Variational Autoencoder (VAE)

Justin Volheim

June 14, 2023

Abstract

In this project, I explored the application of variational autoencoders (VAEs) for generating images, specifically focusing on a data-set downloaded from Kaggle which contained chest X-ray images. By implementing a VAE using the PyTorch framework and training it on the data-set, we aim to generate realistic and high-quality images that resemble the characteristics of real X-rays both with pneumonia and without. To evaluate the generated images, this paper employs the Inception score and Fréchet Inception Distance (FID) metrics. In this report, we present the hyper-parameter settings used for training the VAE as well as the various visualize for quantifying the results.

1 Methodology

1.1 Data Preparation

We load the chest X-ray dataset and create data loaders for training, validation, and testing. The dataset is transformed to grayscale and resized to the desired image size.

1.2 Encoder and Decoder

We define the architecture of the encoder and decoder modules. The encoder encodes the input images into a lower-dimensional latent space, while the decoder reconstructs the images from the latent space.

1.3 Model Creation

We create the VAE model by combining the encoder and decoder modules. The model includes a reparameterization function for sampling from the latent space.

1.4 Loss Function

We define the loss function for the VAE, consisting of the reconstruction loss and the Kullback-Leibler divergence term.

1.5 Optimizer

We initialize the Adam optimizer to optimize the model parameters.

1.6 Tune Hyper-parameters

We define the hyper-parameters for our VAE model, such as the batch size, image size, input and latent dimensions, learning rate, and the number of epochs.

```
batch_size = 32
image_size = 128
x_dim = image_size*image_size
latent_dim = 128
lr = 1e-3
epochs = 10
```

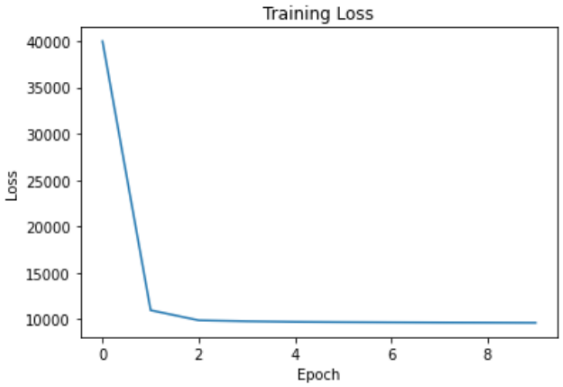
1.7 Training

We train the VAE by iterating over the training data for a specified number of epochs. We calculate the loss, perform backpropagation, and update the model parameters using the optimizer.

2 Results

2.1 Loss Visualization

We plot the training loss curve to visualize the training progress vs number of epochs.



2.2 Images

We generate reconstructed images by passing the test data through the trained VAE.

