Deep Learning

CS-898BD

Yassine Ibork

Y723u998

Assignment 4

CS89BD Deep Learning, Fall 2024

Supervised by: Dr. Lokesh Das

I. Introduction

This assignment aims to provide us with an understanding of unsupervised learning through generative models. It uses a dataset to generate images without relying on image labels as input. In this assignment, we will create two different architectures: VAE and DCGAN. Additionally, we will compare the SSIM and MSE of the two architectures. The code for the VAE is from [1] while the code for the DCGAN is from [2]. The code for the assignment can be found in this GitHub repository: https://github.com/yibork/DL-VAE-DCGAN.

II. Methodology

The dataset used for this assignment is the CIFAR-10 dataset. We have trained two different architectures: the Variational Autoencoder (VAE) and the Deep Convolutional Generative Adversarial Network (DCGAN). In this section, I will outline the approach I used for each architecture.

2.1. VAE

For training the Variational Autoencoder (VAE), we implemented the following approach:

- **Dataset Preparation**:
  The CIFAR-10 dataset was downloaded and split into training and validation sets with sizes of 50,000 and 10,000 images, respectively. Data transformations included resizing, center cropping, tensor conversion, and normalization to ensure consistent input dimensions and values.

- **Model Training**:
  We set the training parameters, including 500 epochs, a batch size of 32, and a logging interval of 10.

- **Loss Function for VAE**:
  For the VAE, a custom loss function combining Binary Cross Entropy (BCE) and Kullback-Leibler Divergence (KLD) was implemented. BCE measured the reconstruction error, while KLD was used to regularize the latent space.

- **Evaluation Metrics**:
  To compare the models' performance, we calculated the Structural Similarity Index (SSIM) and Mean Squared Error (MSE) between generated and original images, providing a quantitative assessment of image quality.

2.2. **DCGAN**

For training the DCGAN we have used the following approach:

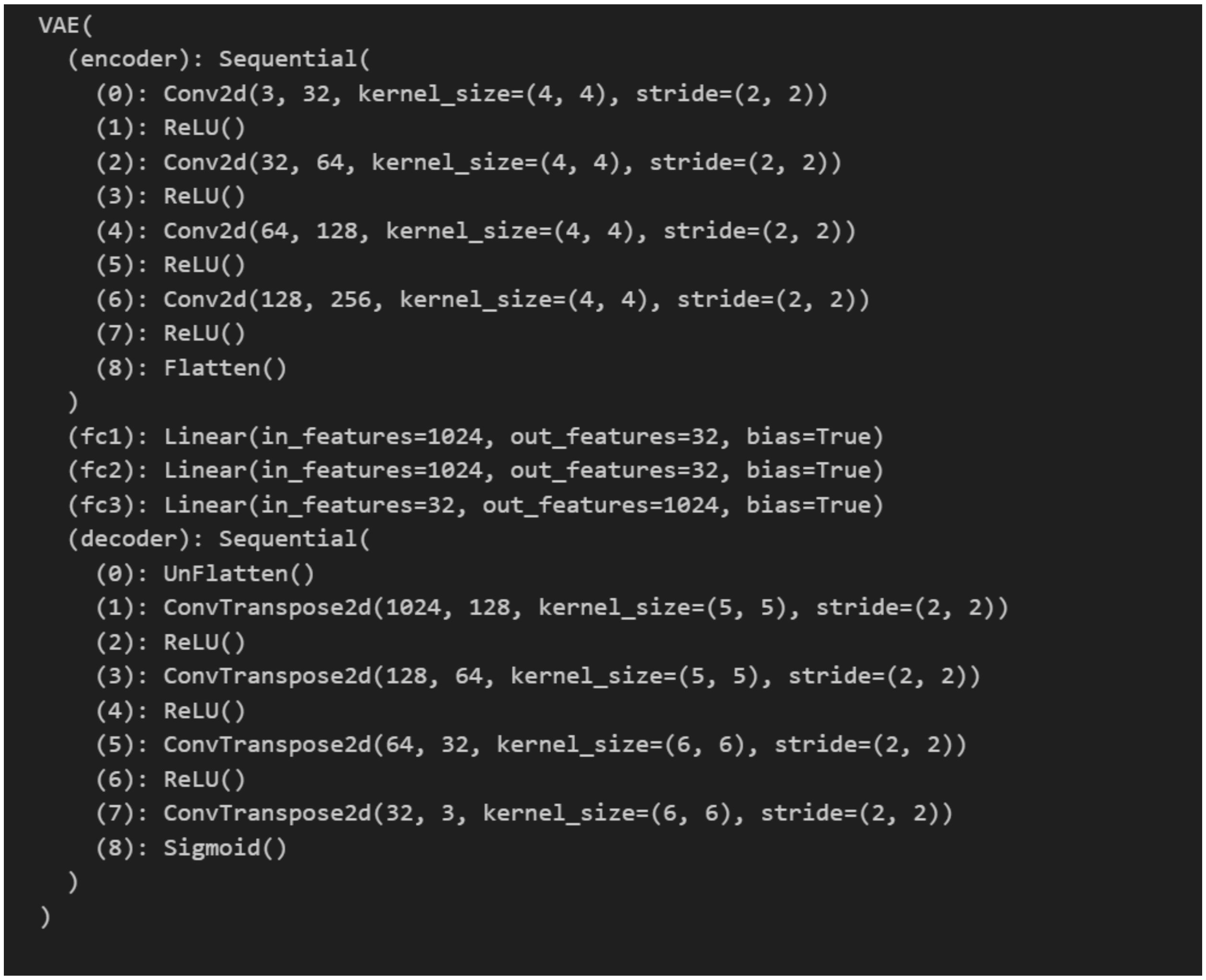
- **Dataset Preparation**:
  The CIFAR-10 dataset was used as the training data source, with all images resized to a spatial size of 64x64 pixels.
- **DCGAN Architecture Setup**:
  - **Generator (G)**: The generator's input is a latent vector (z) of size 100, which is transformed into feature maps of increasing dimensions to produce a synthetic 64x64 color image.
  - **Discriminator (D)**: The discriminator is designed to classify real vs. fake images.
- **Hyperparameters**:
  - **Training Parameters**: The model was trained for 200 epochs with a batch size of 128.
  - **Learning Rate**: A learning rate of 0.0002 was used for both the generator and discriminator.
  - **Adam Optimizer**: The Adam optimizer was applied with a beta1 parameter of 0.5, which helps stabilize training by reducing oscillations.
- **Weight Initialization**: For convolutional layers, weights were initialized with a normal distribution (mean=0, std=0.02), and for batch normalization layers, weights were initialized with mean=1, std=0.02, while biases were set to zero.
- **Training Procedure**: During training, the generator and discriminator compete in a min-max game. The generator attempts to produce realistic images that can fool the discriminator, while the discriminator tries to correctly classify real versus generated images.
- **Evaluation Metrics**: To assess the quality of generated images, evaluation metrics such as Structural Similarity Index (SSIM) and Mean Squared Error (MSE) were used.

III. Deep Learning Architecture

Since we have used two different architectures for this assignment. In this section I will give details about each architecture.

3.1. VAE

The first architecture is the Variational Autoencoder (VAE), which consists of an encoder and a decoder. The encoder's task is to encode the input image into a latent feature, which is then used by the decoder to generate an image. The adopted architecture is depicted in the figure below.



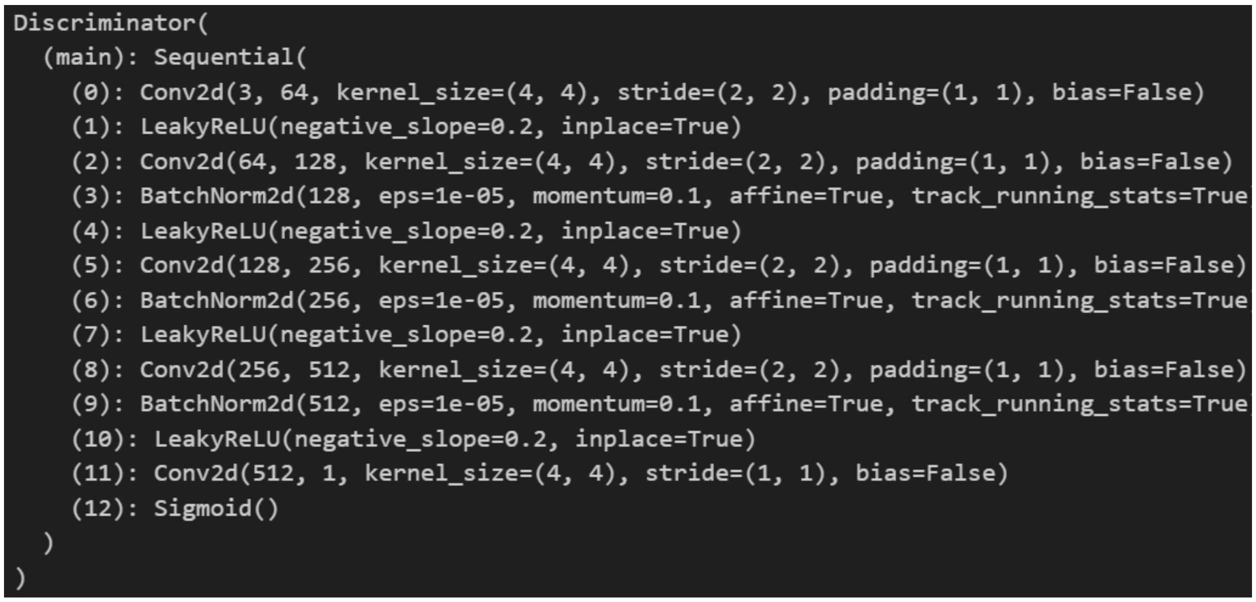
```
VAE(
  (encoder): Sequential(
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (3): ReLU()
    (4): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2))
    (5): ReLU()
    (6): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2))
    (7): ReLU()
    (8): Flatten()
  )
  (fc1): Linear(in_features=1024, out_features=32, bias=True)
  (fc2): Linear(in_features=1024, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=1024, bias=True)
  (decoder): Sequential(
    (0): UnFlatten()
    (1): ConvTranspose2d(1024, 128, kernel_size=(5, 5), stride=(2, 2))
    (2): ReLU()
    (3): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(2, 2))
    (4): ReLU()
    (5): ConvTranspose2d(64, 32, kernel_size=(6, 6), stride=(2, 2))
    (6): ReLU()
    (7): ConvTranspose2d(32, 3, kernel_size=(6, 6), stride=(2, 2))
    (8): Sigmoid()
  )
)
```

Figure 1. VAE Architecture

3.2. DCGAN

The second architecture we used is a DCGAN, which consists of two models that work together. The generator's job is to generate images from a latent space vector, while the discriminator's role is to determine whether the images generated by the generator are real or fake.

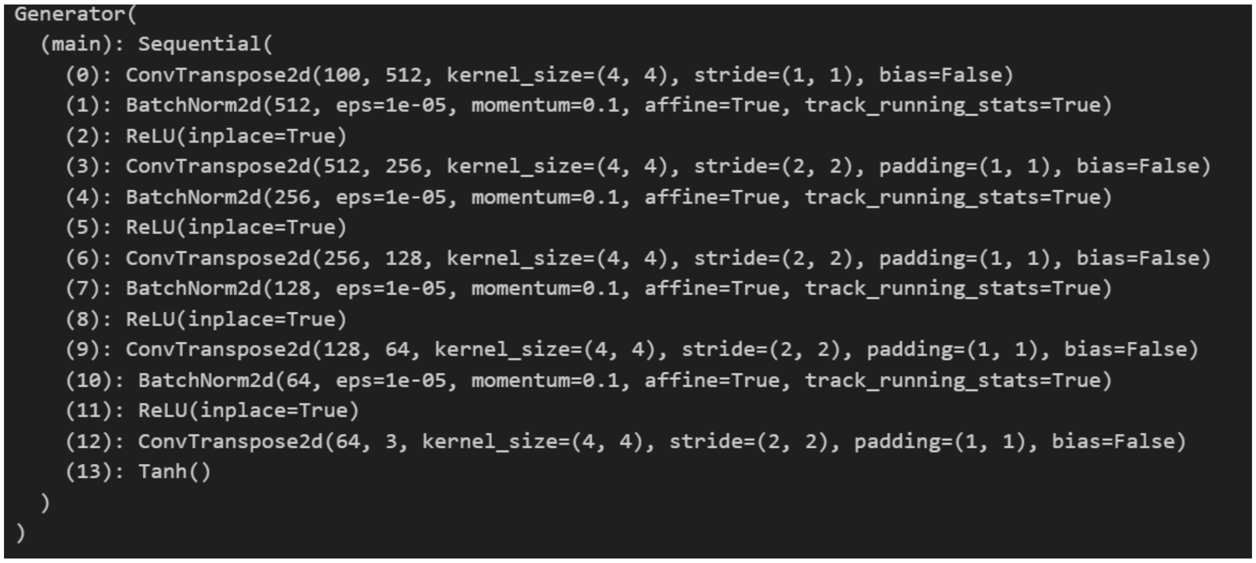
The discriminator takes an input image of shape 3x3x64 and produces either 0 or 1 using the sigmoid function, as shown in the figure below.



```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Fig 2. Discriminator Architecture

The Generator on the other hand takes a latent space as input and generates an images of shape 64x64x3 which will be passed to the generator as seen above to tell whether the image is reel or fake.



```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Fig 3. Generator Architecture

IV. Experiment and Results

In this section, we present the results and experiments for training two models: the Variational Autoencoder (VAE) and the Deep Convolutional Generative Adversarial Network (DCGAN) on the CIFAR-10 dataset. We will analyze and compare the performance of both models in terms of Mean Squared Error (MSE) and Structural Similarity Index Measure (SSIM), evaluating the quality of the reconstructed and generated images. This comparative analysis highlights the models' effectiveness in capturing and replicating the dataset's core features.

4.1. VAE

This section discusses the experiment setup and results for training the Variational Autoencoder (VAE) model on the CIFAR-10 dataset.

Mean Squared Error (MSE) Loss

- **Description**: The figure below illustrates the MSE loss of the VAE model over the course of training and evaluation.
- **Interpretation**: The MSE loss decreases over epochs, indicating that the model is learning to reconstruct images effectively. By the end of training, the model achieves an MSE value of 120, while the evaluation phase reaches an MSE value of 165. This difference suggests that the model has learned key patterns from the training data but shows a slight increase in error on unseen data, reflecting generalization limitations.

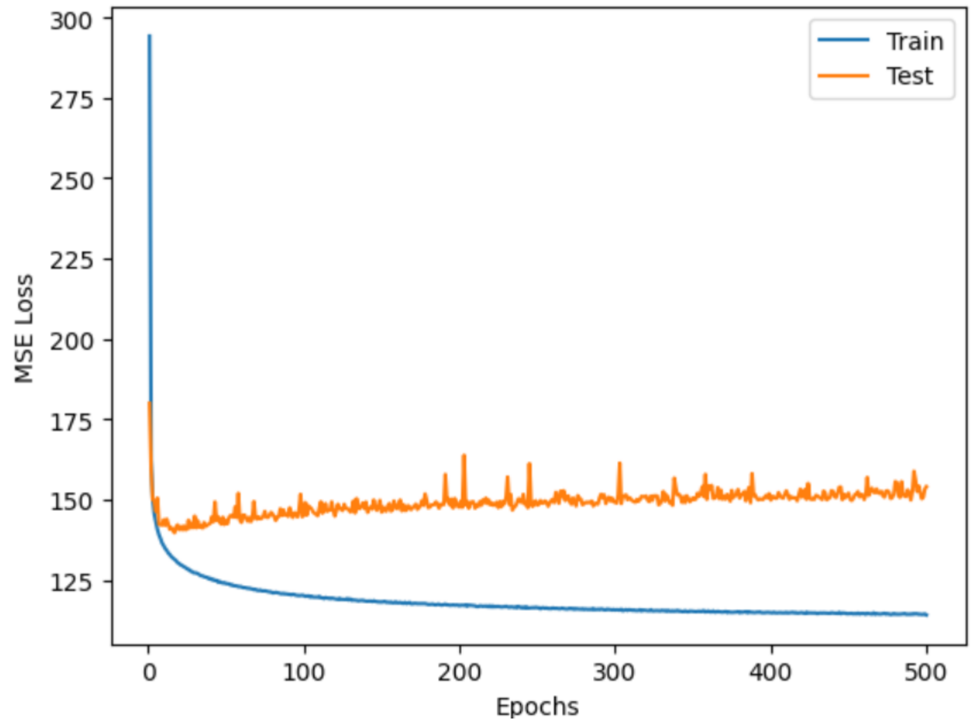


Fig 4. MSE Loss over epochs

Structural Similarity Index Measure (SSIM) Loss

- **Description**: The SSIM loss, shown in the figure below, tracks the model's structural accuracy in image reconstruction across epochs.
- **Interpretation**: Similar to the MSE, the SSIM metric follows a consistent trend over the training process. The model reaches an SSIM score of 0.004 during training and 0.0052 during testing, indicating some improvement in image structural similarity between generated and original images. However, the relatively low SSIM scores reflect the challenges of achieving high structural fidelity at this dataset resolution.

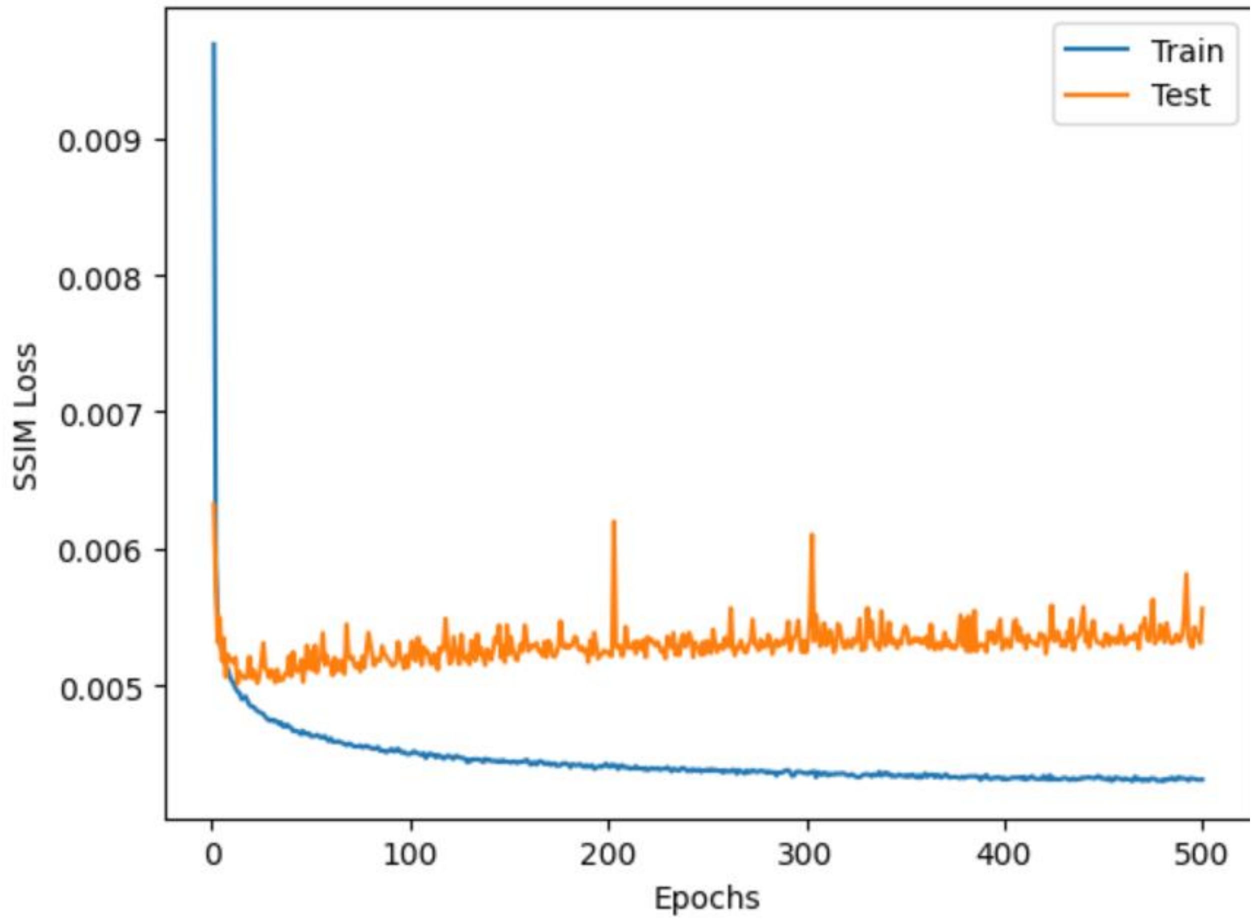


Fig 5. SSIM Loss over epochs

VAE Reconstructed Images

- **Description**: The following figure shows side-by-side comparisons of original images and their corresponding reconstructions by the VAE model.
- **Interpretation**: The reconstructed images demonstrate the VAE's ability to capture the basic structure and color patterns of the CIFAR-10 dataset. While some images appear blurred due to the model's limitations, key characteristics of the original images are still present, reflecting the model's general reconstruction capability.

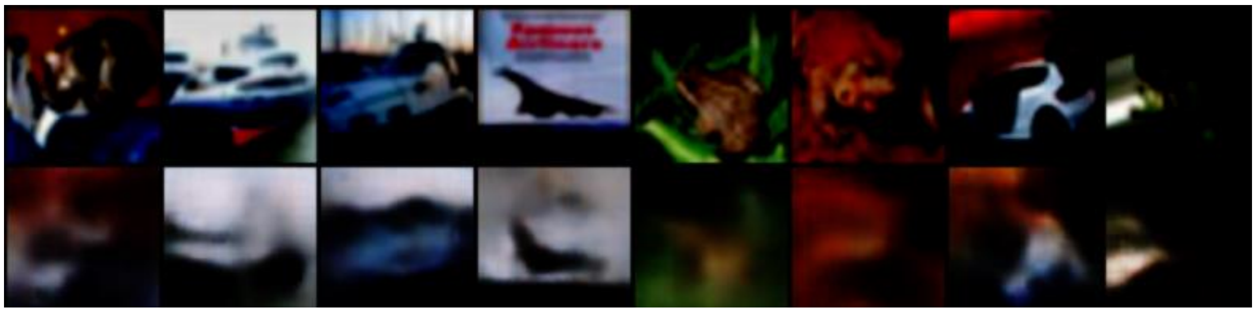


Fig 6. VAE Reconstructed Images

VAE Generated Images

- **Description**: This figure presents a selection of images generated by the VAE from random latent vectors.
- **Interpretation**: The generated images show mostly random pixel patterns, with limited recognizable structure. This lack of coherent image generation may be attributed to the low resolution of the CIFAR-10 dataset, which could limit the model's ability to learn detailed features. Further optimization may be required to improve image clarity.

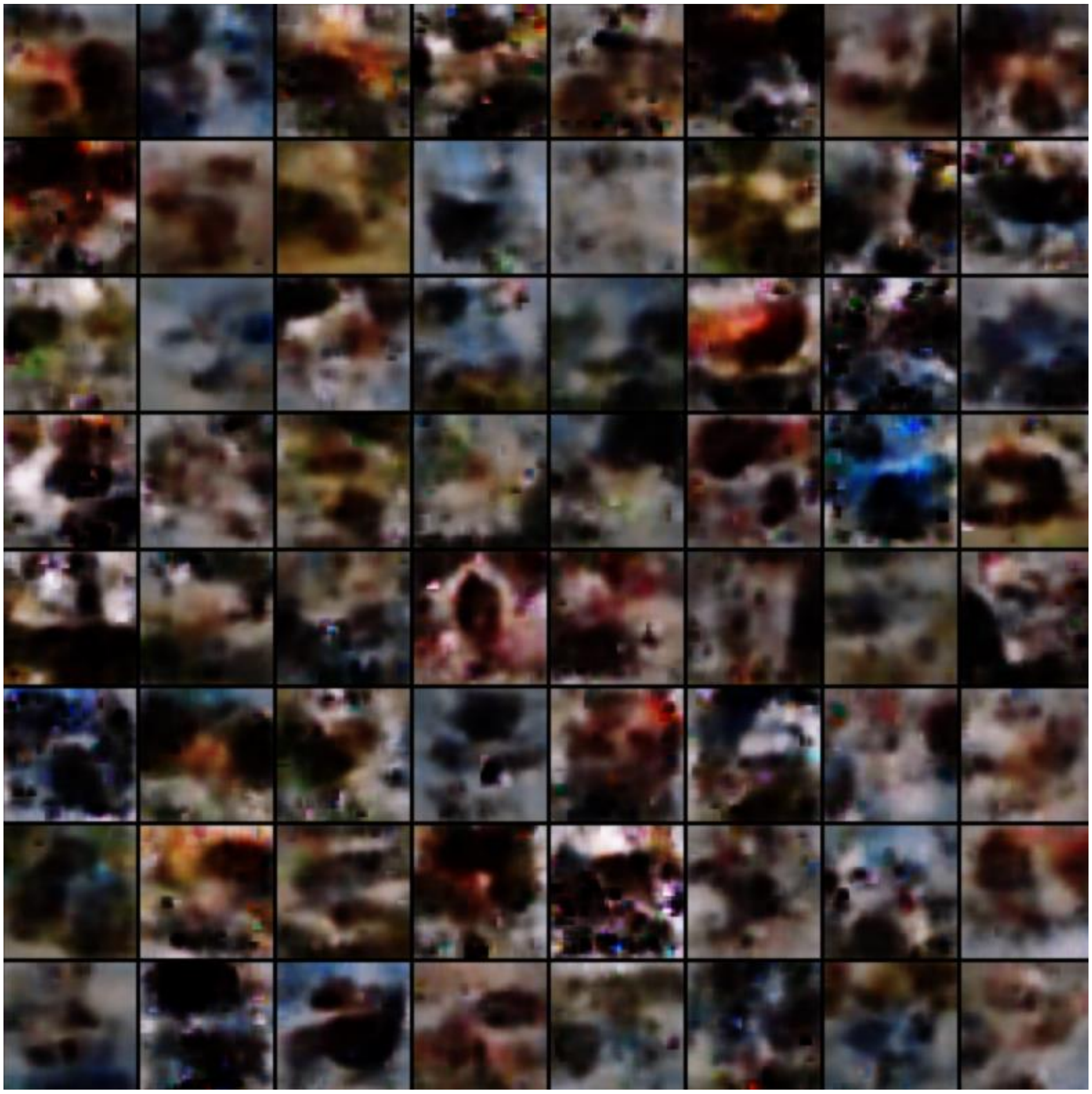


Fig 6. VAE Generated Images

4.2. DCGAN

In this section, we present the experimental setup and results for training the Deep Convolutional Generative Adversarial Network (DCGAN) model on the CIFAR-10 dataset. We examine the model's performance based on the Mean Squared Error (MSE) and Structural Similarity Index (SSIM) metrics. A comparison of these metrics between the DCGAN and Variational Autoencoder (VAE) models will also be provided to understand how each model reconstructs and generates images.

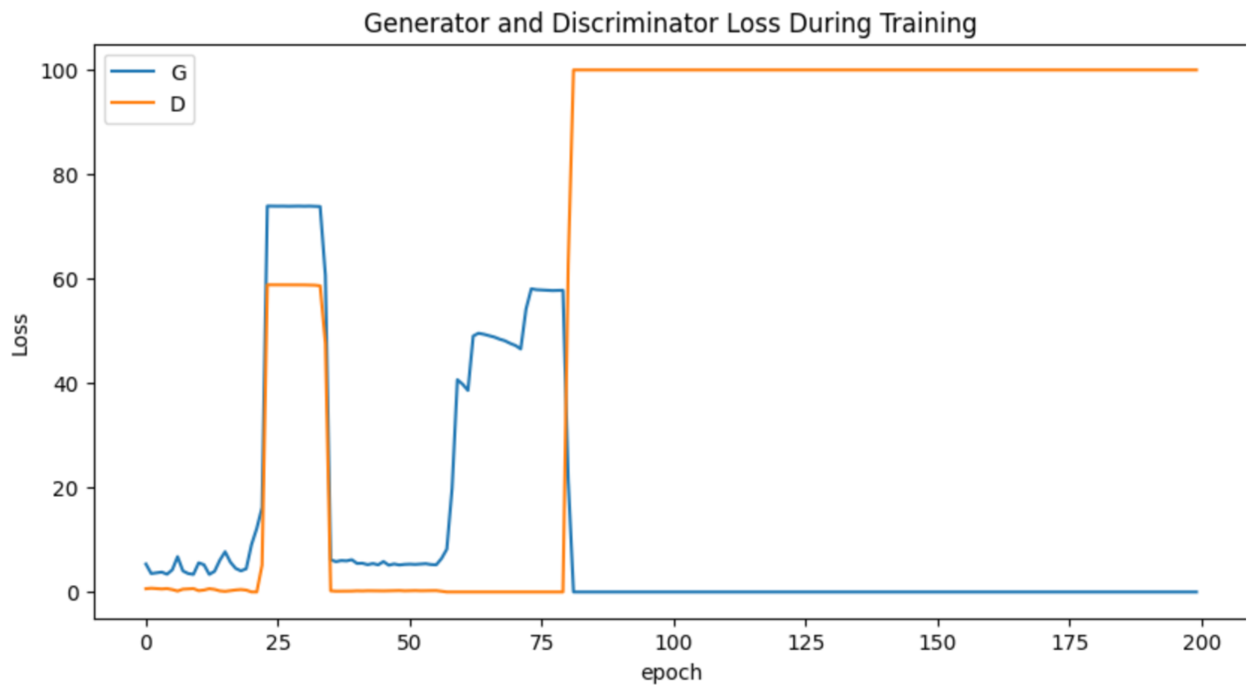


Fig. 7: Generator and Discriminator Loss During Training

This graph illustrates the loss values for both the generator (G) and discriminator (D) throughout the training process of the DCGAN. The generator's loss (in blue) and the discriminator's loss (in orange) show fluctuating trends, with the discriminator loss ultimately reaching a high plateau, while the generator stabilizes close to zero.

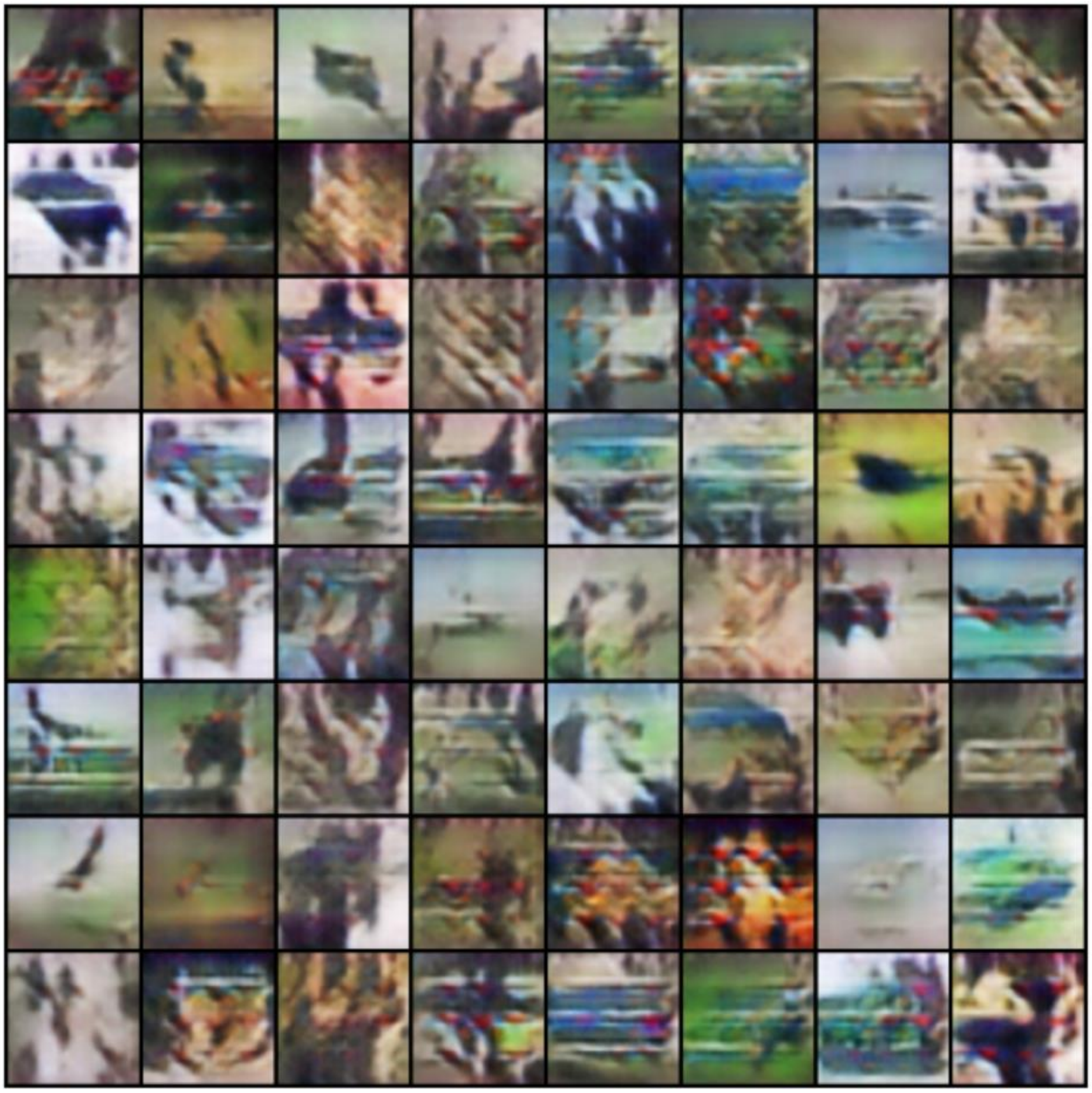
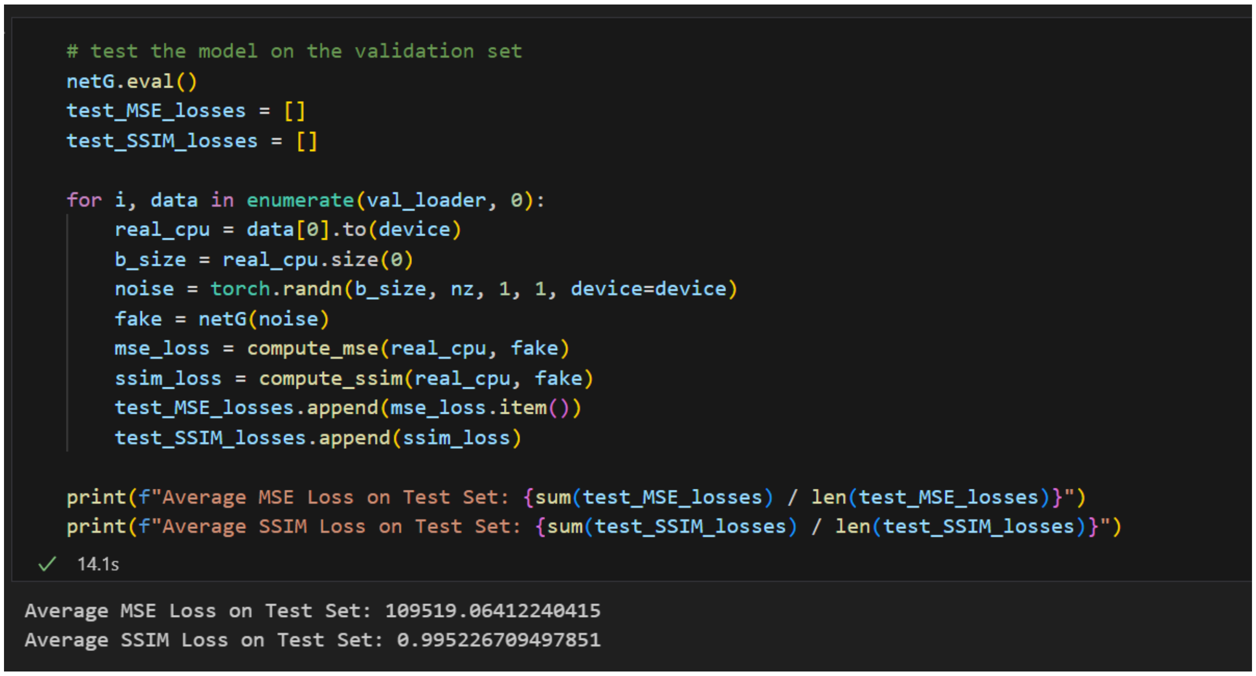


Fig 8: DCGAN Generated Images

This figure displays a grid of images generated by the DCGAN model after training. The images exhibit abstract patterns, with some vague shapes and colors resembling elements in CIFAR-10 classes, though they lack realistic details. The generated images suggest that the model has captured some low-level features, such as colors and textures, but struggles with fine details and structured forms. This limitation could be due to the resolution of the CIFAR-10 dataset and the model's training configuration.



```
# test the model on the validation set
netG.eval()
test_MSE_losses = []
test_SSIM_losses = []

for i, data in enumerate(val_loader, 0):
    real_cpu = data[0].to(device)
    b_size = real_cpu.size(0)
    noise = torch.randn(b_size, nz, 1, 1, device=device)
    fake = netG(noise)
    mse_loss = compute_mse(real_cpu, fake)
    ssim_loss = compute_ssim(real_cpu, fake)
    test_MSE_losses.append(mse_loss.item())
    test_SSIM_losses.append(ssim_loss)

print(f"Average MSE Loss on Test Set: {sum(test_MSE_losses) / len(test_MSE_losses)}")
print(f"Average SSIM Loss on Test Set: {sum(test_SSIM_losses) / len(test_SSIM_losses)}")
```

✓ 14.1s

```
Average MSE Loss on Test Set: 109519.06412240415
Average SSIM Loss on Test Set: 0.995226709497851
```

Fig. 9: Code Snippet and Test Results for MSE and SSIM on DCGAN

This code snippet demonstrates the evaluation phase for the DCGAN model on the validation set. During this phase, the model generates images from random noise, and the Mean Squared Error (MSE) and Structural Similarity Index (SSIM) metrics are computed between these generated images and the real images in the validation set. The model achieved an MSE value of 10,519 and an average SSIM of 0.999 on this test set, indicating the quality of the images generated in terms of pixel-wise accuracy and structural similarity.

In earlier experiments (refer to "Experiment 1" in the GitHub repository), I attempted to track the SSIM metric during the training phase as well. However, an unusual pattern emerged: the training SSIM score increased linearly over time, while the test SSIM score remained stable. This inconsistency suggested that the SSIM metric may not be providing reliable feedback during training. Consequently, in the second experiment, I opted to calculate and average the SSIM only on the test set, as shown in the figure below. This adjustment helps ensure a more accurate and consistent assessment of the model's structural similarity performance on unseen data.

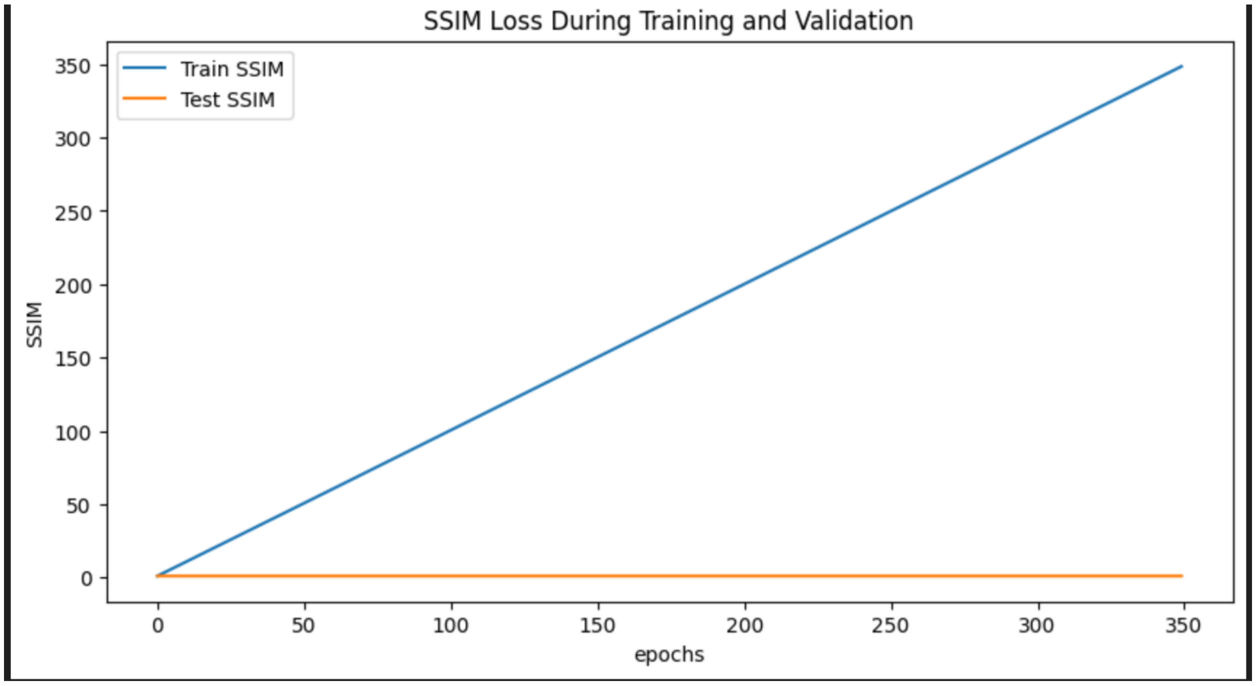


Fig 10.Experiment Train and Test ssim

4.3. Comparison of DCGAN and VAE

Quantitative analysis:

- **MSE:** VAE achieves a significantly lower MSE, indicating better pixel-wise accuracy compared to DCGAN.
- **SSIM:** DCGAN achieves a much higher SSIM, showing it captures structural similarity more effectively than VAE.

Metric	DCGAN	VAE
MSE	10,519 (test)	120 (train), 165 (test)
SSIM	0.999 (test)	0.004 (train), 0.0052 (test)

The VAE excels in pixel accuracy (low MSE) but struggles with structure (low SSIM), while the DCGAN captures structure well (high SSIM) but lacks pixel accuracy (high MSE).

Qualitative analysis:

- **VAE**: Generates less distinct images, often blurry with less structure. More stable during training
- **DCGAN**: Produces images with clearer structures and textures but is prone to instability and artifacts during training.

V. Conclusion

In summary, this assignment provided a comprehensive exploration of unsupervised learning via generative models by implementing and analyzing Variational Autoencoder (VAE) and Deep Convolutional Generative Adversarial Network (DCGAN) architectures on the CIFAR-10 dataset. The quantitative comparison of these models showed clear distinctions in performance:

- MSE Analysis: The VAE model demonstrated significantly lower Mean Squared Error (MSE) scores, underscoring its superior accuracy in pixel-wise reconstruction.
- SSIM Analysis: Despite VAE's low MSE, DCGAN achieved higher Structural Similarity Index (SSIM) values, indicating a stronger capacity to preserve structural features within images.

Overall, VAE showed stable training behavior with a focus on accurate pixel reproduction, while DCGAN excelled in capturing structural details but faced stability challenges. This assignment highlights the trade-offs between these generative models and their unique strengths in producing realistic images. The code and additional details can be found in the GitHub repository: https://github.com/yibork/DL-VAE-DCGAN

VI. Reference

[1] "pytorch-vae/vae.py at master · sksq96/pytorch-vae." Accessed: Nov. 08, 2024. [Online]. Available: https://github.com/sksq96/pytorch-vae/blob/master/vae.py

[2] "DCGAN Tutorial — PyTorch Tutorials 2.5.0+cu124 documentation." Accessed: Nov. 08, 2024. [Online]. Available: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html