IMAGE GENERATION AND UPSCALING USING GENERATIVE ADVERSIAL NETWORK

A MAJOR-PROJECT REPORT

Submitted by

SAKTHI MURUGAN V	(211520205122)
SARAN NITHISH NA	(211520205131)
SHYAM GANESH J	(211520205138)

in partial fulfilment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

PANIMALAR INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY : CHENNAI 600 025

MAY-2024

PANIMALAR INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"IMAGE GENERATION AND UPSCALING USING GENERATIVE ADVERSIAL NETWORK"** is the bonafide work of **"SAKTHI MURUGAN V (211520205122),SARAN NITHISH NA (211520205131), SHYAM GANESH J (211520205138)"** who carried out the project work under my supervision.

SIGNATURE

Dr. S. SUMA CHRISTAL MARY M.E, Ph.D

HEAD OF THE DEPARTMENT,

Department of Information Technology,

Panimalar Institute of Technology

,Poonamallee,Chennai 600 123.

SIGNATURE

Mrs. M. RAMYA, M.E.,

ASSISTANT PROFESSOR,

Department of Information Technology,

Panimalar Institute of Technology,

Poonamallee,Chennai 600 123.

Certified that the candidates were examined in the university project viva-voce held on_____at Panimalar Institute of Technology, Chennai 600123.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

A project of this magnitude and nature requires kind co-operation and support from many, for successful completion. We wish to express our sincere thanks to all those who were involved in the completion of this project.

We seek the blessing from the **Founder** of our institution **Dr.JEPPIAAR, M.A, Ph.D,** for having been a role model who has been our source of inspiration behind our success in education in his premier institution. Our sincere thanks to the Honorable Chairman of our prestigious institution **Mrs.REMIBAI JEPPIAAR** for her sincere endeavor in educating us in her premier institution.

We would like to express our deep gratitude to our beloved **Secretary and Correspondent Dr.P.CHINNADURAI, M.A, Ph.D,** for his kind words and enthusiastic motivation which inspired us a lot in completing this project.

We also express our sincere thanks and gratitude to our dynamic **Directors Mrs.C.VIJAYA RAJESHWARI, Dr.C.SAKTHI KUMAR, M.E, Ph.D,** and **Mrs. SARANYA SREE SAKTHI KUMAR, B.E, MBA, Ph.D** for providing us with necessary facilitiesfor completion of this project.

We also express our appreciation and gratefulness to our respected **Principal Dr. T. JAYANTHY, M.E, Ph.D,** who helped us in the completion of the project. We wish to convey our thanks and gratitude to our **Head of the Department, Dr. S. SUMA CHRISTAL MARY, M.E, Ph.D,** for her full support by providing ample time to complete our project. Special thanks to our Project Coordinator **MRS. G. DHANALAKSHMI, M.E., Associate Professor** and Internal Guide **Mrs. M. RAMYA, M.E., Assistant Professor,** for her expert advice, valuable information and guidance throughout the completion of the project.

Last, we thank our parents and friends for providing their extensive moral support and encouragement during the course of the project

ABSTRACT

Generative Adversarial Networks (GANs) have emerged as a cornerstone in the realm of image generation and upscaling, presenting exciting prospects for synthesizing high-fidelity images and enhancing the resolution of low-quality inputs. In this project, we delve into the application of GANs in these tasks, employing a robust toolkit comprising PyTorch, Tensor Board, CUDA, and cuDNN. By harnessing the parallel processing capabilities of GPUs, we expedite model training and improve performance, highlighting the importance of sophisticated frameworks and visualization tools in facilitating streamlined development and evaluation processes. Our research transcends theoretical exploration to tackle practical challenges across diverse domains, including computer vision, medical imaging, and entertainment. Through methodical experimentation and rigorous evaluation, we aim to showcase the effectiveness of our approach in generating high-quality images and preserving crucial visual details during upscaling. By pushing the boundaries of image synthesis and resolution enhancement techniques, we aspire to provide tangible solutions with real-world applications. The significance of our work lies in its potential to advance the field of computer vision by introducing novel methodologies for image generation and upscaling. By delivering comprehensive insights and practical solutions, our research aims to address pressing challenges and open new avenues for innovation across various industries. In essence, this project represents a concerted effort to harness the power of GANs and advanced computational tools to push the boundaries of image synthesis and enhancement. Through systematic experimentation, rigorous evaluation, and practical applications, we endeavor to contribute meaningfully to the ongoing evolution of computer vision technologies and their real-world impact.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

In our project, we're delving into Generative Adversarial Networks (GANs) using PyTorch to create and enhance images. GANs consist of two parts: a generator and a discriminator, working together to produce lifelike images. We're harnessing the capabilities of PyTorch, a robust deep learning framework, to train our GAN model effectively. Alongside, we're employing TensorBoard, a visualization tool, to monitor and analyze the training process. This helps us understand how well our model is performing and make necessary tweaks. The ability to generate and enhance images has practical applications across many fields, including computer vision, medical imaging, and entertainment. Our project aims to advance image processing techniques, improving image quality and realism. By combining GANs, PyTorch, and TensorBoard, we strive to make significant strides in image generation and enhancement tasks. Ultimately, our goal is to produce high-quality, realistic images that can be used in various applications. from creating immersive digital experiences to assisting medical professionals in diagnosing diseases from medical images. With the power of GANs, PyTorch, and TensorBoard combined, we are confident in our ability to make meaningful contributions to the field of image processing and artificial intelligence as a whole. In our pursuit of advancing image processing techniques, we aspire to push the boundaries of what's achievable in generating high-quality, realistic images. By seamlessly integrating GANs, PyTorch, and TensorBoard, we are poised to catalyze transformative innovations across diverse domains.

1.2 PROBLEM DEFINITION

i. Image Quality Enhancement: Our project addresses the challenge of improving image quality, particularly by enhancing low-resolution images to higher resolutions, which is crucial in various domains such as surveillance, medical imaging, and satellite imagery analysis.

ii. Realistic Image Generation: We tackle the problem of generating realistic images, which is essential for applications like virtual reality, gaming, and content creation, where high-fidelity visuals are paramount for user immersion and engagement.

iii. Resource Efficiency in Training: By leveraging PyTorch and Tensor Board, we aim to streamline the training process of Generative Adversarial Networks (GANs), making it more resource-efficient and accessible to developers and researchers, thereby reducing computational costs and time requirements.

iv. Versatility Across Industries: Our project offers solutions that transcend industry boundaries, catering to the diverse needs of sectors ranging from healthcare to entertainment. The ability to generate and upscale images efficiently has implications in fields like diagnosis through medical imaging, artistic content creation, and enhancing visual experiences in multimedia applications

1.3 LITERATURE SURVEY

INTRODUCTION

A literature review is a body of text that aims to review the criticalpoints of current knowledge on and/or methodological approachesto a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes informationin a particular subject area within a certain time period. Its ultimategoal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources

LITERATURE SURVEY

REFERENCE 1: Model-Guided Generative Adversarial Networks for Unsupervised Fine-Grained Image Generation.

AUTHOR : Jian Xiao, Xiaojun Bi

DESCRIPTION :

Unsupervised fine-grained image generation is a challenging issue in computer vision. Although many recent significant advances have improved performance, the ability to synthesize photo-realistic images in an unsupervised manner remains extremely difficult. The existing methods compose an image via complex three-stage generative adversarial networks and impose constraints between the latent codes. This pipeline focuses on the disentanglement and ignores the quality of generated images. In this article, we propose a novel two-stage approach for unsupervised fine-grained image generation, termed Model-Guided Generative Adversarial Networks (MG-GAN). We introduce an attention module for exploring the correlation between fine-grained latent codes and image features in the foreground generation stage. The attention module

enables the network to automatically focus on the color details and semantic concepts of objects related to different fine-grained classes. Furthermore, we incorporate knowledge distillation strategy and design a simple but effective inverse background image generator as a teacher to guide the background image generation. With the help of knowledge learned in the pre-trained inverse background image generator, a comfortable canvas is synthesized and combined with foreground object more reasonably. Extensive experiments on three popularly fine-grained datasets demonstrate that our approach achieves state-of-the-art performance and is even competitive with semi-supervised method.

REFERENCE 2: CONTEXT-GAN: CONTROLLABLE CONTEXT IMAGE GENERATION USING GANS.

AUTHOR : Marc-Adrien Hostin, Vladimir Sivtsov, Shahram Attarian, David Bendahan, Marc-Emmanuel Bellemare.

DESCRIPTION:

It propose an enhancement to label-to-image GANs. Based on a Pix2Pix architecture, ConText-GAN allows generating images in a controlled way. Given a feature map as input, ConText-GAN can generate images with a specified layout and label content. As an application, ConText-GAN is used to perform a more realistic than usual data augmentation from an MRI dataset. We show the validity of the generated images with respect to the input feature maps. The relevance of the approach is demonstrated by the improvement of the segmentation result following a data augmentation performed with ConText-GAN compared to classical methods. A practical application is presented in the challenging context of U-Net segmentation of MRI of fat infiltrated muscles.

REFERENCE 3: Study on the Adversarial Sample Generation Algorithm Based on Adversarial Quantum Generation Adversarial Network.

AUTHOR : Wen Cheng, ShiBin Zhang , Yusheng Lin.

DESCRIPTION :

This paper proposes a quantum adversarial sample generation algorithm (QASGA) based on adversarial quantum generative adversarial networks. First, the real samples are encoded into quantum states, then the generator G of QGAN is used to generate adversarial perturbations, which are superimposed with normal samples to obtain adversarial samples. At the same time, the SWAP-TEST method is used to calculate the similarity between all real samples and adversarial samples at once, thereby accelerating adversarial attacks. Experimental results show that the QASGA algorithm proposed in this paper can generate high-fidelity adversarial samples with a one-time similarity calculation on the IRIS dataset, and can reduce the classification accuracy of the BP neural network from 96.67% to 26.67%, verifying the effectiveness of the proposed QASGA algorithm. This paper proposed the QASGA method by utilizing the advantages of generative adversarial networks in data generation and quantum computing in computing efficiency. It presents a novel approach that harnesses the strengths of both generative adversarial networks and quantum computing to enhance the efficiency and potency of adversarial attacks. By introducing QASGA, the research community gains valuable insights into the intersection of quantum computing and machine learning security, paving the way for future advancements in adversarial defense mechanisms and cybersecurity strategies.

REFERENCE4: Research on Generative Adversarial Networks and Their Applications in Image Generation

AUTHOR : Jiayi Zhou.

DESCRIPTION :

This paper first introduces the basic principles, model structures, and advantages and disadvantages of generative adversarial networks. Then we give a detailed introduction to three application areas of generative adversarial networks in image generation: medical imaging, 3D reconstruction, and image fusion. Finally, the development trend of generative adversarial networks and their applications in the field of image generation prospects.

REFERENCE 5: Profile SR-GAN: A GAN Based Super-Resolution Method for Generating High-Resolution Load Profiles.

AUTHOR: Lidong Song, Yiyan Li , Ning Lu.

DESCRIPTION:

The paper presents ProfileSR-GAN, a two-stage framework for enhancing low-resolution load profiles (LRLPs) to high-resolution (HRLPs). In the first stage, a GAN-based model restores high-frequency components from LRLPs, incorporating weather data for improved accuracy. The second stage introduces a polishing network to refine HRLPs, improving realism and matching accuracy. Simulation results show ProfileSR-GAN outperforms existing methods in shape-based metrics and enhances downstream tasks like load monitoring, demonstrating its effectiveness in restoring high-frequency components and improving performance.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

- Traditional Image Processing Methods: While traditional methods like interpolation techniques can upscale images, they often result in loss of detail and produce unrealistic outputs. Our project surpasses these methods by utilizing advanced deep learning techniques, specifically GANs, to generate high-quality, realistic images with enhanced details and textures

- Conventional GAN Implementations: Many existing GAN implementations lack user-friendly interfaces and robust training mechanisms, making them challenging to use effectively. In contrast, our project provides a seamless integration with PyTorch and leverages Tensor Board for efficient training visualization and monitoring. This improves user experience and enables better control over the training process.

- Manual Image Enhancement Techniques: Manual image enhancement techniques require expertise and are often time-consuming, limiting their practicality in real-world applications. Our project automates the image enhancement process using GANs, reducing the need for manual intervention and enabling scalable and efficient image enhancement solutions across various domains.

2.2 PROPOSED SYSTEM

- Advanced Deep Learning Techniques: Our proposed system leverages state-of-the-art deep learning techniques, particularly Generative Adversarial Networks (GANs), to enhance image quality and realism. By training the model on large datasets, it learns intricate patterns and textures, resulting in high-fidelity image upscaling without loss of detail.

- Streamlined User Interface and Training Mechanisms: Our system provides an intuitive user interface and incorporates efficient training mechanisms, ensuring a seamless experience for users. Integration with PyTorch facilitates straightforward model development and customization, while TensorBoard enables real-time visualization of training progress and metrics, empowering users to optimize model performance effectively.

- Automated Image Enhancement Pipeline: The proposed system automates the image enhancement process through GANs, eliminating the need for manual intervention. By harnessing the power of deep learning, it offers scalable and efficient image enhancement solutions across diverse domains, saving time and resources while maintaining superior output quality.

- Scalable Infrastructure and Cloud Integration: Our system is designed to be highly scalable and adaptable, capable of handling large datasets and complex models with ease. Integration with cloud-based computing resources enables seamless scalability, allowing users to leverage distributed computing for accelerated training and improved efficiency.

2.3 DEVELOPMENT ENVIROMENT

SOFTWARE REQUIREMENT

- Python -3.11.7
- PyTorch -2.2.1
- Tensorboard -2.16.2
- CUDA and cuDNN -10.1
- Conda -24.1.2
- Jupyter Notebook

HARDWARE REQUIREMENT

- Processor: Minimum 3.8GHz
- Memory (RAM): 16 GB
- 6gb Graphics Card
- Hard Drive: 200GB

CHAPTER 3

SYSTEM DESIGN

3.1 UML DIAGRAMS

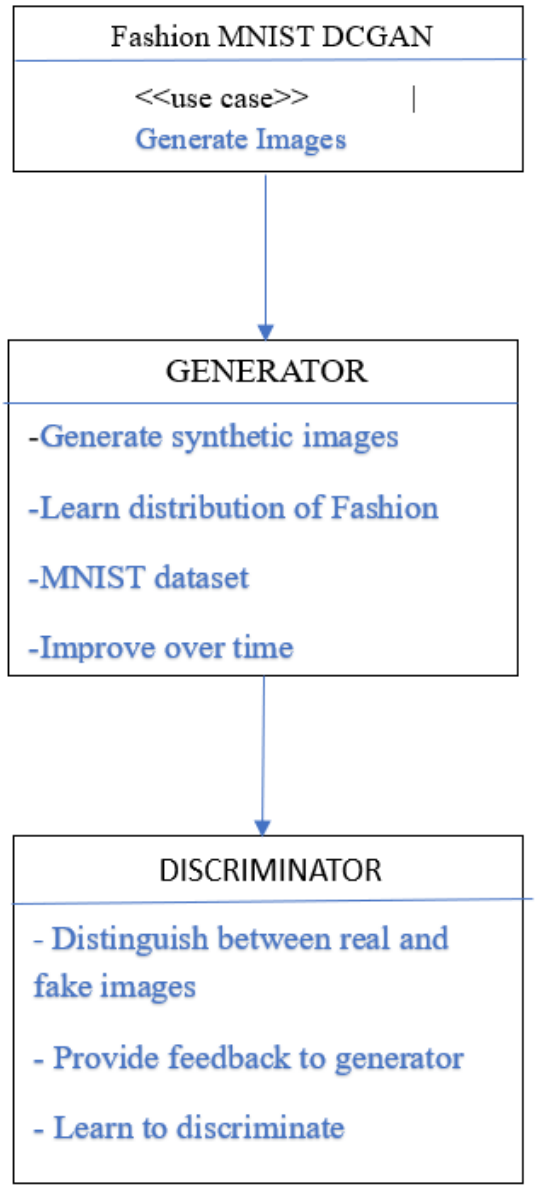


Fig 3.1.1 Use case diagram for Deep convolutional Generative Adversarial Network

This use case diagram for DCGAN depict scenarios such as "Training the DCGAN model on a dataset" and "Generating synthetic images using the trained DCGAN model."

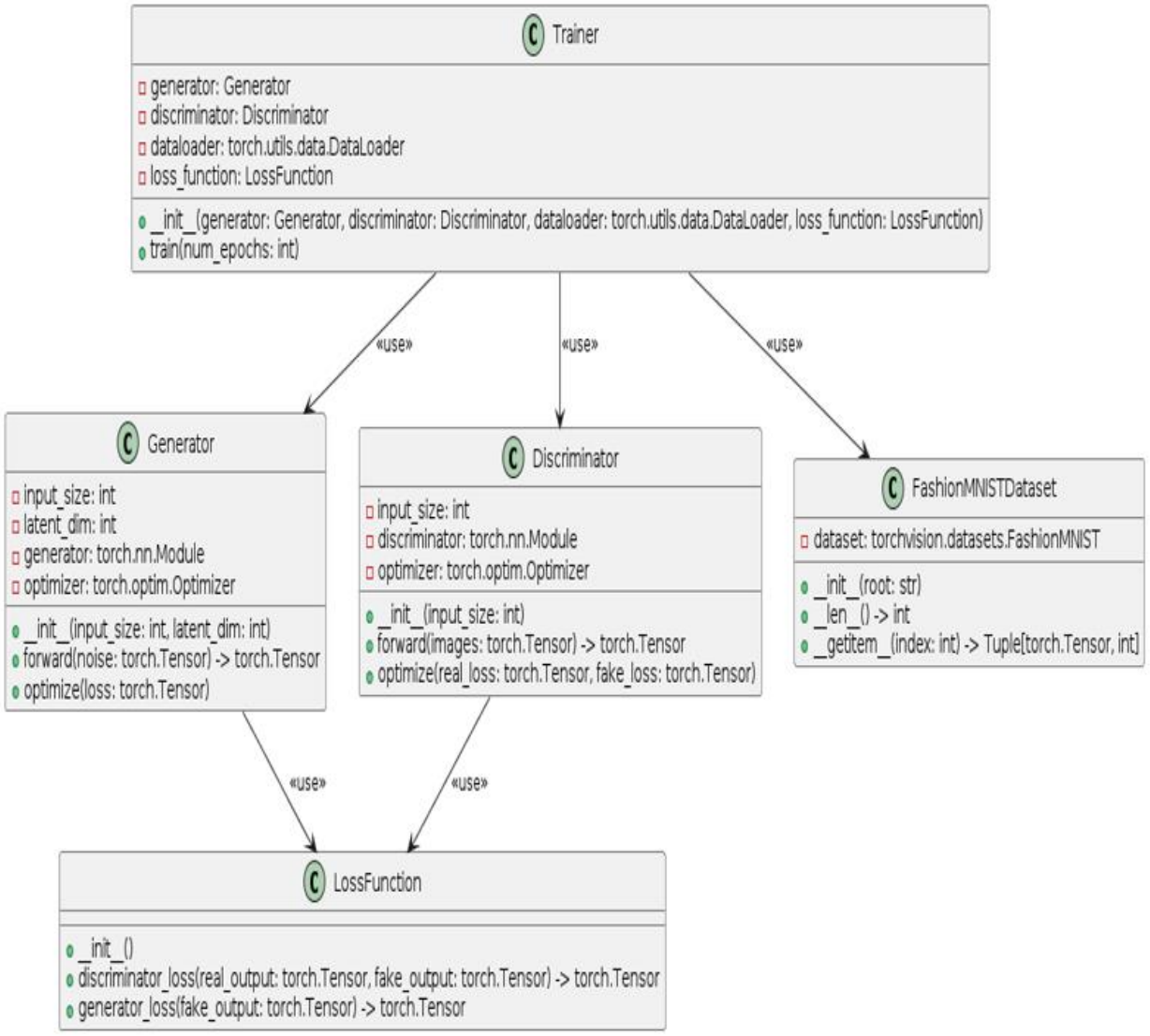


Fig 3.1.2 Class diagram for Generative Adversarial Network

This class diagram for a Generative Adversarial Network (GAN) depicts the structure of the network, including classes such as Generator, Discriminator, Loss functions, and Optimizers, illustrating their interactions and relationships.

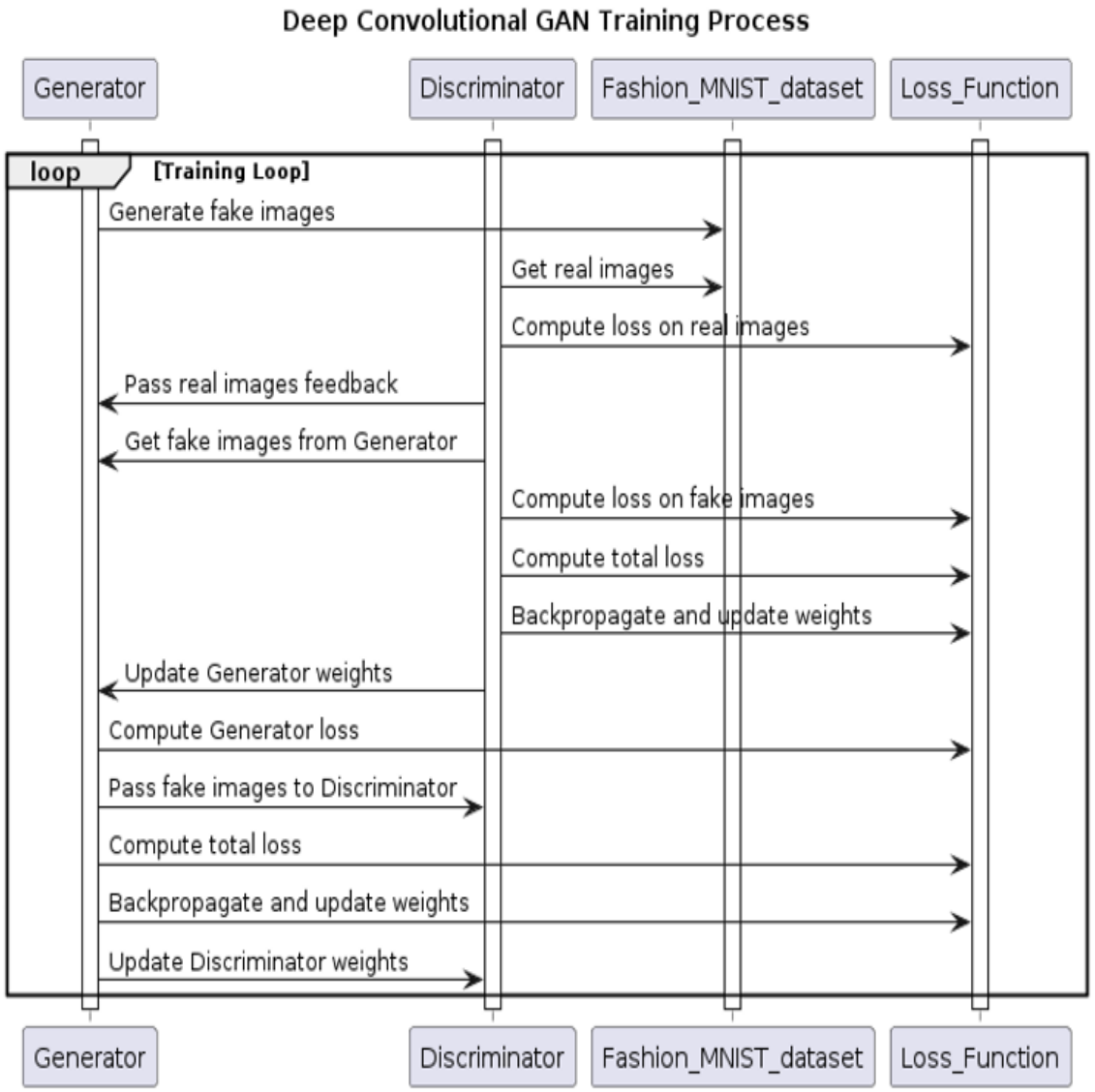


Fig 3.1.3 Sequence diagram for Deep convolutional Generative Adversarial Network

The sequence diagram for a DC Generative Adversarial Network (GAN) illustrates the flow of interactions between the generator and discriminator networks during the training process.