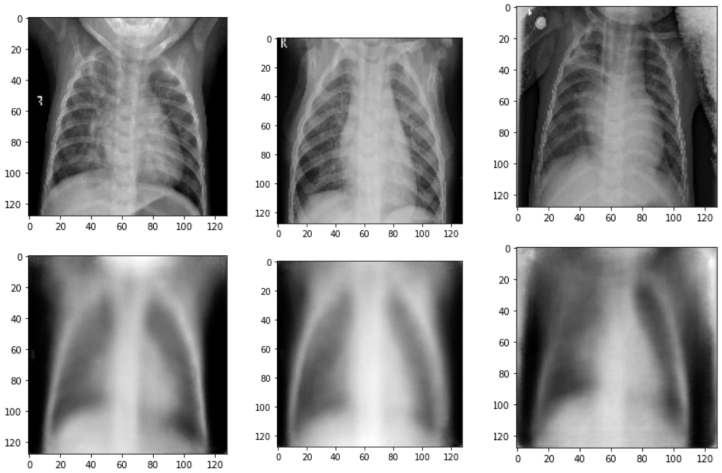


Figure 1: Normal

2.3 Images

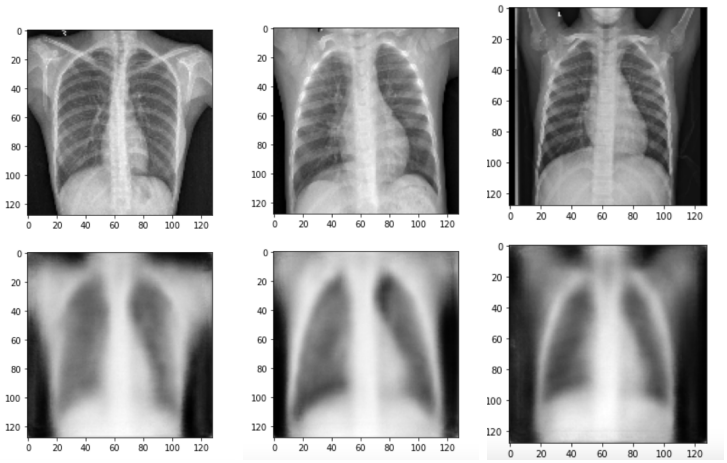


Figure 2: Pneumonia

2.4 Model Metrics

We apply Inception score and Frechet Inception Distance (FID) metrics to quantify how accurate our VAE regenerates the images after passing through the model.

```
IS : 1.788653
FID : 0.635207
```

ECE175B Final

June 14, 2023

0.1 Imports

[1]:
```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import numpy as np
from torchvision import datasets
from tqdm import tqdm
from torchvision.utils import save_image, make_grid
import torchvision.transforms as transforms

from ignite.metrics import FID, InceptionScore
from ignite.engine import Engine, Events
import os
```

0.2 Hyperparameters

[2]:
```
path = "chest_xray/"
DEVICE = torch.device("cuda")

batch_size = 32
image_size = 128
x_dim = image_size*image_size
latent_dim = 128
lr = 1e-3
epochs = 10
```

0.3 Load Data

[3]:
```
d_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Grayscale(),
    transforms.Resize((image_size, image_size))

])
```

```
# Load the datasets
train_dataset = datasets.ImageFolder(path+"train/", transform=d_transforms)
validation_dataset = datasets.ImageFolder(path+"val/", transform=d_transforms)
test_dataset = datasets.ImageFolder(path+"test/", transform=d_transforms)

# Make Dataloaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers = 8, prefetch_factor = 8, pin_memory=True,␣
 ↪persistent_workers=True)
val_loader = DataLoader(validation_dataset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers = 8, prefetch_factor = 8, pin_memory=True,␣
 ↪persistent_workers=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers = 8, prefetch_factor = 8, pin_memory=True,␣
 ↪persistent_workers=True)
```

0.4 Encoder

[4]:
```
class Encoder(nn.Module):
    def __init__(self,input_dim, latent_dim):
        super(Encoder,self).__init__()
        self.encConv1 = nn.Conv2d(1,16,5)
        self.encConv3 = nn.Conv2d(16,32,5)
        self.encConv4 = nn.Conv2d(32,64,5)

        self.FC_mean = nn.Linear(64*(image_size-12)**2, latent_dim)
        self.FC_var = nn.Linear(64*(image_size-12)**2, latent_dim)

        self.LeakyReLU = nn.LeakyReLU(0.2)


    def forward(self,x):

        x = self.LeakyReLU(self.encConv1(x))

        x = self.LeakyReLU(self.encConv3(x))
        x = self.LeakyReLU(self.encConv4(x))
        h_ = x.view(-1, 64*(image_size-12)**2)
        mean = self.FC_mean(h_)
        log_var = self.FC_var(h_)

        return mean, log_var
```

0.5 Decoder

[5]:
```
class Decoder(nn.Module):
    def __init__(self, latent_dim,output_dim):
        super(Decoder, self).__init__()

        self.FC_hidden = nn.Linear(latent_dim, 64*(image_size-12)**2)
        self.decConv2 = nn.ConvTranspose2d(64, 32, 5)
        self.decConv3 = nn.ConvTranspose2d(32, 16, 5)
        self.decConv5 = nn.ConvTranspose2d(16, 1, 5)
        self.LeakyReLU = nn.LeakyReLU(0.2)

    def forward(self,x):

        h = self.LeakyReLU(self.FC_hidden(x))
        x = h.view(-1,64,image_size-12,image_size-12)
        x = self.LeakyReLU(self.decConv2(x))
        x = self.LeakyReLU(self.decConv3(x))
        x = torch.sigmoid(self.decConv5(x))
        return x
```

0.6 Model

[6]:
```
class Model(nn.Module):
    def __init__(self,Encoder,Decoder):
        super(Model, self).__init__()
        self.Encoder = Encoder
        self.Decoder = Decoder

    def reparameterization(self, mean, var):
        epsilon = torch.randn_like(var).to(DEVICE)
        z = mean + var*epsilon
        return z

    def forward(self,x):
        mean, log_var = self.Encoder(x)
        z = self.reparameterization(mean, torch.exp(0.5*log_var))
        x_hat = self.Decoder(z)
        return x_hat, mean, log_var
```

0.7 Initializations

[7]:
```
encoder = Encoder(input_dim = x_dim, latent_dim = latent_dim)
decoder = Decoder(latent_dim = latent_dim, output_dim = x_dim)

model = Model(Encoder = encoder, Decoder = decoder).to(DEVICE)
```

0.8 Loss Functions

[8]:
```
from torch.optim import Adam

BCE_loss = nn.MSELoss()

def loss_function(x, x_hat, mean, log_var):
    reproduction_loss = nn.functional.binary_cross_entropy(x_hat,x,reduction =␣
 ↪'sum')
    KLD = -0.5*torch.sum(1+log_var - mean.pow(2) - log_var.exp())

    return reproduction_loss +KLD
```

0.9 Optimizer

[9]:
```
optimizer = Adam(model.parameters(),lr=lr)
```

0.10 Train

[10]:
```
print("Start training VAE.....")
model.train()
```

```
Start training VAE…
```

[10]:
```
Model(
  (Encoder): Encoder(
    (encConv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1))
    (encConv3): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
    (encConv4): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
    (FC_mean): Linear(in_features=861184, out_features=128, bias=True)
    (FC_var): Linear(in_features=861184, out_features=128, bias=True)
    (LeakyReLU): LeakyReLU(negative_slope=0.2)
  )
  (Decoder): Decoder(
    (FC_hidden): Linear(in_features=128, out_features=861184, bias=True)
    (decConv2): ConvTranspose2d(64, 32, kernel_size=(5, 5), stride=(1, 1))
    (decConv3): ConvTranspose2d(32, 16, kernel_size=(5, 5), stride=(1, 1))
    (decConv5): ConvTranspose2d(16, 1, kernel_size=(5, 5), stride=(1, 1))
    (LeakyReLU): LeakyReLU(negative_slope=0.2)
  )
```

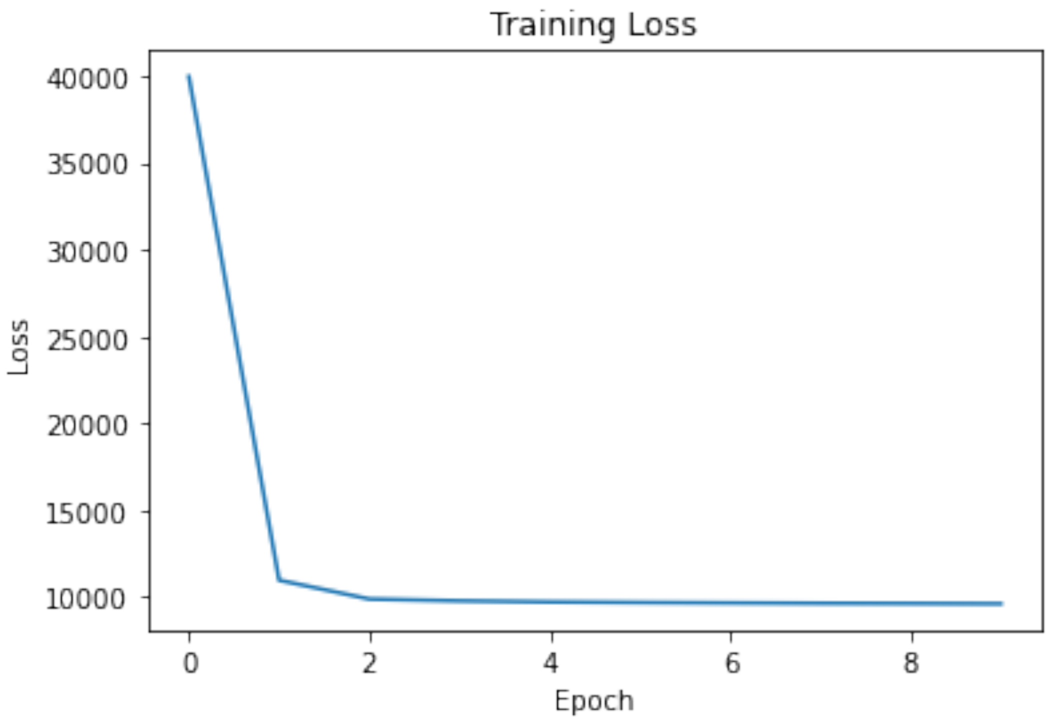
```
)
```

[11]:
```
loss_graph = []
for epoch in range(epochs):
    overall_loss = 0
    for batch_idx, (x,_) in enumerate(train_loader):
        optimizer.zero_grad()
        x = x.to(DEVICE)
        x_hat, mean, log_var = model(x)

        loss = loss_function(x,x_hat,mean,log_var)

        overall_loss +=loss.item()

        loss.backward()
        optimizer.step()

    loss_graph.append(overall_loss/(batch_idx*batch_size))
    print("\tEpoch",epoch+1, "\tloss", loss_graph[-1])

torch.save(model.state_dict(), os.path.join(os.getcwd(), "model.pth"))
```

```
        Epoch 1         loss 39971.37962360147
        Epoch 2         loss 10967.683370707948
        Epoch 3         loss 9882.601990499614
        Epoch 4         loss 9777.855842496141
        Epoch 5         loss 9715.958128375773
        Epoch 6         loss 9684.003442081405
        Epoch 7         loss 9659.468900704089
        Epoch 8         loss 9639.068799430941
        Epoch 9         loss 9628.653043016975
        Epoch 10        loss 9611.905002170139
```

0.11 Loss

[12]:
```
import matplotlib.pyplot as plt

plt.plot(loss_graph)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```



0.12 Test

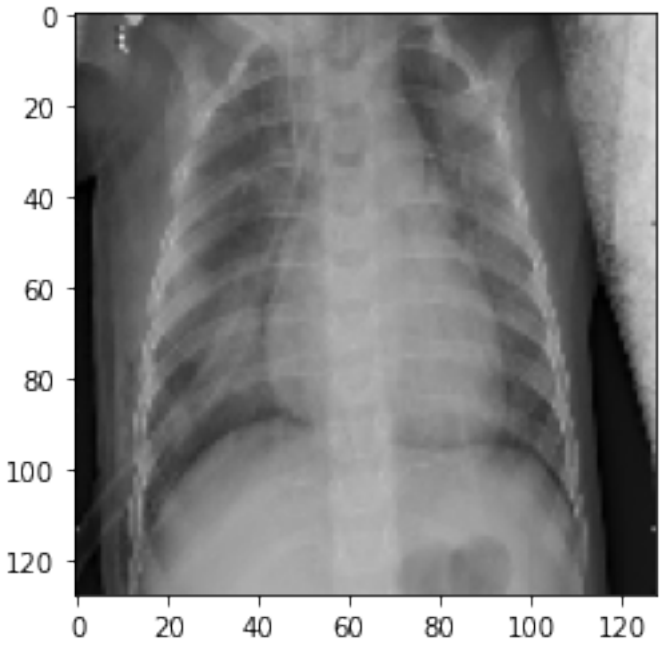
[13]:
```
with torch.no_grad():
    for batch_idx, (x,_) in enumerate(test_loader):
        x = x.to(DEVICE)

        x_hat, _, _ = model(x)

        break
```

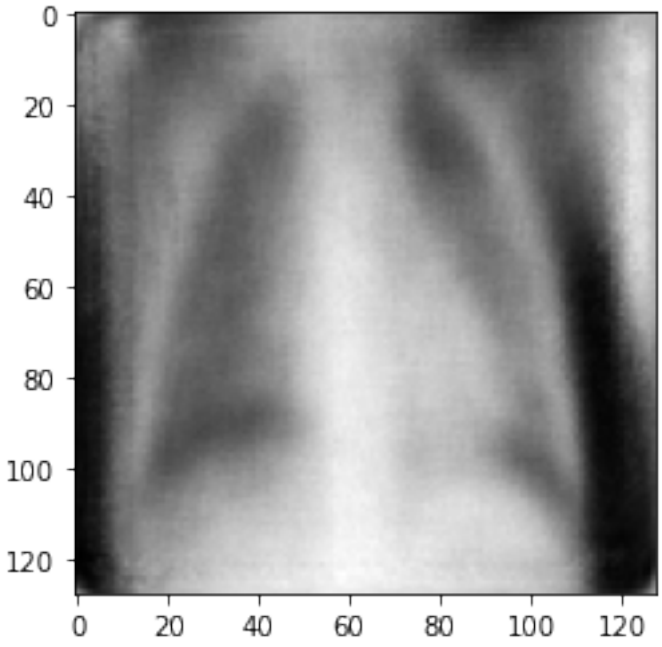
```
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
```

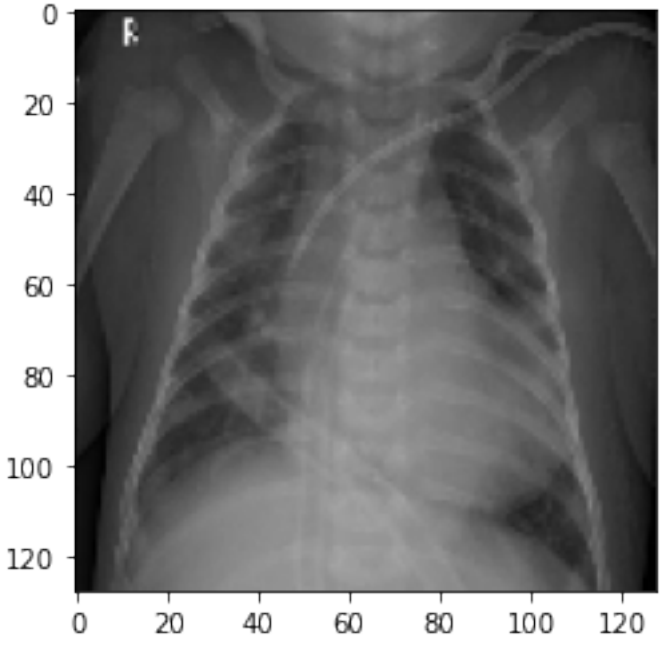
```
[W pthreadpool-cpp.cc:90] Warning: Leaking Caffe2 thread-pool after fork.
(function pthreadpool)
```

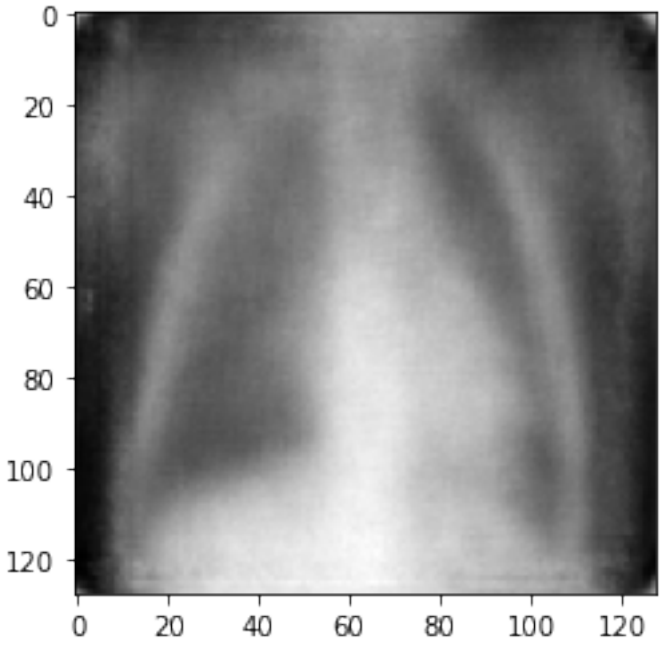
0.13 Display

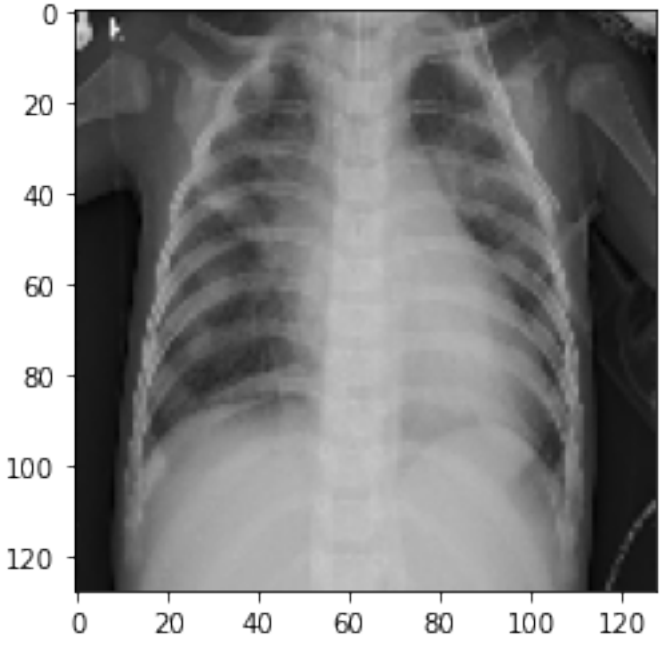
[14]:
```
def show_image(x,idx):
    x = x.view(batch_size,image_size,image_size)

    fig = plt.figure()
    plt.imshow(x[idx].cpu().numpy(), cmap='gray')

for i in range(3):
    show_image(x,idx=i)
    show_image(x_hat,idx=i)
```

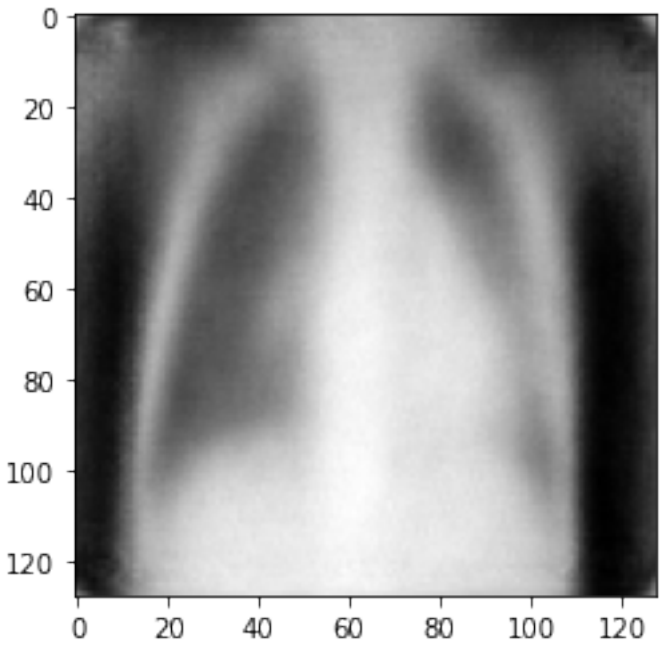



0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120



[15]:
```
model.load_state_dict(torch.load(os.path.join(os.getcwd(), "model.pth")))

FID_metric = FID(device=DEVICE)
IS_metric = InceptionScore(device=DEVICE, output_transform=lambda x: x[0])

def interpolate(batch):
    arr = []
    for img in batch:
        pil_img = transforms.ToPILImage()(img)
        resized_img = pil_img.convert('RGB')
        arr.append(transforms.ToTensor()(resized_img))
    return torch.stack(arr)


def evaluation_step(engine, batch):
    with torch.no_grad():
        model.eval()
        real = batch[0].to(DEVICE)
        fake = model(real)
        return fake[0].repeat(1,3,1,1), real.repeat(1,3,1,1)
```

[16]:
```
evaluator = Engine(evaluation_step)
FID_metric.attach(evaluator, "fid")
IS_metric.attach(evaluator, "is")
```

```
FID_values = []
IS_values = []

evaluator = Engine(evaluation_step)

IS_metric.attach(evaluator, "is")
FID_metric.attach(evaluator, "fid")


evaluator.run(test_loader)

metrics = evaluator.state.metrics

IS_score = metrics['is']
FID_score = metrics['fid']

IS_values.append(IS_score)
FID_values.append(FID_score)

print(f"IS : {IS_score:3f}")
print(f"FID : {FID_score:3f}")
```

```
/opt/conda/lib/python3.9/site-packages/torch/nn/functional.py:718: UserWarning:
Named tensors and all their associated APIs are an experimental feature and
subject to change. Please do not use them for anything important until they are
released as stable. (Triggered internally at
/pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)
```

```
IS : 1.788653
FID : 0.635207
```

[ ]: