu_kid.data.cpu(), prog_bar=False,
logger=True)
        self.log("std_kid", std_kid.data.cpu(), prog_bar=False,
logger=True)

    def on_train_end(self):
        # Loss
        loss_img_file = f"/content/{MODEL_NAME}_loss_plot.png"
        plt.plot(self.model_loss, color="r")
        plt.title("Loss Curves")
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.grid()
        plt.savefig(loss_img_file)
        plt.clf()
        img = cv2.imread(loss_img_file)
        cv2_imshow(img)
        # Evaluation Metrics
        evaluation_metric_img_file =
f"/content/{MODEL_NAME}_fid_plot.png"
        plt.plot(self.val_fid[1:], color="b")
        plt.title("FID Curves")
        plt.xlabel("Epoch")
        plt.ylabel("FID")
        plt.grid()
        plt.savefig(evaluation_metric_img_file)
        plt.clf()
        img = cv2.imread(evaluation_metric_img_file)
        cv2_imshow(img)

        evaluation_metric_img_file =
f"/content/{MODEL_NAME}_kid_plot.png"
        self.val_mean_kid = np.array(self.val_mean_kid)[1:]
        self.val_std_kid = np.array(self.val_std_kid)[1:]
        epochs = list(range(self.max_epoch))
        fig, ax = plt.subplots()
        ax.plot(epochs, self.val_mean_kid)
        ax.fill_between(
            epochs,
            self.val_mean_kid - self.val_std_kid,
            self.val_mean_kid + self.val_std_kid,
            alpha=0.3,
        )
        ax.set_title("KID Curves")
        ax.set_xlabel("Epoch")
        ax.set_ylabel("KID")
        ax.grid()
        plt.savefig(evaluation_metric_img_file)
        plt.clf()
        img = cv2.imread(evaluation_metric_img_file)
        cv2_imshow(img)

    def train_dataloader(self):
        return data.DataLoader(
            dataset=TrainDataset,
            batch_size=self.batch_size,
            shuffle=True,
        )

    def val_dataloader(self):
        return data.DataLoader(
            dataset=ValDataset,
            batch_size=self.batch_size,
            shuffle=False,
        )

    def test_dataloader(self):
        return data.DataLoader(
            dataset=TestDataset,
            batch_size=self.batch_size,
            shuffle=False,
        )

    def configure_optimizers(self):
        optimizer = optim.SGD(
            self.parameters(),
            lr=self.lr,
            weight_decay=1e-4,
            momentum=0.9,
            nesterov=True,
        )

        lr_scheduler = optim.lr_scheduler.MultiStepLR(
            optimizer,
            milestones=[
                int(self.max_epoch * 0.01),
                int(self.max_epoch * 0.12),
                int(self.max_epoch * 0.23),
                int(self.max_epoch * 0.34),
                int(self.max_epoch * 0.45),
                int(self.max_epoch * 0.56),
                int(self.max_epoch * 0.67),
                int(self.max_epoch * 0.78),
                int(self.max_epoch * 0.89),
                int(self.max_epoch * 0.90),
            ],
            gamma=(
                ((1.0 + math.sqrt(5)) / 2.0)
                - (1.0 / math.pi + 1.0 / math.e + 1.0 / math.tau)
            )
            / math.sqrt(2),
        )

        return [optimizer], [lr_scheduler]

MODEL_NAME = DiffusionModel.__name__
```

Training

```
SEED = int(np.random.randint(2147483647))
print(f"Random seed: {SEED}")
```

```
Random seed: 1345331842
```

```
%reload_ext tensorboard
%tensorboard --logdir=logs/lightning_logs/
```

```
seed_everything(SEED, workers=True)

model = DiffusionModel(FORWARD(), BACKWARD(), BATCH_SIZE, LR,
MAX_EPOCH)

tensorboardlogger = TensorBoardLogger(save_dir="logs/")
csvlogger = CSVLogger(save_dir="logs/")
checkpoint = ModelCheckpoint(
    monitor='val_fid',
    dirpath=CHECKPOINT_DIR,
    mode='min',
)

trainer = Trainer(
    accelerator="auto",
    devices=1,
    max_epochs=MAX_EPOCH,
    logger=[tensorboardlogger, csvlogger],
    callbacks=[checkpoint],
    log_every_n_steps=5,
)
trainer.fit(model)
```