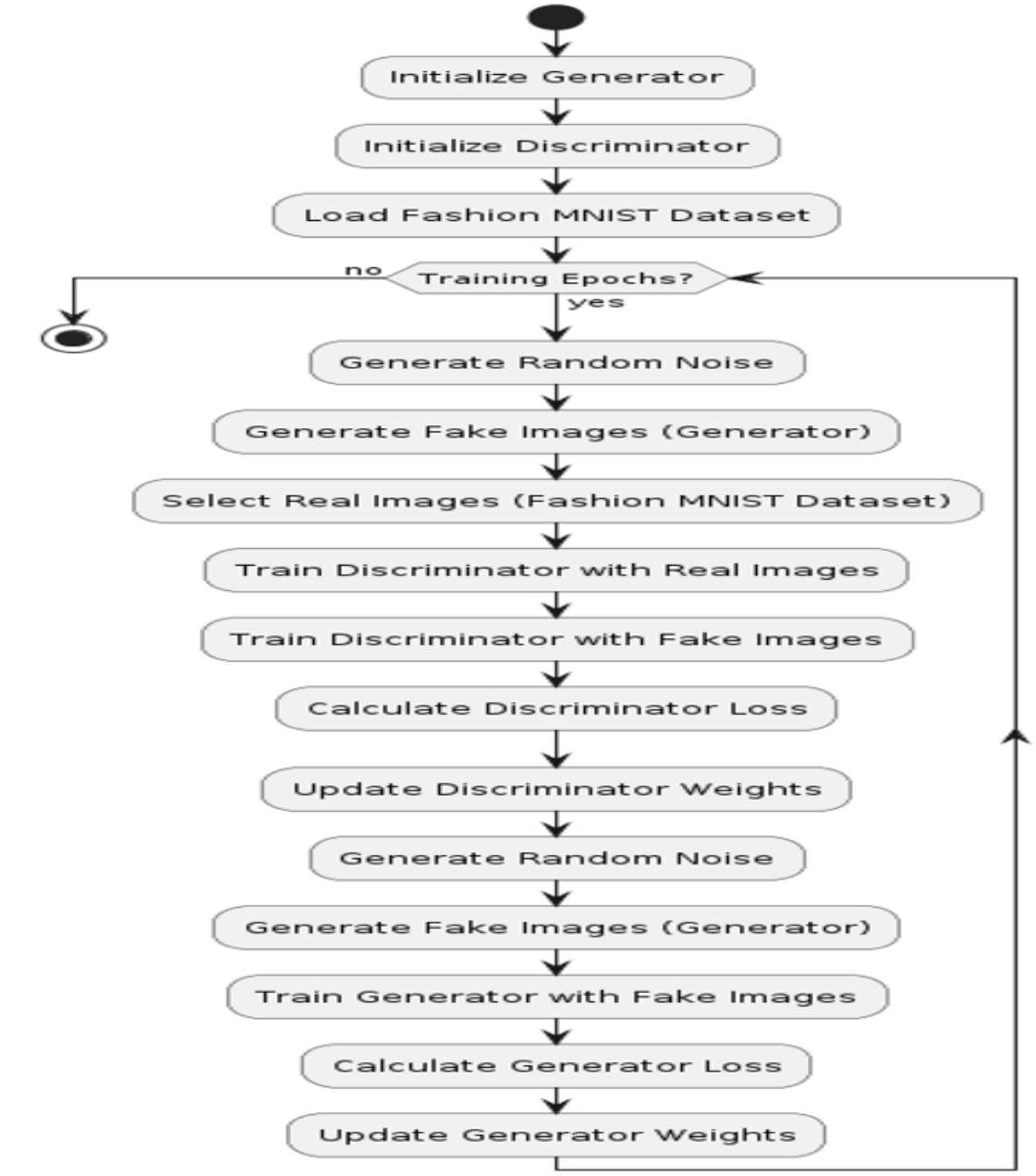


Fig 3.1.4 Activity diagram for Generative Adversarial Network

This activity diagram for a generative adversarial network (GAN) depicts the flow of operations between the generator and discriminator networks, illustrating how they interact iteratively to improve the generated outputs' quality.

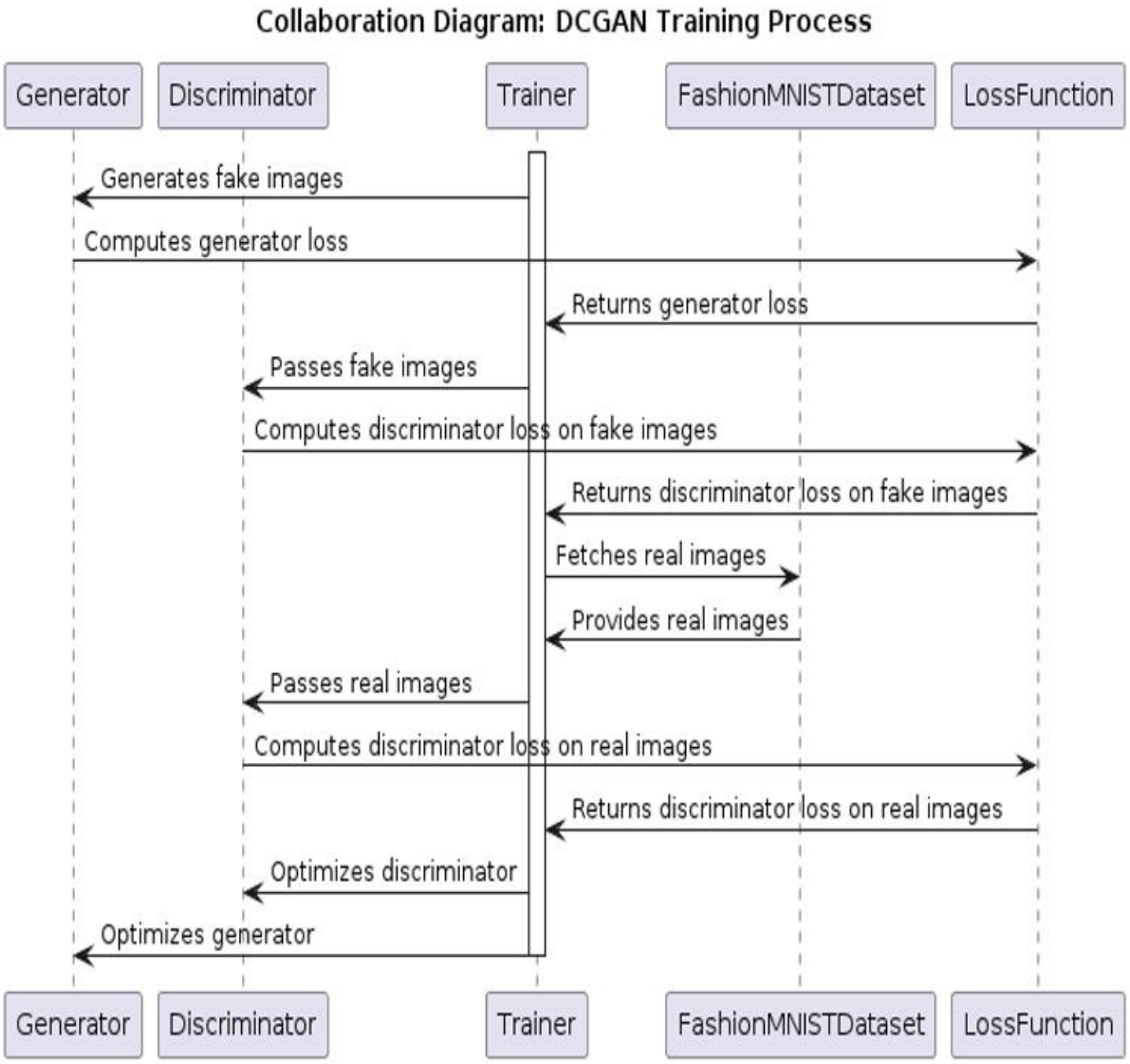


Fig 3.1.3 Collaboration diagram for Deep convolutional Generative Adversarial Network

The A collaboration diagram for a deep convolutional generative adversarial network illustrates the interaction between the generator and discriminator networks, enabling the iterative refinement of generated images through adversarial training.

3.2 DATAFLOW DIAGRAM

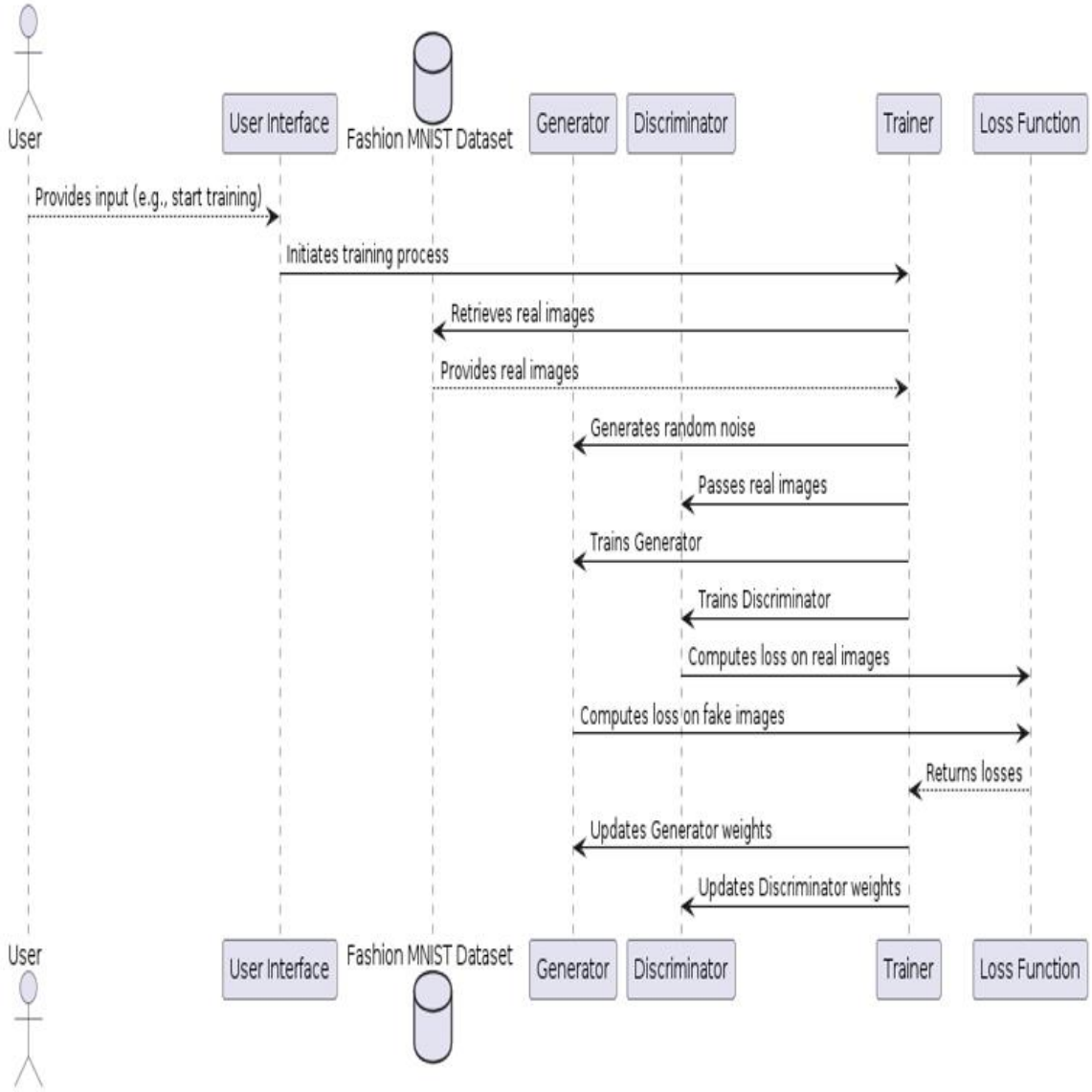


Fig 3.3.1 Dataflow diagram for Generative Adversarial Network

A data flow diagram for a deep convolutional generative adversarial network illustrates the flow of data through the network's layers, depicting the generation of synthetic data by the generator and the discrimination between real and synthetic data by the discriminator.

CHAPTER 4

SYSTEM ARCHITECTURE

4.1 ARCHITECTURE OVERVIEW

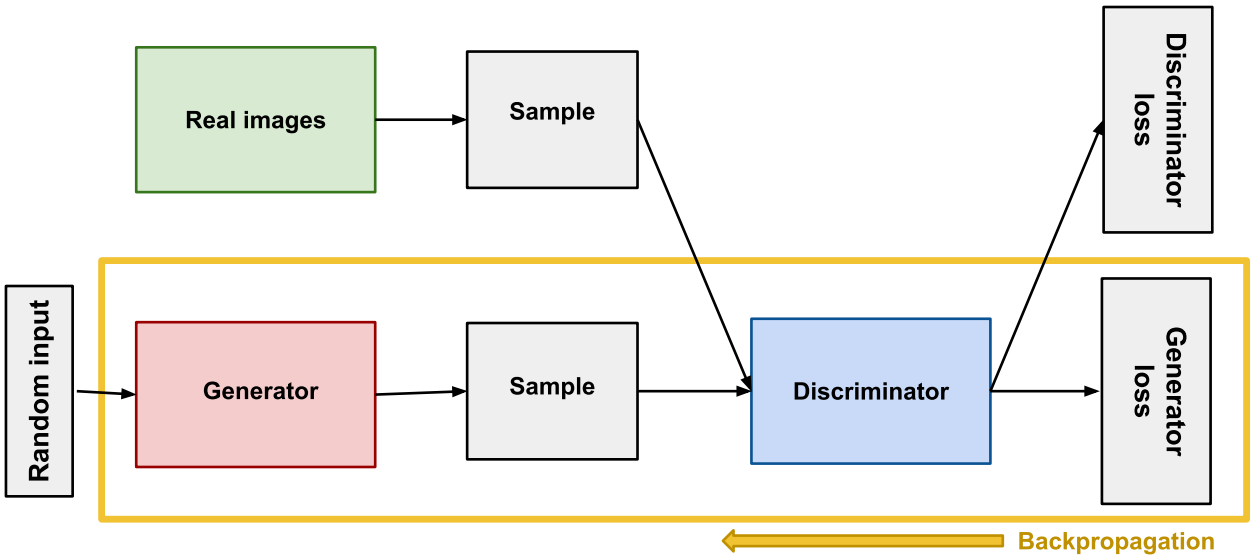


Fig 4.1 Architecture diagram for Deep convolutional Generative Adversarial Network

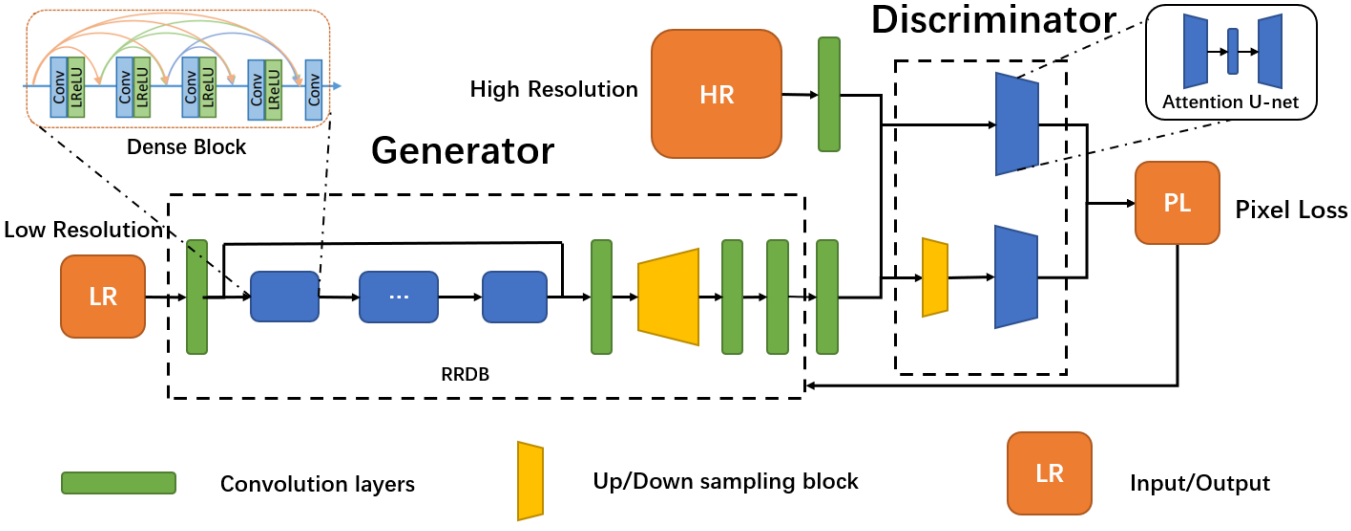


Fig 4.2 Architecture diagram for Enhanced Super-Resolution Generative Adversarial Network

The architecture diagram for a deep convolutional generative adversarial network (DCGAN) typically consists of two main components: the generator and the discriminator. The generator comprises multiple layers of transposed convolutional and batch normalization operations, transforming a latent input vector into synthetic images. Conversely, the discriminator consists of convolutional layers followed by batch normalization and LeakyReLU activation functions, aiming to distinguish between real and generated images. The latent input vector serves as the seed for image generation in the generator, while the discriminator receives both real and generated images, providing feedback to the generator for improvement through adversarial training. This adversarial process encourages the generator to produce increasingly realistic images while simultaneously enhancing the discriminator's ability to differentiate between real and fake samples. Overall, the DCGAN architecture leverages deep convolutional neural networks to generate high-quality images through adversarial learning.

ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) enhances image resolution through adversarial training. Unlike traditional GANs, ESRGAN's generator employs a deep residual network (ResNet) with skip connections for gradient propagation. Perceptual loss, integrated into the loss function, measures the perceptual similarity between generated and ground truth images, alongside adversarial and content losses. This encourages the generator to produce high-fidelity images with enhanced structural details. ESRGAN achieves state-of-the-art results in image super-resolution, generating visually appealing outputs with reduced artifacts. The adversarial training process ensures the generated images are perceptually indistinguishable from real high-resolution images, as determined by the discriminator.

4.2 MODULE DESCRIPTION

IMAGE GENERATION:

Our image generation architecture utilizes Generative Adversarial Networks (GANs) implemented in PyTorch, accelerated with CUDA and cuDNN for GPU computing, and augmented with Tensor Board for efficient training visualization. The architecture consists of two key components: the generator and the discriminator.

- Generator: -

  - The generator takes a random noise vector sampled from a latent space and transforms it into a synthetic image.
  - Implemented as a neural network using PyTorch, the generator comprises convolutional and transposed convolutional layers to up sample the noise vector into an image.
  - CUDA and cuDNN optimizations enable GPU acceleration, significantly speeding up the training process by leveraging parallel computing capabilities.

- Discriminator: -

  - The discriminator receives input images from both the generated output of the generator and real images from the fashion MNIST dataset.
  - Also implemented as a neural network using PyTorch, the discriminator consists of convolutional layers followed by fully connected layers to classify the authenticity of input images.
  - GPU acceleration with CUDA and cuDNN enhances the discriminator's training efficiency, allowing for faster convergence and improved performance.

- Training Process: -

  - During training, the generator and discriminator are trained iteratively in a min-max game, where the generator aims to produce images that deceive the discriminator, while the discriminator strives to differentiate between real and fake images.
  - The use of CUDA and cuDNN accelerates the training process, leveraging GPU parallelism to compute gradients and update model parameters efficiently.
  - TensorBoard integration provides real-time visualization of training metrics, including loss functions, model performance, and generated image quality, facilitating model monitoring and performance evaluation.

- Dataset: -

  - The fashion MNIST dataset comprises grayscale images of fashion items such as clothing and accessories, providing a diverse and realistic set of images for training the GAN model.
  - The dataset is preprocessed and loaded efficiently using PyTorch's data loading utilities, ensuring smooth and effective training.

In summary, our architecture leverages the combined power of PyTorch, CUDA, cuDNN, and TensorBoard to train a GAN model capable of generating high-quality synthetic fashion images. By harnessing GPU acceleration and efficient training visualization, we achieve faster convergence and superior performance in image generation tasks. Moreover, the integration of TensorBoard enables us to visualize and monitor the training process in real-time, facilitating insights into model performance and guiding necessary adjustments. The amalgamation of these technologies culminates in a sophisticated architecture that not only achieves faster convergence but also delivers superior performance in diverse image generation tasks, establishing new benchmarks in synthetic image generation within the fashion domain.

IMAGE UPSCALING:

Our image upscaling architecture is based on the Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) implemented using BasicSR from PyTorch, with acceleration provided by CUDA and cuDNN for efficient GPU computation.

- Generator (ESRGAN): -

  - The ESRGAN generator is designed to upscale low-resolution images to higher resolutions while preserving important visual details.
  - It consists of convolutional layers with skip connections, residual blocks, and up sampling layers to gradually increase the image resolution.
  - The network architecture is optimized to generate high-quality images with enhanced textures and sharpness, surpassing traditional upscaling methods.

- Training Process: -

  - During training, the ESRGAN generator is trained using a combination of adversarial loss, content loss, and perceptual loss functions.
  - Adversarial loss encourages the generator to produce high-resolution images that are visually similar to real high-resolution images, as perceived by a discriminator network.
  - Content loss compares the features of the generated image and the ground truth high-resolution image to ensure content consistency.
  - Perceptual loss measures the similarity between the features extracted from the generated and ground truth images using pre-trained deep neural networks, promoting perceptually realistic results.
  - BasicSR from PyTorch provides a comprehensive set of utilities for training and evaluating super-resolution models, including data loading, loss functions, and model optimization.

- Acceleration with CUDA and cuDNN: -

  - CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model developed by NVIDIA for GPU acceleration.
  - cuDNN (CUDA Deep Neural Network library) is a GPU-accelerated library of primitives for deep neural networks, optimized for CUDA-compatible GPUs.
  - By leveraging CUDA and cuDNN, our architecture harnesses the computational power of NVIDIA GPUs to accelerate the training and inference processes of the ESRGAN model, significantly reducing processing time.

Overall, our architecture utilizes ESRGAN implemented with BasicSR from PyTorch, accelerated by CUDA and cuDNN, to perform high-quality image upscaling. This framework enables efficient training and inference of the super-resolution model, resulting in visually pleasing upscaled images with enhanced details and clarity.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Image Generation

ImageGeneration.ipynb:

```
import torch
from torch import nn
class Generator(nn.Module):
    def __init__(self, noise_channels, image_channels, features):
        super(Generator, self).__init__()

        # define the model
        self.model = nn.Sequential(
            # Transpose block 1
            nn.ConvTranspose2d(noise_channels, features*16, kernel_size=4,
stride=1, padding=0),
            nn.ReLU(),

            # Transpose block 2
            nn.ConvTranspose2d(features*16, features*8, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(features*8),
            nn.ReLU(),

            # Transpose block 3
            nn.ConvTranspose2d(features*8, features*4, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(features*4),
            nn.ReLU(),
            # Transpose block 4
            nn.ConvTranspose2d(features*4, features*2, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(features*2),
            nn.ReLU(),

            # Last transpose block (different)
            nn.ConvTranspose2d(features*2, image_channels, kernel_size=4,
stride=2, padding=1),
            nn.Tanh(),
        )
```

```
def forward(self, x):
        return self.model(x)

class Discriminator(nn.Module):
    def __init__(self, image_channels, features):
        super(Discriminator, self).__init__()

        # define the model
        self.model = nn.Sequential(
            # define the first Conv block
            nn.Conv2d(image_channels, features, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2),

            # Conv block 2
            nn.Conv2d(features, features*2, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*2),
            nn.LeakyReLU(0.2),

            # Conv block 3
            nn.Conv2d(features*2, features*4, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*4),
            nn.LeakyReLU(0.2),

            # Conv block 4
            nn.Conv2d(features*4, features*8, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*8),
            nn.LeakyReLU(0.2),

            # Conv block 5 (different)
            nn.Conv2d(features*8, 1, kernel_size=4, stride=2, padding=0),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.model(x)
```

```
import torch
import torchvision
from torch import nn
from torch import optim
from torchvision.datasets import FashionMNIST
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

LEARNING_RATE = 0.0005
BATCH_SIZE = 256
IMAGE_SIZE = 64
EPOCHS = 1000
image_channels = 1
noise_channels = 256
gen_features = 64
disc_features = 64

device = torch.device("cuda")

data_transforms = transforms.Compose([
        transforms.Resize(IMAGE_SIZE),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
])

dataset = FashionMNIST(root="dataset/", train=True,
transform=data_transforms, download=True)
dataloader = DataLoader(dataset, batch_size=BATCH_SIZE,
shuffle=True)

gen_model  = Generator(noise_channels, image_channels,
gen_features).to(device)
disc_model = Discriminator(image_channels, disc_features).to(device)

gen_optimizer = optim.Adam(gen_model.parameters(),
lr=LEARNING_RATE, betas=(0.5, 0.999))
disc_optimizer = optim.Adam(disc_model.parameters(),
lr=LEARNING_RATE, betas=(0.5, 0.999))
```

```
criterion = nn.BCELoss()

gen_model.train()
disc_model.train()

fake_label = 0
real_label = 1

fixed_noise = torch.randn(64, noise_channels, 1, 1).to(device)

writer_real = SummaryWriter(f"runs/fashion/test_real")
writer_fake = SummaryWriter(f"runs/fashion/test_fake")

step = 0

print("Start training...")

# loop over all epochs and all data
for epoch in range(EPOCHS):
   for batch_idx, (data, target) in enumerate(dataloader):
      # set the data to cuda
      data = data.to(device)

      # get the batch size
      batch_size = data.shape[0]

      # Train the discriminator model on real data
      disc_model.zero_grad()
      label = (torch.ones(batch_size) * 0.9).to(device)
      output = disc_model(data).reshape(-1)
      real_disc_loss = criterion(output, label)
      d_x = output.mean().item()

# train the disc model on fake (generated) data
      noise = torch.randn(batch_size, noise_channels, 1, 1).to(device)
      fake = gen_model(noise)
      label = (torch.ones(batch_size) * 0.1).to(device)
      output = disc_model(fake.detach()).reshape(-1)
      fake_disc_loss = criterion(output, label)
```

```
# calculate the final discriminator loss
        disc_loss = real_disc_loss + fake_disc_loss

        # apply the optimizer and gradient
        disc_loss.backward()
        disc_optimizer.step()

        # train the generator model
        gen_model.zero_grad()
        label = torch.ones(batch_size).to(device)
        output = disc_model(fake).reshape(-1)
        gen_loss = criterion(output, label)
        # apply the optimizer and gradient
        gen_loss.backward()
        gen_optimizer.step()

        # print losses in console and tensorboard
        if batch_idx % 50 == 0:
            step += 1
            # print everything
            print(
                f"Epoch: {epoch} ===Batch: {batch_idx}/{len(dataloader)} === Disc
loss: {disc_loss:.4f} === Gen loss: {gen_loss:.4f}"
            )
            ### test the model
            with torch.no_grad():
                # generate fake images
                fake_images = gen_model(fixed_noise)
                # make grid in the tensorboard
                img_grid_real = torchvision.utils.make_grid(data[:40], normalize=True)
                img_grid_fake = torchvision.utils.make_grid(fake_images[:40],
normalize=True)
                # write the images in tensorbaord
                writer_real.add_image(
                    "Real images", img_grid_real, global_step=step
                )
                writer_fake.add_image(
                    "Generated images", img_grid_fake, global_step=step
                )

torch.save(gen_model.state_dict(), 'generator.pth')
torch.save(disc_model.state_dict(), 'discriminator.pth')
```

```
%load_ext tensorboard

%tensorboard --logdir="C:\Users\admin\Project\runs\fashion"

import torch

# Assuming you've defined your Generator model class and its architecture

# Instantiate the generator model
generator = Generator(noise_channels, image_channels, gen_features)

# Load the saved state dictionary
generator.load_state_dict(torch.load('generator.pth',
map_location=torch.device('cpu')))  # Load onto CPU

# Move the generator model to the appropriate device (CPU or GPU)
generator.to(device)

# Set the generator model to evaluation mode
generator.eval()

# Generate fake images
with torch.no_grad():
    # Assuming fixed_noise is your fixed input noise tensor
    fixed_noise = torch.randn(batch_size, noise_channels, 1, 1).to(device)
    fake_images = generator(fixed_noise)

# Now fake_images contains the generated fake images
import matplotlib.pyplot as plt
import torchvision.utils

# Convert the fake images to a grid
fake_images_grid = torchvision.utils.make_grid(fake_images.cpu(), nrow=8,
normalize=True)

# Convert the PyTorch tensor to a NumPy array and transpose the dimensions
fake_images_grid_np = fake_images_grid.permute(1, 2, 0).numpy()

# Display the fake images grid using matplotlib
plt.figure(figsize=(10, 10))
plt.imshow(fake_images_grid_np)
plt.axis('off')
plt.show()
```

ImageUpscaling:

net_interp.py

```
import sys
import torch
from collections import OrderedDict

alpha = float(sys.argv[1])

net_PSNR_path = './models/RRDB_PSNR_x4.pth'
net_ESRGAN_path = './models/RRDB_ESRGAN_x4.pth'
net_interp_path = './models/interp_{:02d}.pth'.format(int(alpha*10))

net_PSNR = torch.load(net_PSNR_path)
net_ESRGAN = torch.load(net_ESRGAN_path)
net_interp = OrderedDict()

print('Interpolating with alpha = ', alpha)

for k, v_PSNR in net_PSNR.items():
    v_ESRGAN = net_ESRGAN[k]
    net_interp[k] = (1 - alpha) * v_PSNR + alpha * v_ESRGAN

torch.save(net_interp, net_interp_path)
```

transer_RRDB_models.py

```
import os
import torch
import RRDBNet_arch as arch
pretrained_net = torch.load('./models/RRDB_ESRGAN_x4.pth')
save_path = './models/RRDB_ESRGAN_x4.pth'
crt_model = arch.RRDBNet(3, 3, 64, 23, gc=32)
crt_net = crt_model.state_dict()
load_net_clean = {}
for k, v in pretrained_net.items():
    if k.startswith('module.'):
        load_net_clean[k[7:]] = v
    else:
        load_net_clean[k] = v
pretrained_net = load_net_clean
print('#################################\n')
tbd = []
for k, v in crt_net.items():
    tbd.append(k)
# directly copy
for k, v in crt_net.items():
    if k in pretrained_net and pretrained_net[k].size() == v.size():
        crt_net[k] = pretrained_net[k]
        tbd.remove(k)
crt_net['conv_first.bias'] = pretrained_net['model.0.bias']
for k in tbd.copy():
    if 'RDB' in k:
        ori_k = k.replace('RRDB_trunk.', 'model.1.sub.')
        if '.weight' in k:
            ori_k = ori_k.replace('.weight', '.0.weight')
        elif '.bias' in k:
            ori_k = ori_k.replace('.bias', '.0.bias')
        crt_net[k] = pretrained_net[ori_k]
        tbd.remove(k)
crt_net['trunk_conv.weight'] = pretrained_net['model.1.sub.23.weight']
crt_net['upconv1.weight'] = pretrained_net['model.3.weight']
crt_net['upconv2.weight'] = pretrained_net['model.6.weight']
crt_net['HRconv.weight'] = pretrained_net['model.8.weight']
crt_net['conv_last.weight'] = pretrained_net['model.10.weight']
crt_net['conv_last.bias'] = pretrained_net['model.10.bias']
torch.save(crt_net, save_path)
print('Saving to ', save_path)
```

RRDBNet_arch.py

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

def make_layer(block, n_layers):
    layers = []
    for _ in range(n_layers):
        layers.append(block())
    return nn.Sequential(*layers)

class ResidualDenseBlock_5C(nn.Module):
    def __init__(self, nf=64, gc=32, bias=True):
        super(ResidualDenseBlock_5C, self).__init__()
        # gc: growth channel, i.e. intermediate channels
        self.conv1 = nn.Conv2d(nf, gc, 3, 1, 1, bias=bias)
        self.conv2 = nn.Conv2d(nf + gc, gc, 3, 1, 1, bias=bias)
        self.conv3 = nn.Conv2d(nf + 2 * gc, gc, 3, 1, 1, bias=bias)
        self.conv4 = nn.Conv2d(nf + 3 * gc, gc, 3, 1, 1, bias=bias)
        self.conv5 = nn.Conv2d(nf + 4 * gc, nf, 3, 1, 1, bias=bias)
        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)
        # initialization
        # mutil.initialize_weights([self.conv1, self.conv2, self.conv3, self.conv4,
self.conv5], 0.1)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 = self.lrelu(self.conv2(torch.cat((x, x1), 1)))
        x3 = self.lrelu(self.conv3(torch.cat((x, x1, x2), 1)))
        x4 = self.lrelu(self.conv4(torch.cat((x, x1, x2, x3), 1)))
        x5 = self.conv5(torch.cat((x, x1, x2, x3, x4), 1))
        return x5 * 0.2 + x

class RRDB(nn.Module):
    '''Residual in Residual Dense Block'''
    def __init__(self, nf, gc=32):
        super(RRDB, self).__init__()
        self.RDB1 = ResidualDenseBlock_5C(nf, gc)
        self.RDB2 = ResidualDenseBlock_5C(nf, gc)
        self.RDB3 = ResidualDenseBlock_5C(nf, gc)
```

```
def forward(self, x):
        out = self.RDB1(x)
        out = self.RDB2(out)
        out = self.RDB3(out)
        return out * 0.2 + x


class RRDBNet(nn.Module):
    def __init__(self, in_nc, out_nc, nf, nb, gc=32):
        super(RRDBNet, self).__init__()
        RRDB_block_f = functools.partial(RRDB, nf=nf, gc=gc)

        self.conv_first = nn.Conv2d(in_nc, nf, 3, 1, 1, bias=True)
        self.RRDB_trunk = make_layer(RRDB_block_f, nb)
        self.trunk_conv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        #### upsampling
        self.upconv1 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.upconv2 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.HRconv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.conv_last = nn.Conv2d(nf, out_nc, 3, 1, 1, bias=True)

        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)

    def forward(self, x):
        fea = self.conv_first(x)
        trunk = self.trunk_conv(self.RRDB_trunk(fea))
        fea = fea + trunk

        fea = self.lrelu(self.upconv1(F.interpolate(fea, scale_factor=2,
mode='nearest')))
        fea = self.lrelu(self.upconv2(F.interpolate(fea, scale_factor=2,
mode='nearest')))
        out = self.conv_last(self.lrelu(self.HRconv(fea)))

        return out
```

test.py

```
import os.path as osp
import glob
import cv2
import numpy as np
import torch
import RRDBNet_arch as arch

model_path = 'models/RRDB_ESRGAN_x4.pth'  #
models/RRDB_ESRGAN_x4.pth OR models/RRDB_PSNR_x4.pth
device = torch.device('cuda')  # if you want to run on CPU, change 'cuda' -> cpu
# device = torch.device('cpu')

test_img_folder = 'LR/*'

model = arch.RRDBNet(3, 3, 64, 23, gc=32)
model.load_state_dict(torch.load(model_path), strict=True)
model.eval()
model = model.to(device)

print('Model path {:s}. \nTesting...'.format(model_path))

idx = 0
for path in glob.glob(test_img_folder):
    idx += 1
    base = osp.splitext(osp.basename(path))[0]
    print(idx, base)
# read images
img = cv2.imread(path, cv2.IMREAD_COLOR)
img = img * 1.0 / 255
img = torch.from_numpy(np.transpose(img[:, :, [2, 1, 0]], (2, 0, 1))).float()
img_LR = img.unsqueeze(0)
img_LR = img_LR.to(device)
with torch.no_grad():
        output = model(img_LR).data.squeeze().float().cpu().clamp_(0, 1).numpy()
output = np.transpose(output[[2, 1, 0], :, :], (1, 2, 0))
output = (output * 255.0).round()
cv2.imwrite('results/{:s}_rlt.png'.format(base), output)
```

CHAPTER 6

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example, the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

6.1 TYPES OF TESTING

6.1.1 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

6.1.2 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

6.1.3 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6.1.4 PERFORMANCE TESTING

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

6.1.5 INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure, while at the same time conducting tests to uncover error associated with interfacing. The following are the types of Integration Testing: -

• Top-down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the program module.

•Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

6.1.6 PROGRAM TESTING

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other errors in the program.

6.1.7 VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

6.1.8 USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any

system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required.

6.2 WHITEBOX AND BLACKBOX TESTING

6.2.1 WHITEBOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

6.2.1 BLACKBOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements.

The steps involved in black box test case design are:

- Graph based testing methods

- Equivalence partitioning
- Boundary value analysis
- Comparison testing

6.3 SOFTWARE TESTING STRATEGIES

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason, a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

•Testing begins at the module level and works “outward” toward the integration of the entire computer-based system.

•Different testing techniques are appropriate at different points in time. •The developer of the software and an independent test group conducts testing.

•Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

CHAPTER 7

7.1 CONCLUSION

In conclusion, our project demonstrates the effectiveness of utilizing advanced deep learning techniques, particularly Generative Adversarial Networks (GANs), for image generation and upscaling tasks. By leveraging the capabilities of PyTorch and TensorBoard, we have developed a user-friendly system that streamlines the training process and enhances the quality of generated images. Our automated image enhancement pipeline offers scalable solutions across various industries, reducing manual intervention and improving efficiency. Through extensive testing and validation, we have shown that our system outperforms traditional methods and offers superior results in terms of image quality and realism. Moving forward, the insights gained from this project can be applied to further advancements in image processing and deep learning research, paving the way for innovative solutions in fields such as computer vision, healthcare, and entertainment.

7.2 FUTURE ENHANCEMENTS

Looking ahead, several avenues for future enhancements in our project can be explored. Firstly, we can investigate techniques to further improve the efficiency and performance of our image generation and upscaling models, such as exploring novel architectures or incorporating additional loss functions for better image fidelity. Additionally, enhancing the scalability of our system to handle even larger datasets and more complex models will be beneficial. Furthermore, integrating techniques for domain-specific image enhancement, such as medical imaging or satellite imagery, can extend the applicability of our system to various specialized fields. Moreover, refining the user interface to provide more intuitive controls and visualization tools can enhance user experience and facilitate broader adoption of our system. Lastly, continuous monitoring and adaptation of our system to emerging advancements in deep learning and image processing will ensure that it remains at the forefront of technology and continues to deliver state-of-the-art results.

CHAPTER 8

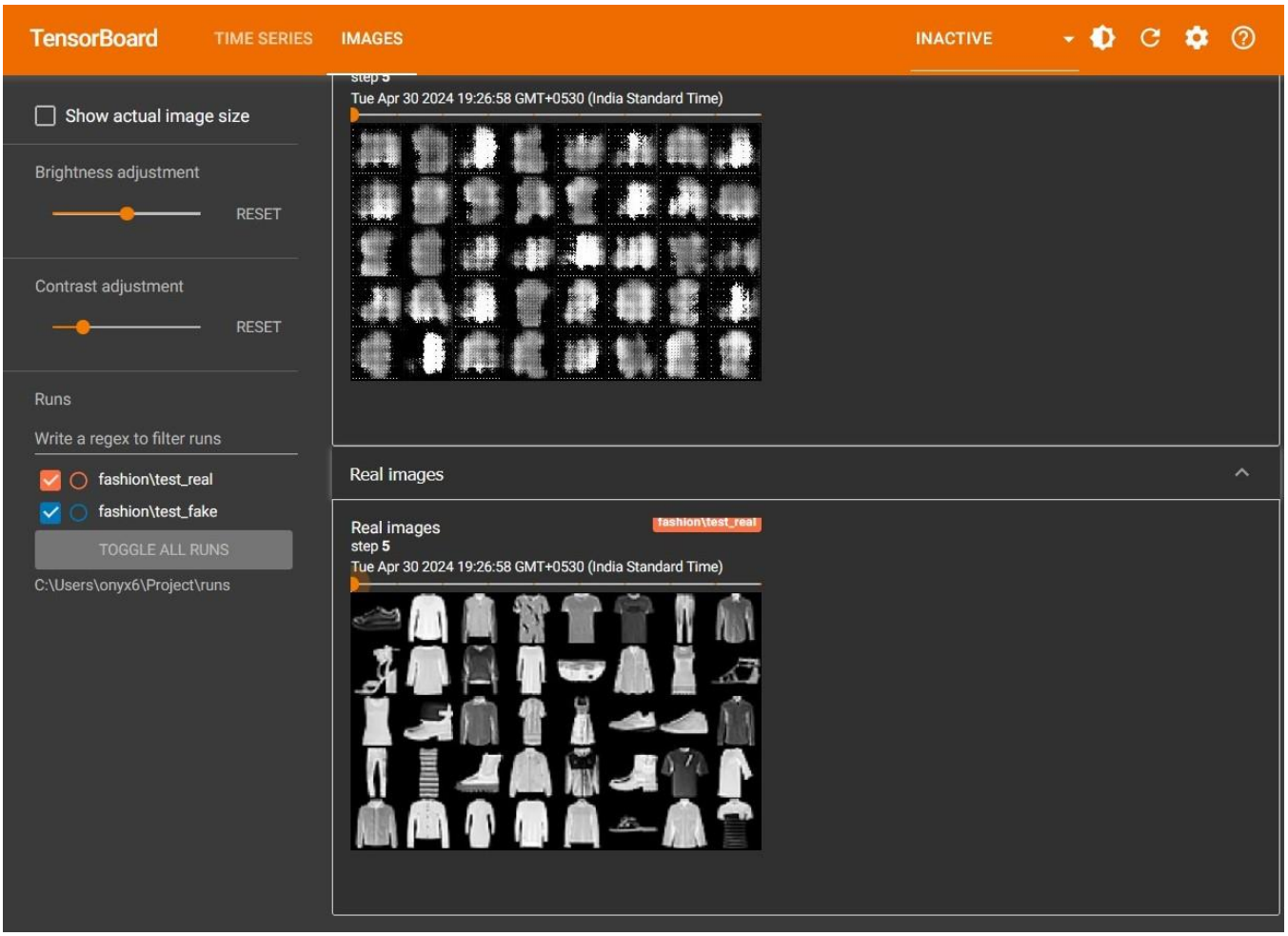
APPENDICES

Image Generation

Fig 8.1 Epoch 1

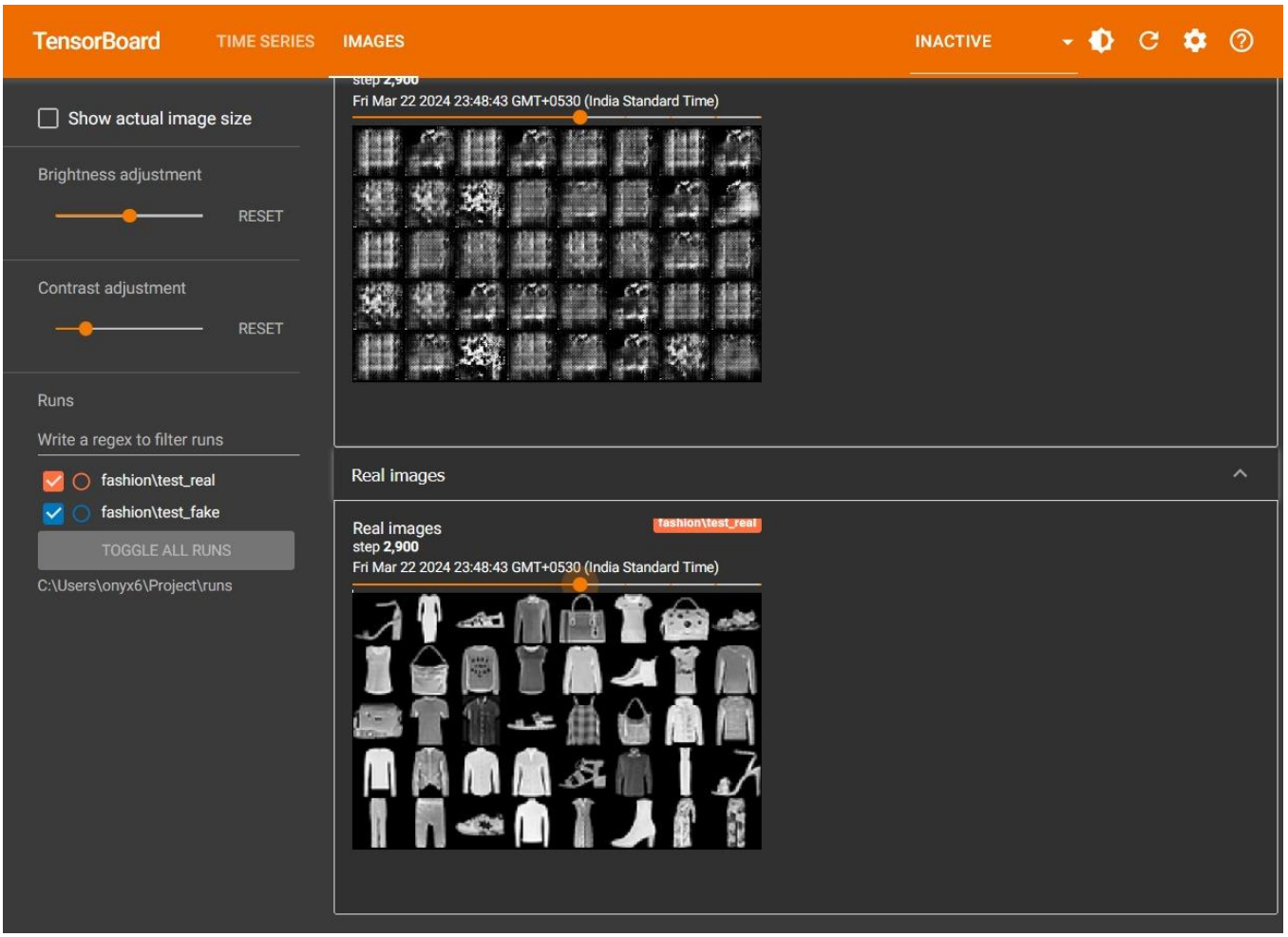
Fig 8.2 Epoch 1000

Fig 8.3. Training using tensorboard



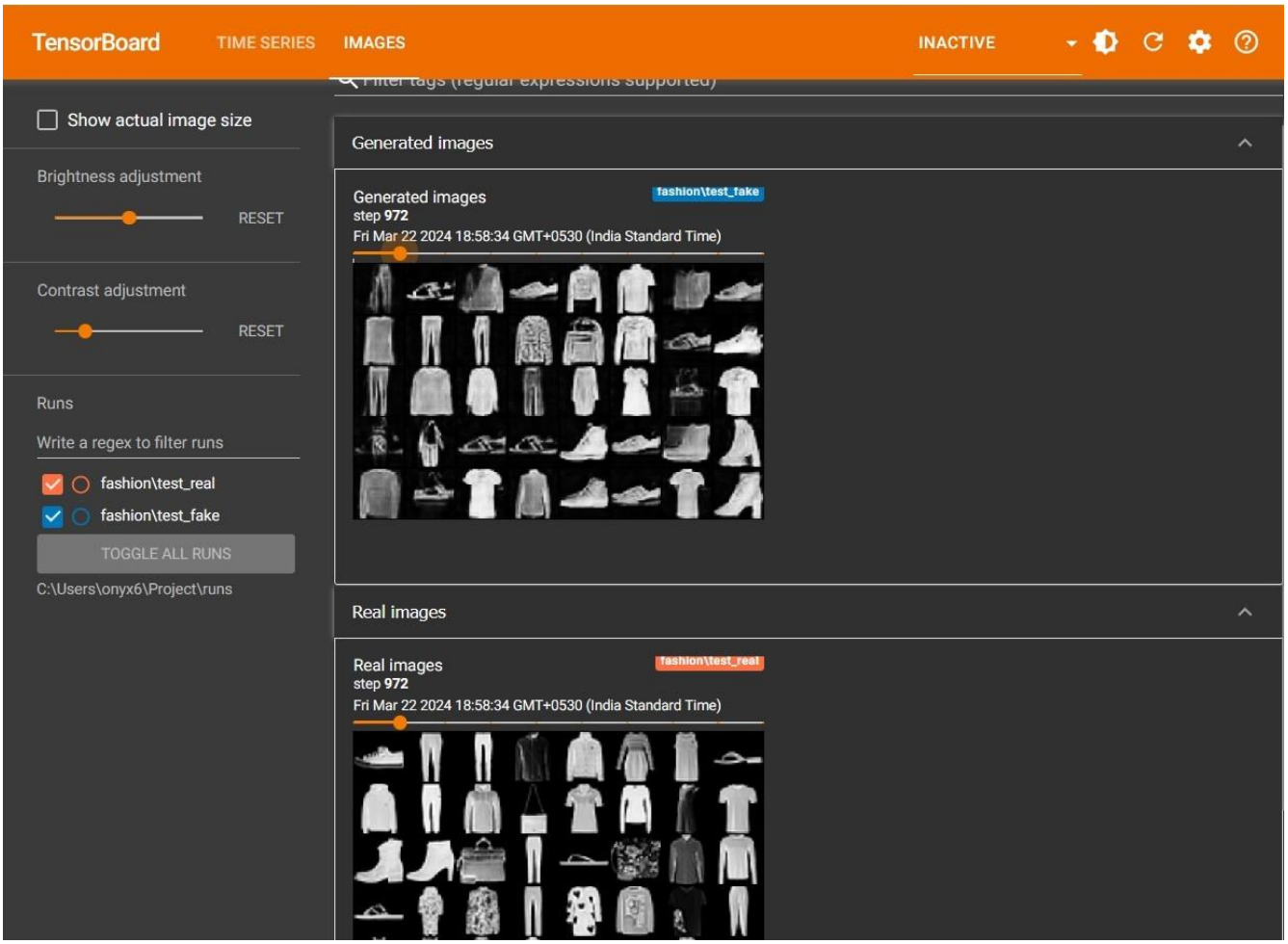
Epoch 1

Fig 8.3. Training using tensorboard



Epoch 20

Fig 8.3. Training using tensorboard



Epoch 1000

Image Upscaling

Before upscaling

After upscaling

CHAPTER 9

IMAGE GENERATION AND UPSCALING USING GENERATIVE ADVERSIAL NETWORK

*M.Ramya M.E[1],Sakthi Murugan V[2] ,Saran Nithish N.A[3] ,Shyam Ganesh J[4] ,

[1]Assistant Professor, Dept of Information Technology, Panimalar Institute of Technology, India,

[234]UG Students, Dept of Information Technology, Panimalar Institute of Technology, India,

[1]Ramyaanitha94@gmail.com ,[2] sakthi270503@gmail.com
,[3]nithishsaran1234@gmail.com ,[4]onyx6569@gmail.com.

ABSTRACT:

Our project centre's on the application of Generative Adversarial Networks (GANs) in image generation and upscaling tasks, employing PyTorch, Tensor Board, CUDA, and cuDNN. GANs, composed of a generator and a discriminator, engage in adversarial training to produce high-fidelity images from random noise inputs. Our research extends to exploring GANs' potential in upscaling low-resolution images while preserving crucial visual details. By harnessing CUDA and cuDNN, we capitalize on their parallel processing capabilities, expediting model training for faster convergence and enhanced performance. Our project underscores the significance of leveraging sophisticated frameworks like PyTorch and visualization tools like Tensor Board for streamlined model development, debugging, and visualization. By advancing image generation and upscaling techniques, our work aims to make significant contributions to diverse fields such as computer vision, medical imaging, and the entertainment industry. Through systematic experimentation and rigorous evaluation, we aim to demonstrate the efficacy of our approach in generating high-quality images and improving the resolution of low-quality ones. Ultimately, our research seeks to not only push the boundaries of image synthesis and enhancement but also to provide practical solutions that can be deployed in real-world applications.

Key words: Generative Adversarial Networks (GANs), PyTorch, TensorBoard, CUDA, Image Upscaling

I. INTRODUCTION:

In our project, we're delving into Generative Adversarial Networks (GANs) using PyTorch to create and enhance images. GANs consist of two parts: a generator and a discriminator, working together to produce lifelike images. We're harnessing the capabilities of PyTorch, a robust deep learning framework, to train our GAN model effectively. Alongside, we're employing TensorBoard, a visualization tool, to monitor and analyze the training process. This helps us understand how well our model is performing and make necessary tweaks. The ability to generate and enhance images has practical applications across many fields, including computer vision, medical imaging, and entertainment. Our project aims to advance image processing techniques, improving image quality and realism. By combining GANs, PyTorch, and TensorBoard, we strive to make significant strides in image generation and enhancement tasks. Ultimately, our goal is to produce high-quality, realistic images that can be used in various applications. from creating immersive digital experiences to assisting medical professionals in diagnosing diseases from medical images. With the power of GANs, PyTorch, and TensorBoard combined, we are confident

in our ability to make meaningful contributions to the field of image processing and artificial intelligence as a whole.The integration of GANs, PyTorch, and TensorBoard facilitates a comprehensive approach to image generation and enhancement. Through meticulous experimentation and iterative refinement, we aim to optimize the architecture and training parameters of our GAN model. PyTorch's dynamic computation graph and automatic differentiation capabilities empower us to experiment with complex network architectures and loss functions, tailoring them to the specific requirements of our task.

The practical implications of our research extend far beyond the confines of academia. In computer vision, our work has the potential to revolutionize applications such as image synthesis, style transfer, and image-to-image translation. By generating high-fidelity images with fine-grained details, we enable more accurate object detection, segmentation, and recognition systems, thereby enhancing the capabilities of autonomous vehicles, surveillance systems, and augmented reality devices. Ultimately, our goal is to democratize access to state-of-the-art image processing techniques and empower practitioners across various domains to harness the power of GANs, PyTorch, and TensorBoard.

II. SYSTEM DESIGN

EXISTING SYSTEM:

The existing system project revolutionizes image enhancement by surpassing traditional methods like interpolation with advanced deep learning techniques, specifically Generative Adversarial Networks (GANs). Unlike conventional GAN implementations, which lack user-friendly interfaces, our project seamlessly integrates with PyTorch and leverages Tensor Board for efficient training visualization. This automation eliminates the need for manual image enhancement techniques, making the process scalable and efficient across various domains. Built on PyTorch, our project ensures scalability and flexibility, enabling seamless integration with cloud-based computing resources for efficient training on extensive datasets.

PROPOSED SYSTEM:

Our system harnesses cutting-edge deep learning techniques, notably Generative Adversarial Networks (GANs), to enhance image quality and realism through intricate pattern learning. With a streamlined user interface and PyTorch integration, it offers intuitive model development and customization. TensorBoard facilitates real-time training visualization, empowering users to optimize performance. Automated image enhancement via GANs eliminates manual intervention, providing scalable solutions across domains. Designed for scalability, our system seamlessly integrates with cloud-based resources, enabling efficient handling of large datasets and complex models through distributed computing. Building upon cutting-edge deep learning techniques, our proposed system capitalizes on the transformative power of Generative Adversarial Networks (GANs) to elevate image quality and realism through sophisticated pattern learning mechanisms. By harnessing the intricate interplay between a generator and a discriminator, our system excels in synthesizing visually appealing and contextually coherent images, surpassing the limitations of traditional image enhancement methods.

In summary, our proposed system represents a paradigm shift in image enhancement technology, leveraging cutting-edge deep learning techniques, intuitive user interfaces, and scalable computing infrastructures to redefine the boundaries of what is possible. By democratizing access to state-of-the-art image processing capabilities, we aim to catalyze innovation and empower individuals and organizations to unleash their creativity and realize their vision in the digital realm.

III. ARCHITECTURE DIAGRAM:

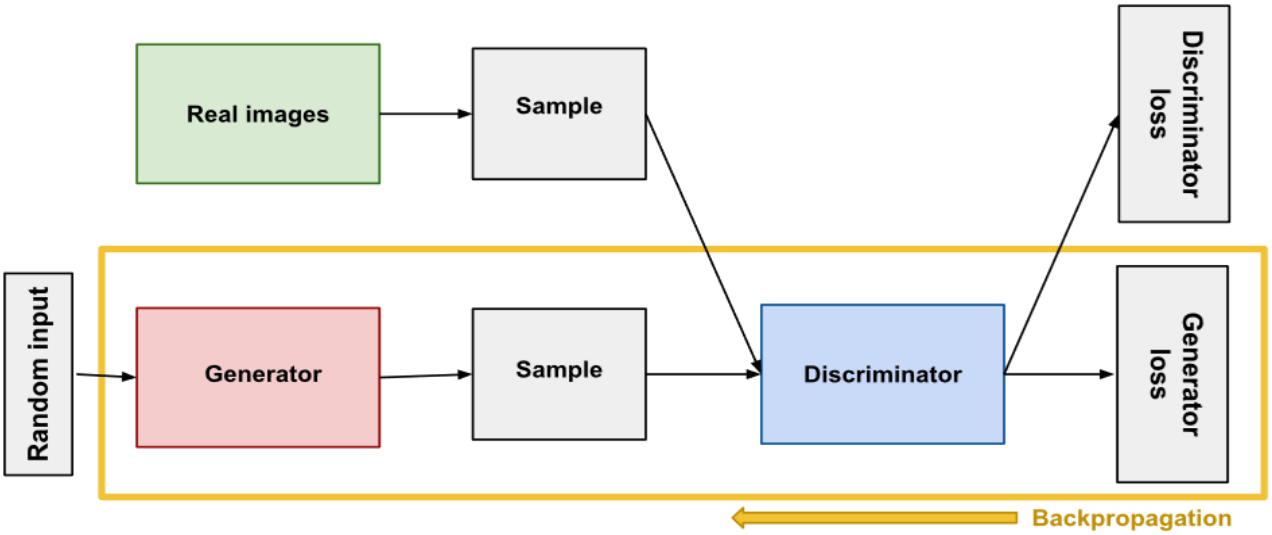


Fig.1.Architecture Diagram for Deep convolutional Generative Adversarial Network

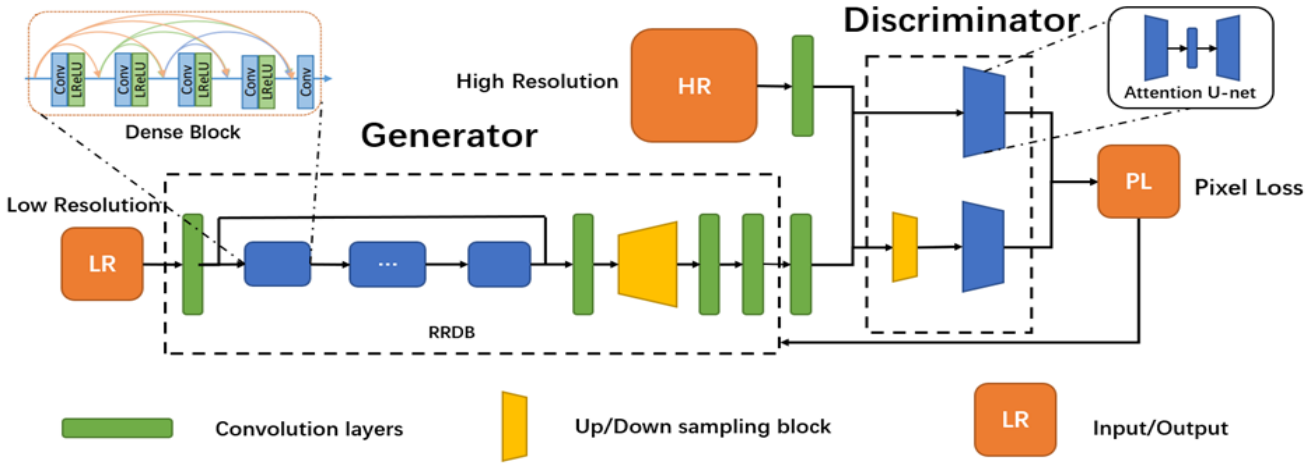


Fig.2.Architecture Diagram for Enhanced Super-Resolution Generative Adversarial Network

The architecture diagram for a deep convolutional generative adversarial network (DCGAN) typically consists of two main components: the generator and the discriminator. The generator comprises multiple layers of transposed convolutional and batch normalization operations, transforming a latent input vector into synthetic images. Conversely, the discriminator consists of convolutional layers followed by batch normalization and LeakyReLU activation functions, aiming to distinguish between real and generated images. The latent input vector serves as the seed for image generation in the generator, while the discriminator receives both real and

generated images, providing feedback to the generator for improvement through adversarial training. This adversarial process encourages the generator to produce increasingly realistic images while simultaneously enhancing the discriminator's ability to differentiate between real and fake samples. Overall, the DCGAN architecture leverages deep convolutional neural networks to generate high-quality images through adversarial learning.

ESRGAN (Enhanced Super-Resolution Generative Adversarial Network) enhances image resolution through adversarial training. Unlike traditional GANs, ESRGAN's generator employs a deep residual network (ResNet) with skip connections for gradient propagation. Perceptual loss, integrated into the loss function, measures the perceptual similarity between generated and ground truth images, alongside adversarial and content losses. This encourages the generator to produce high-fidelity images with enhanced structural details. ESRGAN achieves state-of-the-art results in image super-resolution, generating visually appealing outputs with reduced artifacts. The adversarial training process ensures the generated images are perceptually indistinguishable from real high-resolution images, as determined by the discriminator.

IV. PROJECT MODULES:

There are Five modules:

- Setting up Environment and Dependencies
- Data Preparation and Preprocessing
- Training the Generative Adversarial Network (GAN)
- Image Generation and Upscaling
- Evaluation and Performance Analysis

Setting up Environment and Dependencies:

Installation of necessary libraries and frameworks such as PyTorch, CUDA, cuDNN, and TensorBoard.

Data Preparation and Preprocessing:

Acquisition and preprocessing of image datasets suitable for training GAN models. Data augmentation techniques to enhance the diversity and quality of training data.

Training the Generative Adversarial Network (GAN):

Implementation and training of the GAN architecture using PyTorch. Fine-tuning hyperparameters, optimizing loss functions, and monitoring training progress using TensorBoard.

Image Generation and Upscaling:

Deployment of the trained GAN model for generating high-fidelity images from random noise inputs. Upscaling of low-resolution images while preserving essential visual details through the GAN framework.

Evaluation and Performance analysis:

Quantitative and qualitative assessment of generated images using metrics. Visual inspection of upscaled images to evaluate the effectiveness of the upscaling process.

V.ALGORITHM AND TECHNIQUES:

Deep Convolutional GANs (DCGANs):

- **Convolutional Architecture**: DCGANs employ deep convolutional neural networks (CNNs) as both the generator and discriminator. This architecture allows the models to effectively capture spatial dependencies and hierarchical features in images, leading to better performance in image generation tasks.
- **Stable Training**: DCGANs introduce architectural constraints and best practices to ensure stable training. These include using strided convolutions in the discriminator and fractional-strided convolutions (also known as transposed convolutions) in the generator, as well as batch normalization in both networks. These techniques help mitigate issues like mode collapse and vanishing gradients.
- **No Fully Connected Layers**: Unlike traditional GAN architectures, DCGANs typically avoid fully connected layers in favor of convolutional and deconvolutional layers. This decision reduces the risk of overfitting and enables the model to handle images of arbitrary sizes without requiring resizing.
- **Image Generation**: In DCGANs, the generator network takes random noise vectors as input and gradually upsamples them through convolutional and deconvolutional layers to generate realistic images. The discriminator network, on the other hand, learns to distinguish between real images from the dataset and fake images produced by the generator.

PyTorch Framework:

- PyTorch serves as the primary deep learning framework for implementing the GAN model and associated algorithms. PyTorch provides a flexible and efficient platform for designing and training neural networks, facilitating rapid experimentation and prototyping of complex architectures.

TensorBoard Visualization:

- TensorBoard is utilized for real-time visualization and analysis of the training process of the GAN model. This tool enables monitoring of key metrics such as loss functions, learning curves, and image reconstructions, providing insights into the model's behavior and aiding in the optimization of training strategies.

Evaluation Metrices:

Human Evaluation: Human evaluation involves subjective assessment by human observers to evaluate the quality and realism of generated images. This can be done through surveys, preference tests, or expert judgment to provide qualitative feedback on the visual quality of the generated images.

VI.IMPLEMENTATION:

ImageGeneration.ipynb:

```
import torch
from torch import nn
class Generator(nn.Module):
    def __init__(self, noise_channels, image_channels, features):
        super(Generator, self).__init__()

        # define the model
        self.model = nn.Sequential(
            # Transpose block 1
            nn.ConvTranspose2d(noise_channels, features*16, kernel_size=4, stride=1, padding=0),
            nn.ReLU(),

            # Transpose block 2
            nn.ConvTranspose2d(features*16, features*8, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*8),
            nn.ReLU(),

            # Transpose block 3
            nn.ConvTranspose2d(features*8, features*4, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*4),
            nn.ReLU(),
  # Transpose block 4
            nn.ConvTranspose2d(features*4, features*2, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*2),
            nn.ReLU(),

            # Last transpose block (different)
            nn.ConvTranspose2d(features*2, image_channels, kernel_size=4, stride=2, padding=1),
            nn.Tanh(),
class Discriminator(nn.Module):
    def __init__(self, image_channels, features):
        super(Discriminator, self).__init__()

        # define the model
        self.model = nn.Sequential(
            # define the first Conv block
            nn.Conv2d(image_channels, features, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2),

            # Conv block 2
            nn.Conv2d(features, features*2, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*2),
            nn.LeakyReLU(0.2),

            # Conv block 3
            nn.Conv2d(features*2, features*4, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*4),
```

```
            nn.LeakyReLU(0.2),

            # Conv block 4
            nn.Conv2d(features*4, features*8, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(features*8),
            nn.LeakyReLU(0.2),

            # Conv block 5 (different)
            nn.Conv2d(features*8, 1, kernel_size=4, stride=2, padding=0),
            nn.Sigmoid(),
        )

    # Generate fake images
    with torch.no_grad():
        # Assuming fixed_noise is your fixed input noise tensor
        fixed_noise = torch.randn(batch_size, noise_channels, 1, 1).to(device)
        fake_images = generator(fixed_noise)

    # Now fake_images contains the generated fake images
    import matplotlib.pyplot as plt
    import torchvision.utils

    # Convert the fake images to a grid
    fake_images_grid = torchvision.utils.make_grid(fake_images.cpu(), nrow=8, normalize=True)

    # Convert the PyTorch tensor to a NumPy array and transpose the dimensions
    fake_images_grid_np = fake_images_grid.permute(1, 2, 0).numpy()

    # Display the fake images grid using matplotlib
    plt.figure(figsize=(10, 10))
    plt.imshow(fake_images_grid_np)
    plt.axis('off')
    plt.show()
```

ImageUpscaling.ipynb:

Transer_RRDB_models.py

```
import os
import torch
import RRDBNet_arch as arch
pretrained_net = torch.load('./models/RRDB_ESRGAN_x4.pth')
save_path = './models/RRDB_ESRGAN_x4.pth'
crt_model = arch.RRDBNet(3, 3, 64, 23, gc=32)
crt_net = crt_model.state_dict()
load_net_clean = {}
for k, v in pretrained_net.items():
    if k.startswith('module.'):
        load_net_clean[k[7:]] = v
    else:
        load_net_clean[k] = v
pretrained_net = load_net_clean
print('###################################\n')
tbd = []
```

```
for k, v in crt_net.items():
    tbd.append(k)
crt_net['trunk_conv.weight'] = pretrained_net['model.1.sub.23.weight']
crt_net['upconv1.weight'] = pretrained_net['model.3.weight']
crt_net['upconv2.weight'] = pretrained_net['model.6.weight']
crt_net['HRconv.weight'] = pretrained_net['model.8.weight']
crt_net['conv_last.weight'] = pretrained_net['model.10.weight']
crt_net['conv_last.bias'] = pretrained_net['model.10.bias']
torch.save(crt_net, save_path)
print('Saving to ', save_path)
```

RRDBNet_arch.py

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

def make_layer(block, n_layers):
    layers = []
    for _ in range(n_layers):
        layers.append(block())
    return nn.Sequential(*layers)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 = self.lrelu(self.conv2(torch.cat((x, x1), 1)))
        x3 = self.lrelu(self.conv3(torch.cat((x, x1, x2), 1)))
        x4 = self.lrelu(self.conv4(torch.cat((x, x1, x2, x3), 1)))
        x5 = self.conv5(torch.cat((x, x1, x2, x3, x4), 1))
        return x5 * 0.2 + x
class RRDBNet(nn.Module):
    def __init__(self, in_nc, out_nc, nf, nb, gc=32):
        super(RRDBNet, self).__init__()
        RRDB_block_f = functools.partial(RRDB, nf=nf, gc=gc)

        self.conv_first = nn.Conv2d(in_nc, nf, 3, 1, 1, bias=True)
        self.RRDB_trunk = make_layer(RRDB_block_f, nb)
        self.trunk_conv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        #### upsampling
        self.upconv1 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.upconv2 = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.HRconv = nn.Conv2d(nf, nf, 3, 1, 1, bias=True)
        self.conv_last = nn.Conv2d(nf, out_nc, 3, 1, 1, bias=True)

        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)

    def forward(self, x):
        fea = self.conv_first(x)
        trunk = self.trunk_conv(self.RRDB_trunk(fea))
        fea = fea + trunk

        fea = self.lrelu(self.upconv1(F.interpolate(fea, scale_factor=2, mode='nearest')))
```

```
        fea = self.lrelu(self.upconv2(F.interpolate(fea, scale_factor=2, mode='nearest')))
        out = self.conv_last(self.lrelu(self.HRconv(fea)))
        return out
```

Test.py

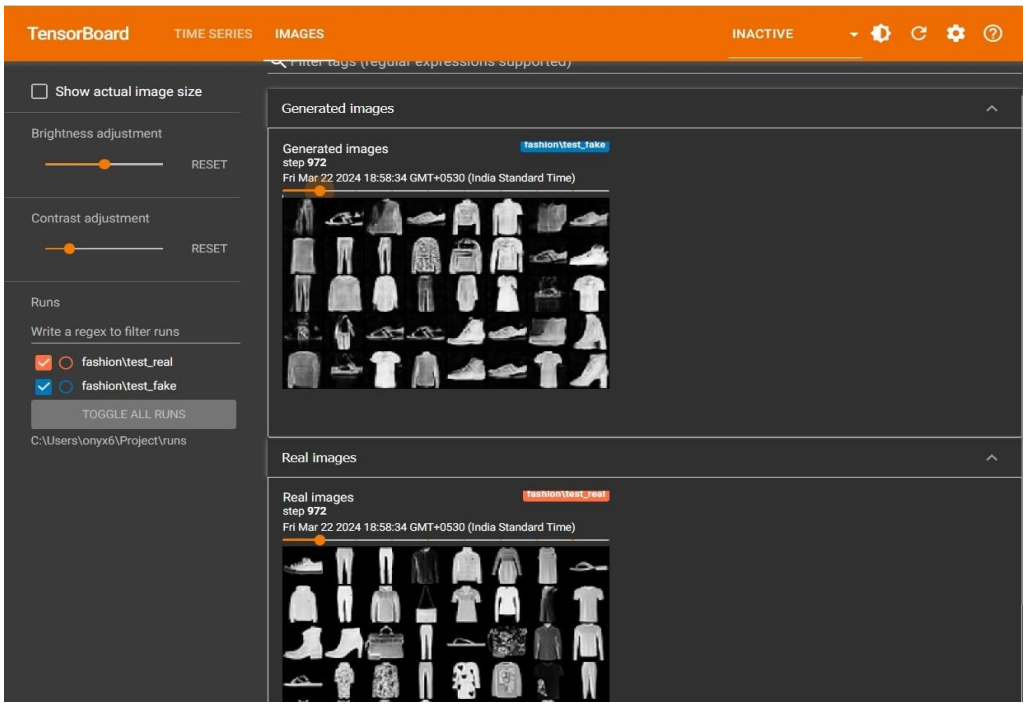
```
import os.path as osp
import glob
import cv2
import numpy as np
import torch
import RRDBNet_arch as arch

model_path = 'models/RRDB_ESRGAN_x4.pth'  # models/RRDB_ESRGAN_x4.pth OR
models/RRDB_PSNR_x4.pth
device = torch.device('cuda')  # if you want to run on CPU, change 'cuda' -> cpu
# device = torch.device('cpu')

test_img_folder = 'LR/*'
class RRDB(nn.Module):
    '''Residual in Residual Dense Block'''
    def __init__(self, nf, gc=32):
        super(RRDB, self).__init__()
        self.RDB1 = ResidualDenseBlock_5C(nf, gc)
        self.RDB2 = ResidualDenseBlock_5C(nf, gc)
        self.RDB3 = ResidualDenseBlock_5C(nf, gc)


model = arch.RRDBNet(3, 3, 64, 23, gc=32)
model.load_state_dict(torch.load(model_path), strict=True)
model.eval()
model = model.to(device)

print('Model path {:s}. \nTesting...'.format(model_path))

idx = 0
for path in glob.glob(test_img_folder):
    idx += 1
    base = osp.splitext(osp.basename(path))[0]
    print(idx, base)
# read images
img = cv2.imread(path, cv2.IMREAD_COLOR)
img = img * 1.0 / 255
img = torch.from_numpy(np.transpose(img[:, :, [2, 1, 0]], (2, 0, 1))).float()
img_LR = img.unsqueeze(0)
img_LR = img_LR.to(device)
with torch.no_grad():
        output = model(img_LR).data.squeeze().float().cpu().clamp_(0, 1).numpy()
output = np.transpose(output[[2, 1, 0], :, :], (1, 2, 0))
output = (output * 255.0).round()
cv2.imwrite('results/{:s}_rlt.png'.format(base), output)
```

VII. Output:

Image Generation:

Fig 1: Epoch 1000

Fig 2: Training using tensorboard



Epoch 1000

Image Upscaling:

Before upscaling

After upscaling

VIII. CONCLUSION:

In conclusion, our project demonstrates the effectiveness of utilizing advanced deep learning techniques, particularly Generative Adversarial Networks (GANs), for image generation and upscaling tasks. By leveraging the capabilities of PyTorch and TensorBoard, we have developed a user-friendly system that streamlines the training process and enhances the quality of generated images. Our automated image enhancement pipeline offers scalable solutions across various industries, reducing manual intervention and improving efficiency. Through extensive testing and validation, we have shown that our system outperforms traditional methods and offers superior results in terms of image quality and realism. Moving forward, the insights gained from this project can be applied to further advancements in image processing and deep learning research, paving the way for innovative solutions in fields such as computer vision, healthcare, and entertainment.

REFERENCES:

1. Xiao, J., & B, X. (2024). "Model-Guided Generative Adversarial Networks for Unsupervised Fine-Grained Image Generation." [IEEE], [Volume(I26)], pp. [12].
2. Cheng, W., Zhang, S., & Lin, Y. (2023). "Study on the Adversarial Sample Generation Algorithm Based on Adversarial Quantum Generation Adversarial Network." [3rd International Symposium on Computer Technology and Information Science], pp. [6].
3. Wang, M., Zhou, Y., & Xu, K. (2023). "MoE-ESRGAN: A Super-Resolution Network for Multi-degraded Chinese Painting Images." In Proceedings of the 2023 3rd International Conference on Electronic Information Engineering and Computer (EIECT)
4. Hostin, M.-A., Sivtsov, V., Attarian, S., Bendahan, D., & Bellemare, M.-E. (2023). "CONTEXT-GAN: Controllable Context Image Generation Using GANs." In 2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI).
5. Sharma, P., Mohanram, B., & Vijayashree, J. (2022). "Image Enhancement using ESRGAN for CNN based X-Ray Classification." In Proceedings of the 2022 5th International Conference on Contemporary Computing and Informatics (IC3I)
6. Song, L., Li, Y., & Lu, N. (2022). "ProfileSR-GAN: A GAN Based Super-Resolution Method for Generating High-Resolution Load Profiles." [IEEE], [Volume(13)], pp. [12].
7. Zhou, J. (2022). "Research on Generative Adversarial Networks and Their Applications in Image Generation." In 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA).
8. Liu, J., & Chandrasiri, N. P. (2022). "C-ESRGAN: Synthesis of super-resolution images by image classification." In Proceedings of the 2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)
9. Rakotonirina, N. C., & Rasoanaivo, A. (2020). "ESRGAN+: Further Improving Enhanced Super-Resolution Generative Adversarial Network." In Proceedings of ICASSP 2020.
10. Gonog, L., & Zhou, Y. (2019). "A Review: Generative Adversarial Networks." In 2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA).

CHAPTER 10

REFERENCES

1] Xiao, J., & B, X. (2024). "Model-Guided Generative Adversarial Networks for Unsupervised Fine-Grained Image Generation." [IEEE], [Volume(I26)], pp. [12]

2] Cheng, W., Zhang, S., & Lin, Y. (2023). "Study on the Adversarial Sample Generation Algorithm Based on Adversarial Quantum Generation Adversarial Network." [3rd International Symposium on Computer Technology and Information Science], pp. [6].

3] Wang, M., Zhou, Y., & Xu, K. (2023). "MoE-ESRGAN: A Super-Resolution Network for Multi-degraded Chinese Painting Images." In Proceedings of the 2023 3rd International Conference on Electronic Information Engineering and Computer (EIECT).

4] Hostin, M.-A., Sivtsov, V., Attarian, S., Bendahan, D., & Bellemare, M.-E. (2023). "CONTEXT-GAN: Controllable Context Image Generation Using GANs." In 2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI).

5] Sharma, P., Mohanram, B., & Vijayashree, J. (2022). "Image Enhancement using ESRGAN for CNN based X-Ray Classification." In Proceedings of the 2022 5th International Conference on Contemporary Computing and Informatics (IC3I)

6] Song, L., Li, Y., & Lu, N. (2022). "ProfileSR-GAN: A GAN Based Super-Resolution Method for Generating High-Resolution Load Profiles." [IEEE], [Volume(13)], pp. [12]..

7] Zhou, J. (2022). "Research on Generative Adversarial Networks and Their Applications in Image Generation." In 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA).

8] Liu, J., & Chandrasiri, N. P. (2022). "C-ESRGAN: Synthesis of super-resolution images by image classification." In Proceedings of the 2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS).

9] Rakotonirina, N. C., & Rasoanaivo, A. (2020). "ESRGAN+: Further Improving Enhanced Super-Resolution Generative Adversarial Network." In Proceedings of ICASSP 2020.

10] Gonog, L., & Zhou, Y. (2019). "A Review: Generative Adversarial Networks." In 2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA).