BOĞAZIÇI UNIVERSITY

DEEP LEARNING
CMPE 597

Assignment 3

Author:
Y. Harun Kıvrıl
23 June 2021

Introduction

This assignment aims to implement a Variational Auto Encoder (VAE) on MNIST[5] dataset using a single LSTM layer as the decoder. For this purpose, a VAE model is implemented using Pytorch[7]. The model is trained on the trained validated and tested and results for losses, reconstruction and generation are presented with a discussion and conclusion at the end.

1 Network

The Variational Auto Encoders are consists of two parts. One is the part that takes the original image and outputs the parameters of the probability distributions which is called Encoder. The second one is the part that takes random numbers generated from the probability distributions of encoder and tries to construct the input image again which is called Decoder. In this assignment the distributions are selected as Gaussian. Therefore, encoder's expected output is mean and variance values. On the other hand the decoder will expect random numbers from the Gaussian Distribution.

1.1 Encoder

The assignment guidelines restricts the encoder start with a LSTM layer which takes the rows of the images in sequence. Therefore the shape of the data needed to be modified to proper dimension by squeezing the channel dimension. Since the image data is not a time sequence and order of sequence can be both considered from both bottom and top bidirectional LSTM is used.

At first as an encoder only LSTM layer is considered however, it is decided to add a fully connected layer after LSTM, since it helped to produce visually better results. No activation function is introduced after fully connected layers. The fully connected network considered to has two advantages, first it increases depth by one which seems help to obtain better encoded distribution to train the decoder, secondly it allows to use the same LSTM features for construction sigma and mu at the same time.

The LSTM layer is able to give hidden states for each element in the sequence however for this task a complete picture of the data is sufficient to keep the model simple and effective and this information lays at the final output of each direction. Therefore, the outputs of the final hidden state in each direction is used as input to fully connected layer.

At the end, the decoder is constructed as a bidirectional LSTM layer and a full connected for each distribution parameter, in this case for μ and σ^2. For the practical purposes the instead of σ^2 the output considered as $\log(\sigma^2)$. Using $\log(\sigma^2)$ creates stability and ease of training since it maps very small sigma values to larger domain and enforces σ^2 to be positive at the same time.

For the network parameters, different values are tried and the reconstructed and generated examples are investigated. As a result 64 is selected as hidden size, which is doubled with the other direction. And the fully connected layers are used to reduce the number of dimensions to 64 again.

1.2 Reparametrization

In VAE the output of the decoder is just parameters and in order to continue with the decoding, it is needed to generate random numbers using these parameters. This process creates problems for the back propagation algorithm and a trick called reparametrization is required to overcome it. Before the reparametrization the latent space goes into the encoder $l \sim N(\mu, \sigma^2)$ which is a random number and it is not possible to take the derivative. When, it is reparametrized it becomes $l = \mu + \sigma\varepsilon$ where $\varepsilon \sim N(0, I)$. In this equation derivative with respect to mu and sigma becomes available and the network can be trained.

1.3 Decoder

As a decoder at first only transposed convolutional layers are used however adding a fully connected layer with ReLU activation at the beginning gave better visual result for both reconstruction and generation and it is decided to keep this layer in the model. Also in here, fully connected layer allowed to control the number of features that enters to the transposed convolutional layers.
The output of the fully connected layer is considered as 1x1 pixel with many channels and it shaped into 28x28 output through 4 transposed convolutional layers with ReLu activation functions for the first 3 and sigmoid at the end. The control of the output size for the given kernel size, stride and padding is made by the equations of transposed convolution arithmetic. After several trials with visual control, the architecture of the variational encoder is finalized in the following form:

```
VAE(
  (encoder_lstm): LSTM(28, 64, batch_first=True, bidirectional=True)
  (encoder_mu): Linear(in_features=128, out_features=64, bias=True)
  (encoder_logvar): Linear(in_features=128, out_features=64, bias=True)
  (decoder_fc): Linear(in_features=64, out_features=64, bias=True)
  (decoder_list): ModuleList(
    (0): ConvTranspose2d(64, 64, kernel_size=(4, 4), stride=(1, 1))
    (1): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(1, 1))
    (2): ConvTranspose2d(32, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): ConvTranspose2d(16, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  )
)
```

Figure 1: Final Architecture

1.4 Loss Function

VAE's loss function is consists of two terms. First one is the reconstruction loss. This loss measures how far away the decoded picture from the encoded one. Assuming the pixels takes values between

0 and 1. Binary cross entropy is a proper choice for this task. The binary cross entropy is defined in the following form:

$$Loss_{BCE} = \sum_i [y_i \log x_i + (1 - y_i) \log(1 - x_i)]$$

The other term is a regularization term and it regularizes the latent distribution through a given distribution. In this case the given distribution is standard Gaussian distribution. Having standard Gaussian as the latent distribution allows to have a generative process after training process because the values coming from this distribution creates a latent space and this new latent space allows to generate new images.
The enforcement to standard Gaussian is made by using KL divergence [4] as the regularization term. KL divergence measure the difference of a probability distribution with respect to another one. Therefore in VAE it is expected to have less loss if the latent distribution from encoder is similar to the standard Gaussian.

When the KL divergence between to n dimensional multivariate Gaussian distributions is considered to construct the regularization term:

$$D_{KL}[q(z|x) \mid\mid p(z)] = \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - n + tr\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1)\right]$$

In VAE following setting is considered:
$p_1 = q(z|x)$ and $p_2 = p(z)$, so $\mu_1 = \mu$, $\Sigma_1 = \Sigma$, $\mu_2 = \vec{0}$, $\Sigma_2 = I$ than,

$$= \frac{1}{2}\left[\log\frac{|I|}{|\Sigma|} - n + tr\{I^{-1}\Sigma\} + (\vec{0} - \mu)^T I^{-1}(\vec{0} - \mu)\right]$$

$$= \frac{1}{2}\left[-\log|\Sigma| - n + tr\{\Sigma\} + \mu^T \mu\right]$$

$$= \frac{1}{2}\left[-\log\prod_i \sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right]$$

$$= \frac{1}{2}\left[-\sum_i \log\sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right]$$

$$= \frac{1}{2}\left[-\sum_i \left(\log\sigma_i^2 + 1\right) + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right]$$

[9]

Since we use logarithm of the variance the equations becomes has the following form in the implementation:

$$Loss_{KL} = -\frac{1}{2}\sum_i \left[\log\sigma_i^2 + 1 + e^{\log\sigma_i^2} + \mu_i^2\right]$$

Adding this component to the BCE loss, the final loss is obtained:

$$Loss_{final} = Loss_{BCE} + Loss_{KL}$$

2 Training

For the training, the data is spitted into train, validation and test sets. For test default test is used and for validation 10% of train data is randomly selected. Then with the following configuration the model is trained for max 50 epochs using Adam with $lr = 0.001$. As batch size 64 is used. The training is stopped if the validation total loss is not improved for 10 epochs. During the training the model with best total loss is saved.

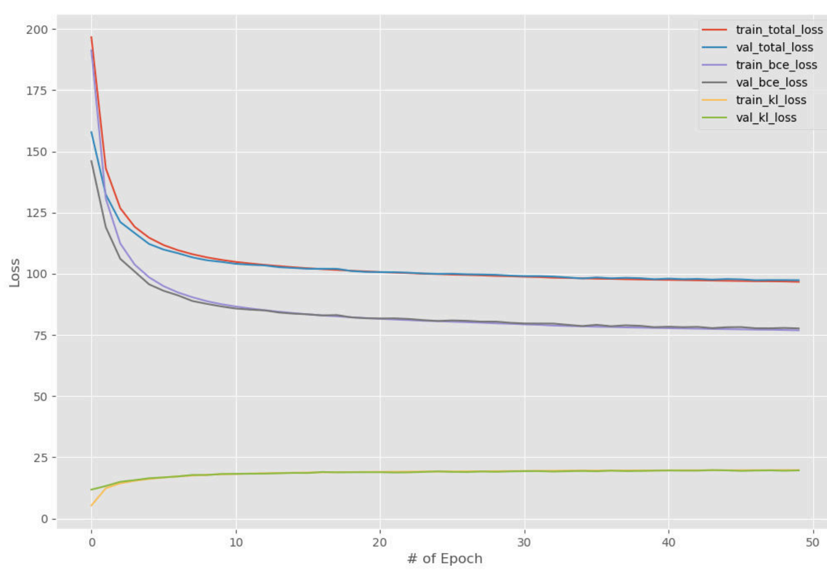

Figure 2: Loss values obtained during training

The loss plot shows that KL loss starts with a low values and it increases to a point where it almost stays constant. While KL increases, a large amount of decrease in BCE loss is observed. This situation occurs because the initial weights generates parameters closer parameters to standard Gaussian however the data is not very likely to represented as standard Gaussian and it favors BCE loss since to minimize the total loss.

$$\textit{Train Loss}_{Total} = 96.71$$

$$\textit{Train Loss}_{BCE} = 76.97$$

$$\textit{Train Loss}_{KL} = 19.74$$

Figure 3: A sample of images constructed from train data

$$\textit{Validation Loss}_{Total} = 97.35$$

$$\textit{Validation Loss}_{BCE} = 77.73$$

$$\textit{Validation Loss}_{KL} = 19.62$$

3 Test Results

After the training, the model is tested on the test data and following loss values are obtained.

$$Loss_{Total} = 97.57$$

$$Loss_{BCE} = 78.04$$

$$Loss_{KL} = 19.53$$

Also a regenerated sample from the test set is given below:

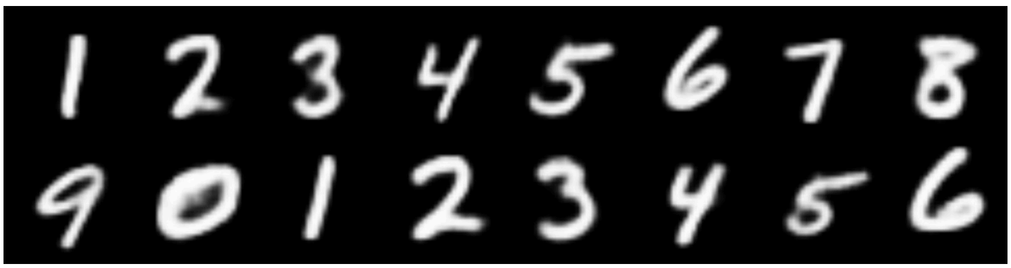

Figure 4: A sample of images constructed from test data

4 Generating New Digits

Generating digits are made by creating a random latent space by sampling from standard Gaussian distribution and passing that latent space to the decoder. The image below shows 100 generated digits.

Figure 5: Generated Images

5 Discussion & Conclusion

In this project a VAE with a single layer LSTM is implemented, trained and tested. The plot of the training losses showed that network is trained since the total loss decreases through epochs and it gets a balance between KL loss and BCE loss. Also the test set gives very similar loss values which means overfitting didn't occur. When the reconstructed images of both train end test is examined, model seems to construct aesthetic digits except for a few glitches. On the other hand, the generated images are less aesthetic but almost all of them are recognizable with the human eye. This is expected since the KL loss is not zero so the outputs of the encoder does not exactly standard Gaussian and the random numbers that is used are from standard Normal. In order to get better generation from the decoder, the KL loss should be decreased. This can be done by giving more weight to the KL loss or changing the encoder structure. Especially convolutional encoders for this task seems to work much better when the different studies are investigated. These two can be considered as future work of this study.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[5] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[6] Michael A. Nielsen. Neural networks and deep learning, 2018.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Neo Reo. Beginner guide to variational autoencoders (vae) with pytorch lightning. *Towards Data Science*, 05.04.2021. https://towardsdatascience.com/beginner-guide-to-variational-autoencoders-vae-with-pytorch-lightning-13dbc559ba4b.

[9] user3658307 (https://stats.stackexchange.com/users/128284/user3658307). Deriving the kl divergence loss for vaes. Cross Validated. URL:https://stats.stackexchange.com/q/370048 (version: 2020-03-03).

6 Appendix

6.1 Appendix A (Codes)

main.py

```
import json
import os

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import sampler
from torchvision.utils import make_grid
from tqdm import tqdm

from model import VAE

config = {
    "zdim":64,
    "image_size":28,
    "bidirect":True,
    "fc_out_size":64,
    "channels":(64, 32, 16, 1),
    "kernel_sizes":(4,4,4,4),
    "pads":(0,0,1,1),
    "strides":(1,1,2,2),
    "max_epoch":50,
    "early_stop_steps":10,
    "batch_size":64,
    "lr":1e-3
}

save_folder = "./models/deneme/"
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

MAX_EPOCH = config["max_epoch"]
early_stop_steps = config["early_stop_steps"]
batch_size= config["batch_size"]
lr = config["lr"]


os.makedirs(save_folder, exist_ok=True)
with open(save_folder + "config.json", "w") as file:
    json.dump(config, file)

transformations = transforms.Compose([transforms.ToTensor()])
resizer = transforms.Resize(800)

trainset = torchvision.datasets.MNIST(root='./data', train=True,
```

```
                                      download=True, transform=transformations)

valset = torchvision.datasets.MNIST(root='./data', train=True,
                                     download=True, transform=transformations)
testset = torchvision.datasets.MNIST(root='./data', train=False,
                                     download=True, transform=transformations)

split_ratio = 0.1
seed = 3136
n_obs = len(trainset)
indices = list(range(n_obs))
split_idx = int(np.floor(split_ratio * n_obs))
np.random.seed(seed)
np.random.shuffle(indices)

train_idx, valid_idx = indices[split_idx:], indices[:split_idx]
train_sampler = sampler.SubsetRandomSampler(train_idx)
valid_sampler = sampler.SubsetRandomSampler(valid_idx)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, sampler=train_sampler)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, sampler=valid_sampler)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)


model = VAE(**config).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = lr)
criterion = nn.BCELoss(reduction='sum')

def loss_fn(bce_loss, mu, logvar, gamma=0):
    kldiv = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return bce_loss + (1+gamma)*kldiv

print(model)

val_total_losses = []
val_bce_losses = []
train_total_losses = []
train_bce_losses = []

best_val_loss = 1000
early_stop_count = 0

for epoch in range(MAX_EPOCH):
    train_total_loss = 0
    train_bce_loss = 0
    for images, __ in tqdm(trainloader):
        images = images.to(device)
        images_in = images.squeeze(dim=1)
        out, mu, logvar = model(images_in)
        bce_loss = criterion(out, images)
        loss = loss_fn(bce_loss, mu, logvar)
        optimizer.zero_grad()
```

```
        loss.backward()
        optimizer.step()
        train_total_loss += loss.item()
        train_bce_loss += bce_loss.item()

    val_total_loss = 0
    val_bce_loss = 0
    print(f"Validating epoch: {epoch}")
    with torch.no_grad():
        for val_images, __ in tqdm(valloader):
            val_images = val_images.to(device)
            val_images_in = val_images.squeeze(dim=1)
            val_out, val_mu, val_logvar = model(val_images_in)
            val_bce = criterion(val_out, val_images)
            vloss = loss_fn(val_bce, val_mu, val_logvar)
            val_total_loss += vloss.item()
            val_bce_loss += val_bce.item()

    images = resizer(make_grid(val_out)).permute(1, 2, 0).cpu().numpy()
    plt.imsave(save_folder + f"epoch{epoch}.png", images)

    train_total_loss = train_total_loss/(len(trainloader)*batch_size)
    train_total_losses.append(train_total_loss)

    train_bce_loss = train_bce_loss/(len(trainloader)*batch_size)
    train_bce_losses.append(train_bce_loss)

    val_total_loss = val_total_loss/(len(valloader)*batch_size)
    val_total_losses.append(val_total_loss)

    val_bce_loss = val_bce_loss/(len(valloader)*batch_size)
    val_bce_losses.append(val_bce_loss)

    if val_total_loss < best_val_loss:
        early_stop_count=0
        best_val_loss = val_total_loss
        torch.save(model.state_dict(), save_folder + "best_model.pkl")
    else:
        early_stop_count+=1

    print(f"""
    Epoch {epoch}: Train Total Loss: {train_total_loss}, Val Total Loss: {val_total_loss}
    Epoch {epoch}: Train BCE Loss: {train_bce_loss}, Val BCE Loss: {val_bce_loss}
    """)

train_kl_losses = np.array(train_total_losses) - train_bce_losses
val_kl_losses = np.array(val_total_losses) - val_bce_losses

plt.style.use("ggplot")
plt.figure(figsize=(12,8))
plt.plot(list(range(len(train_total_losses))), train_total_losses, label='train_total_loss')
plt.plot(list(range(len(val_total_losses))), val_total_losses, label='val_total_loss')
```

```python
plt.plot(list(range(len(train_bce_losses))), train_bce_losses, label='train_bce_loss')
plt.plot(list(range(len(val_bce_losses))), val_bce_losses, label='val_bce_loss')
plt.plot(list(range(len(train_kl_losses))), train_kl_losses, label='train_kl_loss')
plt.plot(list(range(len(val_kl_losses))), val_kl_losses, label='val_kl_loss')
plt.xlabel("# of Epoch")
plt.ylabel("Loss")
plt.legend()
plt.savefig(f"{save_folder}/loss_plot.png")

test_total_loss = 0
test_bce_loss = 0
for images, __ in tqdm(testloader):
    images = images.to(device)
    images_in = images.squeeze(dim=1)
    out, mu, logvar = model(images_in)
    bce_loss = criterion(out, images)
    loss = loss_fn(bce_loss, mu, logvar)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    test_total_loss += loss.item()
    test_bce_loss += bce_loss.item()

test_results ={}
test_results["total_loss"] = test_total_loss/(len(testloader)*batch_size)
test_results["bce_loss"] = test_bce_loss/(len(testloader)*batch_size)
test_results["kl_loss"] = test_results["total_loss"] - test_results["bce_loss"]

images = resizer(make_grid(out)).permute(1, 2, 0).cpu().numpy()
plt.imsave(save_folder + f"test_sample.png", images)

with open(f"{save_folder}/test_results.json", "w") as file:
    json.dump(test_results, file)
```

model.py

```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class VAE(nn.Module):

    def __init__(self,
                 zdim=64,
                 image_size=28,
                 bidirect=True,
                 channels=(64, 32, 16, 1),
                 kernel_sizes=(4,4,4,4),
                 pads=(0,0,1,1),
                 strides=(1,1,2,2),
                 encoder_fc_out_size=64,
                 decoder_fc_out_size=64,
                 **kwargs):

```

```
        super().__init__()
        self.zdim = zdim
        self.image_size = image_size
        self.kernel_sizes = kernel_sizes
        self.pads = pads
        self.strides = strides
        self.n_decoder_layers = len(channels)
        self.fc_in = 2*zdim if bidirect else zdim
        self.encoder_fc_out_size = encoder_fc_out_size
        self.decoder_fc_out_size = decoder_fc_out_size
        self.bidirect = bidirect

        self.encoder_lstm = nn.LSTM(image_size, zdim, bidirectional=bidirect,
            batch_first=True)
        self.encoder_mu = nn.Linear(self.fc_in, encoder_fc_out_size)
        self.encoder_logvar = nn.Linear(self.fc_in, encoder_fc_out_size)

        self.decoder_fc = nn.Linear(encoder_fc_out_size, decoder_fc_out_size)
        self.decoder_list = nn.ModuleList([])
        for i, kernel in enumerate(kernel_sizes):
            in_ch = channels[i-1] if i>0 else self.decoder_fc_out_size
            out_ch = channels[i]
            self.decoder_list.append(
                nn.ConvTranspose2d(in_ch, out_ch, kernel, strides[i], pads[i])
            )

        self.check_out_dim()

    def encode(self, x):
        _, (lstm_out, _) = self.encoder_lstm(x)
        if self.bidirect:
            lstm_out_dir0 = lstm_out[0].view(-1, self.zdim)
            lstm_out_dir1 = lstm_out[1].view(-1, self.zdim)
            lstm_out = torch.cat([lstm_out_dir0, lstm_out_dir1], axis=1)
        else:
            lstm_out = lstm_out.view(-1, self.fc_in)
        mu = self.encoder_mu(lstm_out)
        logvar = self.encoder_logvar(lstm_out)
        return mu, logvar

    def reparametrize(self, mu, logvar):
        sigma = torch.exp(0.5*logvar)
        z = torch.randn_like(mu)
        return mu + z*sigma

    def decode(self, x):
        x = self.decoder_fc(x)
        x = F.relu(x)
        x = x.view(-1, self.decoder_fc_out_size, 1 , 1)

        for i, layer in enumerate(self.decoder_list):
            x = layer(x)
            if i+1 < self.n_decoder_layers:
                x = F.relu(x)
```

```python
            else:
                x = torch.sigmoid(x)

        return x


    def forward(self, x):
        mu, logvar = self.encode(x)
        sample = self.reparametrize(mu, logvar)
        decoded = self.decode(sample)
        return decoded, mu, logvar

    def check_out_dim(self):
        out = 1
        for kernel_size, pad, stride in zip(self.kernel_sizes, self.pads, self.strides):
            out = stride*(out-1) + kernel_size - 2*pad #+ (out+2*pad-kernel_size)%stride

        print(f"Output Dim: {out}" )
        assert out == self.image_size
```

generator.py

```python
from model import VAE
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision.utils import make_grid
from torchvision.transforms import Resize

config = {
    "zdim":64,
    "image_size":28,
    "bidirect":True,
    "fc_out_size":64,
    "channels":(64, 32, 16, 1),
    "kernel_sizes":(4,4,4,4),
    "pads":(0,0,1,1),
    "strides":(1,1,2,2),
    "max_epoch":15,
    "early_stop_steps":10,
    "batch_size":64,
    "lr":1e-3,
}

model = VAE(**config)
model.load_state_dict(torch.load("./best_model.pkl"))

resizer = Resize(800)
latent = torch.Tensor(np.ones((100, 64)))
latent = torch.randn_like(latent)
decoded = model.decode(latent)
images =resizer(make_grid(decoded, nrow=10)).permute(1, 2, 0).numpy()
plt.figure(figsize=(10,10))
plt.imsave("./generated_images.png", images)
print("Generated images saved to ./generated_images.png")
```