```
INFO: Global seed set to 1345331842
INFO:lightning.fabric.utilities.seed:Global seed set to 1345331842
Downloading:
"https://github.com/toshas/torch-fidelity/releases/download/v0.2.0/
weights-inception-2015-12-05-6726825d.pth" to
/root/.cache/torch/hub/checkpoints/weights-inception-2015-12-05-
6726825d.pth
100%|██████████| 91.2M/91.2M [00:02<00:00, 41.7MB/s]
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda),
used: True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False,
using: 0 TPU cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False,
using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False,
using: 0 HPUs
WARNING: Missing logger folder: logs/lightning_logs
WARNING:lightning.pytorch.loggers.tensorboard:Missing logger folder:
logs/lightning_logs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
INFO:
  | Name           | Type                      | Params
-------------------------------------------------------------
0 | forward_model  | NoiseScheduler            | 0
1 | backward_model | UNet                      | 3.9 M
2 | fid            | FrechetInceptionDistance | 23.9 M
3 | kid            | KernelInceptionDistance  | 23.9 M
-------------------------------------------------------------
3.9 M     Trainable params
47.7 M    Non-trainable params
51.6 M    Total params
206.509   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name           | Type                      | Params
-------------------------------------------------------------
0 | forward_model  | NoiseScheduler            | 0
1 | backward_model | UNet                      | 3.9 M
2 | fid            | FrechetInceptionDistance | 23.9 M
3 | kid            | KernelInceptionDistance  | 23.9 M
-------------------------------------------------------------
3.9 M     Trainable params
47.7 M    Non-trainable params
51.6 M    Total params
206.509   Total estimated model params size (MB)
```

```
{"model_id":"02c946b3e9004919b74e316ae1e92b23","version_major":2,"version_minor":0}

{"model_id":"9d6b17f8713344f39f8017d9dc619515","version_major":2,"version_minor":0}

{"model_id":"8dda347970a44974a2bf5aae2a34f416","version_major":2,"version_minor":0}

{"model_id":"2953a77584af4759a7580d30e1d91536","version_major":2,"version_minor":0}

{"model_id":"f5e9587dd9b74ee985f916d0d2fa30f0","version_major":2,"version_minor":0}

{"model_id":"277de28e108e4a7fa55e7330efc48abb","version_major":2,"version_minor":0}

{"model_id":"d4ded8d37cd74e1eadefc6e50365604c","version_major":2,"version_minor":0}

{"model_id":"287dee1e03eb4ea7a08c7ecd4762a375","version_major":2,"version_minor":0}

{"model_id":"bcf1493f6e28406aa128f1ea1bf73f6c","version_major":2,"version_minor":0}
```

```
{"model_id":"18269fe6ac7b49ef8caca532cdb4f6c9","version_major":2,"version_minor":0}

{"model_id":"52e15230f3fa4f1f971472b6677bac27","version_major":2,"version_minor":0}

{"model_id":"fbbfb37bff6b4d11bb6ee9b1becdb929","version_major":2,"version_minor":0}

{"model_id":"3a7490899e38439cb7a04d9767fac1f5","version_major":2,"version_minor":0}

{"model_id":"2682f9ea63c44c71984ef663cccd71d0","version_major":2,"version_minor":0}

{"model_id":"2b734cd47efc4a438bb4781b7183f173","version_major":2,"version_minor":0}

{"model_id":"0c17b8c0e2c441d59634641c479b6523","version_major":2,"version_minor":0}

{"model_id":"5ab1a32d208c41acbf9f9a19355f0d12","version_major":2,"version_minor":0}

{"model_id":"475e47d3895f4b0196f8e2b06a4d356a","version_major":2,"version_minor":0}

{"model_id":"72de64efa74b451797167d8cbe11b755","version_major":2,"version_minor":0}

{"model_id":"d006522cd7d24c34ad98cf910e4d27a2","version_major":2,"version_minor":0}

{"model_id":"ca99007d774e421ba7e5dd8e651df770","version_major":2,"version_minor":0}

{"model_id":"93ff3523879d40d498e6499a1ca9359d","version_major":2,"version_minor":0}

{"model_id":"e905aa73881d4d7aaea8fffb5c48dd71","version_major":2,"version_minor":0}

{"model_id":"8927bf49bad246898af641f5fc193e1c","version_major":2,"version_minor":0}

{"model_id":"285ac643399e416e80646c7fc3eeaad9","version_major":2,"version_minor":0}

{"model_id":"fe21b543e87e465f916e35f6153074af","version_major":2,"version_minor":0}

{"model_id":"960428608eb246bf97901b97507b303b","version_major":2,"version_minor":0}
```

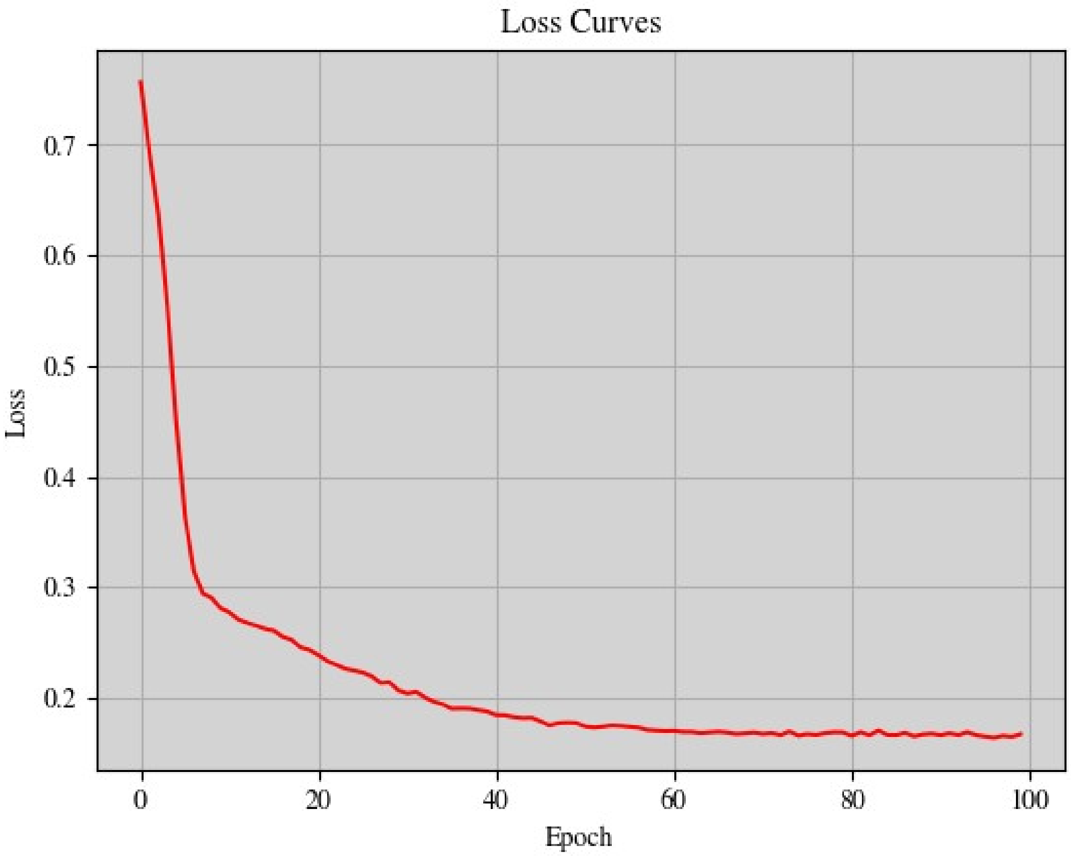
```
{"model_id":"001587aa0c1241319003df25ecd60ad8","version_major":2,"version_minor":0}

{"model_id":"d3131293fd064356a2f9ec199b325b4b","version_major":2,"version_minor":0}

{"model_id":"90eb43524d724015947507cfebeb2d28","version_major":2,"version_minor":0}

{"model_id":"cd9a3a23d0564268bf9c5fd3b5742578","version_major":2,"version_minor":0}

{"model_id":"34f4a2d1d08b4a07a8dea2fe37bf3210","version_major":2,"version_minor":0}

{"model_id":"257e0f0b622a4612b809729b85dd674a","version_major":2,"version_minor":0}

{"model_id":"24f36167b1204ba1bc951d81361c122a","version_major":2,"version_minor":0}

{"model_id":"87f0d2af01c943748f8763a41fd73ca7","version_major":2,"version_minor":0}

{"model_id":"b4c8a71a218c434aa53c67b9e45f45fc","version_major":2,"version_minor":0}

{"model_id":"fc227293ce71457ebe6c6c132cec748f","version_major":2,"version_minor":0}

{"model_id":"65a8def0a733491cb465385935d9248c","version_major":2,"version_minor":0}

{"model_id":"e5c9aabb501e4ba3bd199c5a1b9ddb4e","version_major":2,"version_minor":0}

{"model_id":"62846a6a8ca0478eb878663e5a3517b8","version_major":2,"version_minor":0}

{"model_id":"bd6d01f2e98e44f59ff6e290d1c0aa53","version_major":2,"version_minor":0}

{"model_id":"3a4cd3fc2ae24e1aba471a2b14490e66","version_major":2,"version_minor":0}

{"model_id":"1cc369300c3d41379a31a49f3f6c5c1b","version_major":2,"version_minor":0}

{"model_id":"c66607bcd0294cc2931abfe728910d4f","version_major":2,"version_minor":0}

{"model_id":"0d4070dab3f0401c9613e003850c5bf8","version_major":2,"version_minor":0}
```

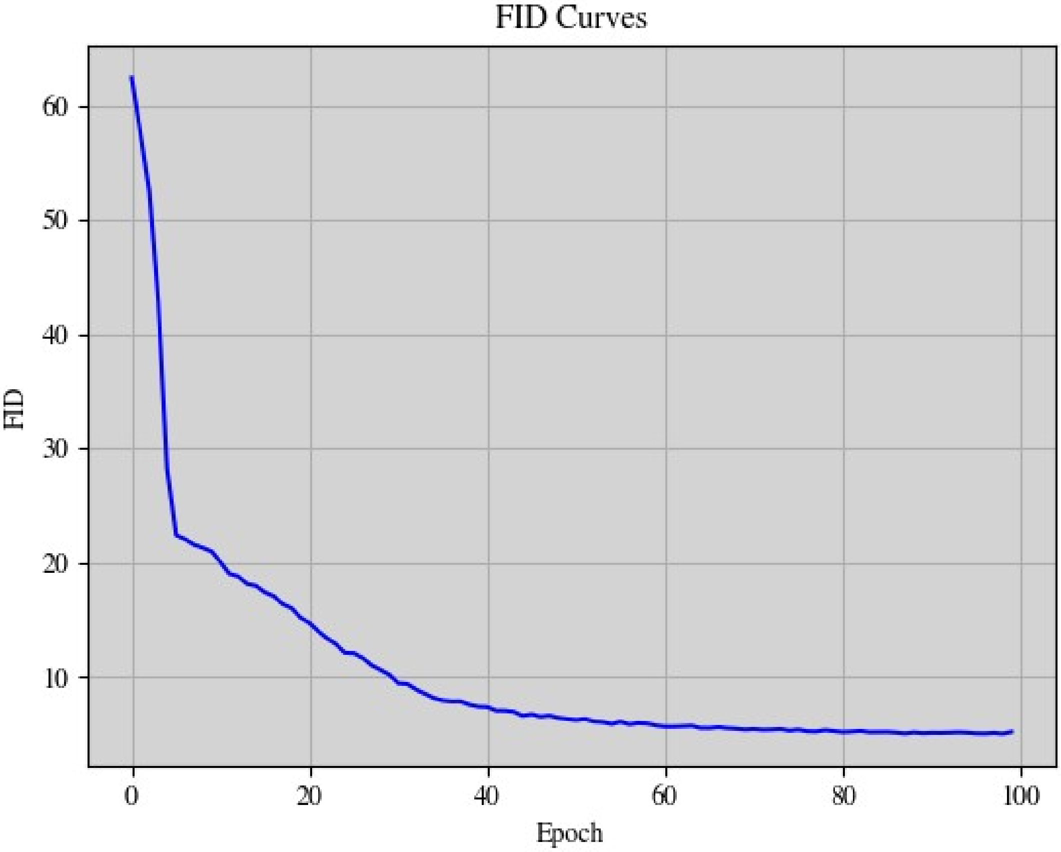
```
{"model_id":"7aa44ac6084b488db71fd02fc8a52127","version_major":2,"version_minor":0}

{"model_id":"6f28f11578274a66bb67ff0a0824b98a","version_major":2,"version_minor":0}

{"model_id":"80d2bdb499894c3a9d501ab65c4013f5","version_major":2,"version_minor":0}

{"model_id":"734e7ec0aabf4e3eb4c68b2e9f4e7796","version_major":2,"version_minor":0}

{"model_id":"e91acc5da85d483ab0964ca45228c20a","version_major":2,"version_minor":0}

{"model_id":"4c4ee7b1ea0d40e4adcbcb498cb4a724","version_major":2,"version_minor":0}

{"model_id":"3a45f55df559449b919909b25d61eeac","version_major":2,"version_minor":0}

{"model_id":"b6a2f4f510a44d9a964d993cf492b03c","version_major":2,"version_minor":0}

{"model_id":"97597620f0614d159930e4555ba9a8c4","version_major":2,"version_minor":0}

{"model_id":"ae027cf3225543d39cbe20d781be46f8","version_major":2,"version_minor":0}

{"model_id":"5b24c021ccb343688cb66d7874a61aa3","version_major":2,"version_minor":0}

{"model_id":"ce4eda83a13c41f5a1eb69b2b893d8c5","version_major":2,"version_minor":0}

{"model_id":"15b4e92ed59241b5af769fdc41189ec4","version_major":2,"version_minor":0}

{"model_id":"c09bbe355ad140dbba8c41f135465c2b","version_major":2,"version_minor":0}

{"model_id":"54cbdd48c93a422a853929db27597c1e","version_major":2,"version_minor":0}

{"model_id":"ce298981d9bf446b9b5ca1db4b3fe64f","version_major":2,"version_minor":0}

{"model_id":"f9bc671f9d984530b533bec464c5209c","version_major":2,"version_minor":0}

{"model_id":"586366960e0147dba5ab699121e77317","version_major":2,"version_minor":0}
```

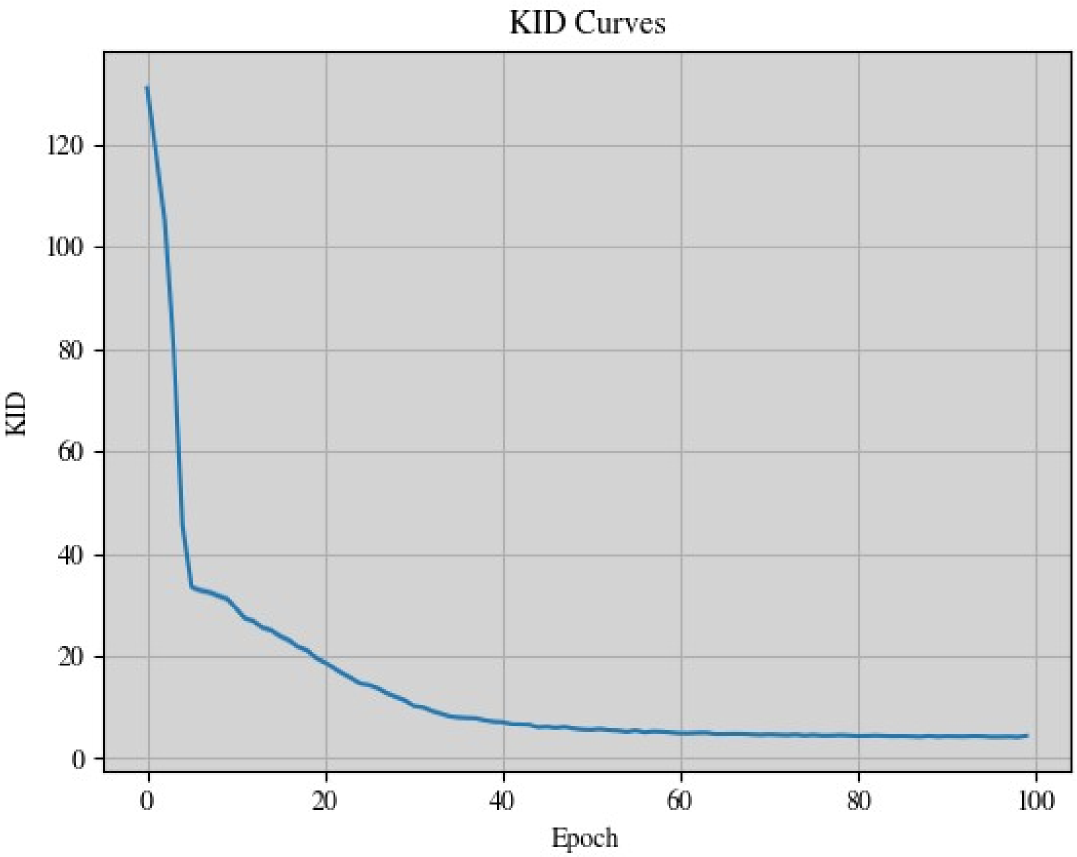
```
{"model_id":"00a3169d957f420dbb42734dea631ec2","version_major":2,"version_minor":0}

{"model_id":"036e22e664cf453ba97b569d590b4aab","version_major":2,"version_minor":0}

{"model_id":"830fd59bfced4bfb815cd6a204ecaed3","version_major":2,"version_minor":0}

{"model_id":"b2e951beaaac4565aec4a03af5d91045","version_major":2,"version_minor":0}

{"model_id":"38a63ebf32984ea7b0bb90a242d101bb","version_major":2,"version_minor":0}

{"model_id":"5c4832a671bb4165950ca0f659cb708f","version_major":2,"version_minor":0}

{"model_id":"9aedf285aa1f40d3aff58ddf2117d0a9","version_major":2,"version_minor":0}

{"model_id":"ffcc281826e44fccbd310e8de8a4cf2e","version_major":2,"version_minor":0}

{"model_id":"4cb3bf0f289b4aa19c3a4af5b09a0312","version_major":2,"version_minor":0}

{"model_id":"0f5ceddd1a5a4b1b999c642f004ad7b0","version_major":2,"version_minor":0}

{"model_id":"f5e515873cb34c67b7679542893d21ba","version_major":2,"version_minor":0}

{"model_id":"728a6ff852f144fab273d49132683faa","version_major":2,"version_minor":0}

{"model_id":"6eb95551cb514d7d8da6d31ec8042643","version_major":2,"version_minor":0}

{"model_id":"b6cbacca1ba2421b8f847799b594d2a9","version_major":2,"version_minor":0}

{"model_id":"911d7d3a1b4840ebae95a92f8e8b2fe4","version_major":2,"version_minor":0}

{"model_id":"35d95c1118c3451fb57188b1194fced4","version_major":2,"version_minor":0}

{"model_id":"96bca7ad350b448d8f9db2328aef04a5","version_major":2,"version_minor":0}

{"model_id":"2d755aac5203482a911952ac7ef8a791","version_major":2,"version_minor":0}
```

```
{"model_id":"3f49356c5db34178bdb18dba87f56722","version_major":2,"version_minor":0}

{"model_id":"10a44ad628694bfab33daeb10cf2c93b","version_major":2,"version_minor":0}

{"model_id":"32078495e0a44366a3326721f4996eb6","version_major":2,"version_minor":0}

{"model_id":"ac5e0c76efa449859077cecbe05cc082","version_major":2,"version_minor":0}

{"model_id":"d97059415df84035984c3cc04fe5dd59","version_major":2,"version_minor":0}

{"model_id":"26d66c302e1440118fd4ac4c5b6b9ada","version_major":2,"version_minor":0}

{"model_id":"90961361d296464b93f9f82137ae10b1","version_major":2,"version_minor":0}

{"model_id":"7f8516d30a104c3caad60bd46b96916c","version_major":2,"version_minor":0}

{"model_id":"329be1e28dbf4a91a13c1bf1b97d5b8a","version_major":2,"version_minor":0}

{"model_id":"f34cc5bf25c84badb740e497c00cb717","version_major":2,"version_minor":0}

{"model_id":"fc62fedaed294d01a0dd12ad4b72f48f","version_major":2,"version_minor":0}

{"model_id":"17992f9d7b534ab2a68b05e5811b4e50","version_major":2,"version_minor":0}

{"model_id":"fa27520423ee4ba091b2d7e1fbbb10bd","version_major":2,"version_minor":0}

{"model_id":"59d81f3ad37e430f98a33cc769dab085","version_major":2,"version_minor":0}

{"model_id":"9ee8aac48ac94db8beaad911919626d7","version_major":2,"version_minor":0}

{"model_id":"af35af92dcca468ea895b7c1f612dc67","version_major":2,"version_minor":0}

{"model_id":"55e50dc9ca434ecabe66dc539ecfc5be","version_major":2,"version_minor":0}

{"model_id":"9789afa0183749d8b0b2a929d8d424c0","version_major":2,"version_minor":0}
```

```
{"model_id":"45a90d2750ba4b329ad467987331974e","version_major":2,"version_minor":0}

{"model_id":"ae636e4c5c3040aebdb32dc124fe4e0c","version_major":2,"version_minor":0}

{"model_id":"a6b0dccc784b48e0a6213f66f1258e94","version_major":2,"version_minor":0}

INFO: `Trainer.fit` stopped: `max_epochs=100` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=100` reached.
```




FID Curves
FID
60
50
40
30
20
10
0
20
40
60
80
100
Epoch



```
<Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 0 Axes>
```

Testing

```
os.rename(
    checkpoint.best_model_path,
    os.path.join(CHECKPOINT_DIR, f"{MODEL_NAME}_best.ckpt")
)

trainer.test(ckpt_path=os.path.join(CHECKPOINT_DIR,
f"{MODEL_NAME}_best.ckpt"))
```

```
INFO: Restoring states from the checkpoint path at
/content/DiffusionModel_best.ckpt
INFO:lightning.pytorch.utilities.rank_zero:Restoring states from the
checkpoint path at /content/DiffusionModel_best.ckpt
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 -
CUDA_VISIBLE_DEVICES: [0]
INFO: Loaded model weights from the checkpoint at
```

```
/content/DiffusionModel_best.ckpt
INFO:lightning.pytorch.utilities.rank_zero:Loaded model weights from
the checkpoint at /content/DiffusionModel_best.ckpt
```

```
{"model_id":"2bbdbe25a7ab4bbabce91c598a3fbe40","version_major":2,"vers
ion_minor":0}
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃        Test metric        ┃       DataLoader 0        ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│          mu_kid           │     4.140571117401123     │
│          std_kid          │    0.3429620862007141     │
│         test_fid          │     5.038834571838379     │
└───────────────────────────┴───────────────────────────┘
```

```
[{'test_fid': 5.038834571838379,
  'mu_kid': 4.140571117401123,
  'std_kid': 0.3429620862007141}]
```

Inference

Utils

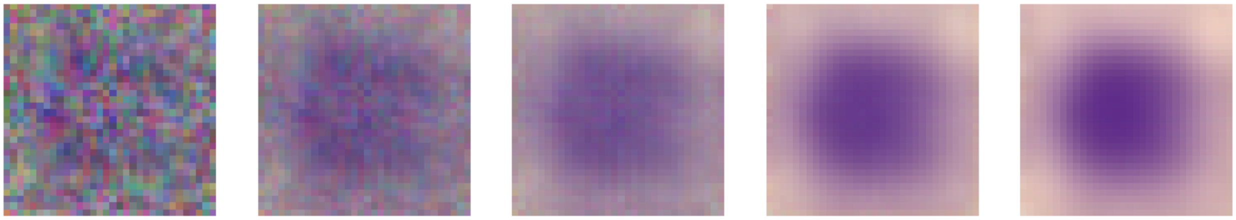
```python
def show_tensor_image(image):
    reverse_transforms = Compose([
        Lambda(lambda t: (t + 1) / 2),
        Lambda(lambda t: t.permute(1, 2, 0)), # CHW to HWC
        Lambda(lambda t: t * 255.),
        Lambda(lambda t: t.numpy().astype(np.uint8)),
        ToPILImage(),
        Resize(IMAGE_SIZE),
    ])

    # Take first image of batch
    if len(image.shape) == 4:
        image = image[0, :, :, :]

    plt.axis('off')
    plt.imshow(reverse_transforms(image))

DEVICE = "cpu" if not torch.cuda.is_available() else 'cuda'

model = DiffusionModel.load_from_checkpoint(
    checkpoint_path=os.path.join(CHECKPOINT_DIR,
f"{MODEL_NAME}_best.ckpt"),
    map_location=DEVICE,
    forward_model=FORWARD(),
    backward_model=BACKWARD(),
    batch_size=BATCH_SIZE,
    lr=LR,
    max_epoch=MAX_EPOCH,
)
model.eval()

T = 500
betas = torch.linspace(1e-04, 2e-02, T)
alphas = 1. - betas
alphas_cumprod = torch.cumprod(alphas, axis=0)
sqrt_recip_alphas = torch.sqrt(1.0 / alphas)
sqrt_one_minus_alphas_cumprod = torch.sqrt(1. - alphas_cumprod)

num_images = 5
stepsize = int(T/num_images)
```

Visualize

```
for _ in range(5):
    plt.figure(figsize=(15, 15))
    _x = torch.randn((1, N_CHANNEL, IMAGE_SIZE, IMAGE_SIZE),
device=DEVICE)
    for _t in range(0, T)[::-1]:
        t = torch.full((1,), _t, device=DEVICE, dtype=torch.long)
        betas_t = get_index_from_list(betas, t, _x.shape)
        sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
            sqrt_one_minus_alphas_cumprod, t, _x.shape
        )
        sqrt_recip_alphas_t = get_index_from_list(sqrt_recip_alphas,
t, _x.shape)
        model_mean = sqrt_recip_alphas_t * (
            _x - betas_t * model(_x, t) /
sqrt_one_minus_alphas_cumprod_t
        )
        _x = model_mean
        _x = torch.clamp(_x, -1.0, 1.0)
        if _t % stepsize == 0:
            plt.subplot(
                1,
                num_images,
                ((num_images + 1) - min(num_images, int(_t / stepsize)
+ 1)),
            )
            show_tensor_image(_x[:,:,1:-1,1:-1].detach().cpu())
    plt.show()
    plt.clf()
```

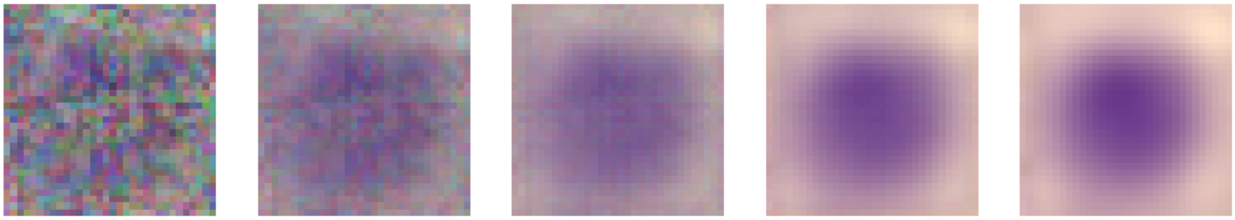
```
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
```



```
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
```



```
<Figure size 640x480 with 0 Axes>
```