REAL-TIME WEB-BASED SAR IMAGE COLORIZATION USING DEEP LEARNING

PROJECT WORK2 (REVIEW3)

Submitted by

SHYAM KUMAR A	**212221230098**
SANJAY S	**212221243002**
SRINIVAS U	**212221230108**

in partial fulfillment for the

award of the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

SAVEETHA ENGINEERING COLLEGE, THANDALAM

An Autonomous Institution Affiliated to

ANNA UNIVERSITY - CHENNAI 600 025

NOVEMBER 2024

SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Affiliated to Anna University | Approved by AICTE

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project report **" REAL-TIME WEB-BASED SAR IMAGE COLORIZATION USING DEEP LEARNING "** is the bonafide work of **SHYAM KUMAR A (212221230098),** who carried out this project work under my supervision.

SIGNATURE

Dr. Lavanya G

Associate Professor

SUPERVISOR

Dept of Artificial Intelligence and DataScience

Saveetha Engineering College, Thandalam, Chennai 602105

SIGNATURE

Dr. Karthi Govindharaju, M.E., Ph.D.,

Professor

HEAD OF THE DEPARTMENT

Dept of Artificial Intelligence and DataScience

Saveetha Engineering College, Chennai 602105.

DATE OF THE VIVA VOCE EXAMINATION: ……

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to our esteemed Founder President **Dr. N. M. Veeraiyan**, our President **Dr. Saveetha Rajesh**, our Director **Dr. S. Rajesh**, and the entire management team for providing the essential infrastructure.

I extend my sincere appreciation to our principal, **Dr. V. Vijaya Chamundeeswari, M.Tech., Ph.D.,** for creating a supportive learning environment for this project.

I am very thankful to our Dean of ICT, **Mr. Obed Otto,** M.E., for facilitating a conducive atmosphere that allowed me to complete my project successfully.

My thanks go to **Dr. Karthi Govindharaju, M.E., Ph.D.,**, Professor and Head of the Department of Artificial Intelligence and Data Science at Saveetha Engineering College, for his generous support and for providing the necessary resources for my project work.

I would also like to express my profound gratitude to my Supervisor, **Dr. Lavanya G**, Associate Professor and my Project Coordinator **Dr. N.S. Gowri Ganesh**, Associate Professor at Saveetha Engineering College, for their invaluable guidance, suggestions, and constant encouragement, which were instrumental in the successful completion of this project. Their timely support and insights during the review process were greatly appreciated.

I am grateful to all my college faculty, staff, and technicians for their cooperation throughout the project. Finally, I wish to acknowledge my loving parents, friends, and well-wishers for their encouragement in helping me achieve this milestone.

ABSTRACT

Satellite imagery is a vital tool in applications such as environmental monitoring, disaster management, and urban planning. However, optical satellite images are often obstructed by weather conditions like cloud cover and poor lighting, making them unreliable in certain scenarios. Synthetic Aperture Radar (SAR) imaging overcomes these limitations by capturing high-resolution images regardless of weather conditions. However, SAR images are difficult to interpret due to speckle noise and a lack of color information. To address this, deep learning techniques have been employed to convert SAR images into optical images, enhancing their usability. This study presents a SAR-to-optical image translation model using Generative Adversarial Networks (GANs), specifically utilizing a UNet-based generator and a PatchGAN discriminator to produce high-quality optical images.

The proposed system consists of both a deep learning model and a simple front-end interface. The model is trained using Sentinel-1 SAR images and Sentinel-2 optical images, ensuring accurate translation. The training process involves preprocessing SAR images, normalizing data, and applying augmentation techniques to improve generalization. The GAN model utilizes adversarial loss and L1 loss to refine image clarity and structural accuracy. To make the system user-friendly, a front-end web interface has been developed where users can upload SAR images, and the model generates and displays the corresponding colorized optical images. The performance of the model is evaluated using metrics such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM), demonstrating its effectiveness in producing visually accurate optical images.

This SAR-to-optical translation system has significant implications in remote sensing and geospatial analysis, facilitating better visualization and interpretation of satellite data. The front-end interface makes it accessible for users without technical expertise, allowing seamless conversion of SAR images to optical images. The generated optical images can be utilized for applications like vegetation analysis, disaster response, and land classification, aiding in data-driven decision-making. Future improvements will focus on enhancing image resolution, optimizing computational efficiency, and integrating additional features such as automated analysis tools for real-time geospatial insights.

TABLE OF CONTENTS

CHAPTER.NO			TITLE	PAGE NUMBER
1.			**Introduction**	
	1.1		Overview of The Project	1
2.			**Problem Definition**	3
3.			**Literature Review**	
4.			**System Analysis**	
	4.1		Existing System	11
	4.2		Disadvantages of Existing System	12
	4.3		Proposed System	12
	4.4		Advantages of Existing System	14
	4.5		Feasibility Study	14
	4.6		Hardware Environment	14
	4.7		Software Environment	14
	4.8		Technology Used	14
		4.8.1	Python	15
		4.8.2	Deep Learning	15
5.			**Scope of the Project**	16
6.			**System Design**	
	6.1		Entity-Relationship Diagram	17
	6.2		Data Flow Diagram	18

	6.3		UML Diagram	
		6.3.1	Use Case Diagram	19
		6.3.2	Class Diagram	20
		6.3.3	Sequence Diagram	21
7.			**System Architecture**	
	7.1		Architecture Diagram	22
	7.2		Algorithm	23
8.			**System Implementation**	
	8.1		Module 1: Data Collection and Implementation	25
	8.2		Module 2: Data Preprocessing	26
	8.3		Module 3: Prediction Of Output	27
9.			**System Testing**	29
10.			**Conclusion And Future Enhancement**	
	10.1		Conclusion	30
	10.2		Future Enhancement	31
11.			**Apendix-1 Sample Code**	32
12.	12.1		**Apendix-2 Sample Output**	33
	12.2		**Apendix-3 Accuracy And Matrix**	36
13.			**References**	40

1. INTRODUCTION

Satellite imagery plays a crucial role in various fields, including environmental monitoring, urban planning, and disaster management. However, traditional optical satellite images often suffer from limitations such as cloud cover, atmospheric disturbances, and low visibility in night-time conditions. Synthetic Aperture Radar (SAR) imaging provides a reliable alternative, as it can capture high-resolution images regardless of weather conditions or lighting. Despite these advantages, SAR images are difficult to interpret due to their grayscale representation and the presence of speckle noise. Converting SAR images into optical images is essential for better visualization and usability, enabling more effective analysis by researchers, policymakers, and professionals in geospatial sciences.

Recent advancements in deep learning have enabled the transformation of SAR images into optical images with improved accuracy and detail. Generative Adversarial Networks (GANs) have proven to be effective in such image-to-image translation tasks by learning complex patterns and textures. In this study, we implement a GAN-based approach for SAR-to-optical image translation, utilizing a UNet-based generator to preserve fine details and a PatchGAN discriminator to enhance image quality. By training the model on high-quality Sentinel-1 SAR and Sentinel-2 optical images, we ensure accurate and realistic colorization. The integration of this model with a simple front-end interface allows users to upload SAR images and receive high-quality optical images as output, making the system accessible and practical for a wide range of applications.

The objective of this project is to bridge the gap between SAR imaging and optical imagery by leveraging deep learning techniques. The proposed system not only improves SAR image interpretation but also provides an easy-to-use web-based platform for users to process SAR data without extensive technical knowledge. The converted optical images can be applied in numerous fields, including agriculture, disaster response, and urban development, enhancing decision-making processes. This report provides an in-depth analysis of the methodology, system architecture, and performance evaluation of our SAR-to-optical translation model, highlighting its potential for real-world applications.

2. PROBLEM DEFINITION

Satellite imaging has become an essential tool for monitoring and analyzing the Earth's surface. Among the different types of satellite imagery, Synthetic Aperture Radar (SAR) images are widely used due to their ability to penetrate clouds and capture data in all weather conditions and at any time of the day. However, a major drawback of SAR images is their grayscale representation, which makes interpretation difficult, especially for users unfamiliar with radar-based imaging. Unlike optical images, which provide natural color representation, SAR images require specialized expertise to analyze and extract meaningful insights. This limitation makes it challenging for non-experts to utilize SAR data effectively for applications such as land use monitoring, disaster management, and environmental assessment.

To address this challenge, researchers have explored various image translation techniques to convert SAR images into optical images. Traditional image processing methods often fail to capture the complex textures and patterns in SAR images, leading to poor-quality outputs. Moreover, manual interpretation of SAR images is time-consuming and requires expert knowledge, making it inaccessible to a broader audience. There is a need for an automated and efficient approach that can accurately convert SAR images into optical images while preserving important details. Deep learning techniques, particularly Generative Adversarial Networks (GANs), have shown promising results in image-to-image translation tasks, offering a potential solution to this problem.

In addition to addressing the technical challenges of SAR-to-optical conversion, this project also focuses on accessibility and usability. Many existing SAR processing tools are complex and require advanced knowledge of remote sensing techniques. To make SAR-to-optical conversion more accessible, we have developed a simple front-end interface where users can easily upload SAR images and receive corresponding optical images. This streamlined approach eliminates the need for extensive pre-processing and manual intervention, allowing individuals across various domains—such as agriculture, urban planning, and disaster response—to leverage satellite imagery effectively. Our project not only enhances SAR image interpretability but also democratizes satellite data analysis by making it more accessible to non-experts.

3.LITERATURE SURVEY SUMMARY

S.NO	Research	Technique	Features Used	Domain	Disadvantage / Advantage	Future Direction
1.	A benchmarking protocol for SAR colorization: From regression to deep learning approaches (Kangqing	Neural Networks	Regression and deep learning-based SAR colorization	Remote Sensing	- Advantage: Benchmarks traditional and deep learning methods for SAR colorization.	A benchmarking protocol for SAR colorization: From regression to deep learning approaches
2.	Image Colorization (Saurabh Sunil Gaikwad, 2024)	Machine Learning and Deep Learning	Various image datasets, CNN-based models	Computer Vision	- Advantage: Provides a broad perspective on image colorization techniques.	Image Colorization (Saurabh Sunil Gaikwad, 2024)
3.	- Disadvantage: Limited focus on SAR-specific colorization.	Enhance models specifically for grayscale images like SAR using GANs and transformers.				- Disadvantage: Limited focus on SAR-specific colorization.
4.	A Study on the Improvement of SAR Image Colorization Performance Using CUT and SPatchGAN)	CUT and SPatchGAN-based GAN models	Patch-based discriminator for realistic colorization	Remote Sensing	- Advantage: Improves SAR colorization realism using adversarial training.	A Study on the Improvement of SAR Image Colorization Performance Using CUT and SPatchGAN Discriminator (Seung-Min Shin et al., 2023)

4. SYSTEM ANALYSIS

4.1 EXISTING SYSTEM

Synthetic Aperture Radar (SAR) images are typically represented in grayscale, as SAR sensors capture intensity information rather than color. These grayscale images are widely used in various applications such as remote sensing, military surveillance, disaster management, and environmental monitoring. However, the lack of color in SAR images makes interpretation difficult, requiring specialized expertise to extract meaningful information. Traditionally, SAR image analysis relies on manual interpretation, where experts analyze intensity variations to identify terrain features, water bodies, and man-made structures. This process is time-consuming, labor-intensive, and prone to human error.

To enhance SAR image visualization, some traditional methods attempt to use false-color mapping techniques, where different intensity values are mapped to predefined color palettes. While this approach can improve visual appeal, it does not add meaningful spectral information to the image, limiting its effectiveness. Another method involves fusing SAR images with optical images captured from other sensors. Although this fusion can provide better interpretability, it depends on the availability of high-quality optical data, which is often limited due to cloud cover, night-time conditions, or other environmental constraints.

Recent advancements in machine learning and deep learning have introduced new approaches for SAR image colorization. Some deep learning-based methods utilize convolutional neural networks (CNNs) or generative adversarial networks (GANs) to generate pseudo-color images from SAR data. However, these methods often rely on offline processing, requiring significant computational resources and long processing times. Additionally, many existing approaches struggle with generalization, meaning that models trained on specific datasets may not perform well on new or unseen SAR images.

Another major limitation of the existing system is the lack of real-time processing. Most SAR colorization techniques are implemented as offline processes, where images are collected, preprocessed, and then passed through a model for colorization. This delayed processing is not suitable for real-time applications such as disaster response, where rapid visualization of SAR images can aid in decision-making.

4.1.1 Disadvantages of Existing System

In the traditional approach, SAR image interpretation relies heavily on manual analysis by experts who use grayscale intensity variations to infer terrain features. Analysts compare SAR images with optical images or reference datasets to identify water bodies, vegetation, urban areas, and other land types. This process is time-consuming, requires extensive training, and is prone to human error. Additionally, manual interpretation lacks consistency, as different experts may derive different insights from the same image. Various software tools exist to enhance SAR images through filtering and contrast adjustments, but they do not provide natural colorization, making it difficult for non-experts to analyze and utilize SAR data effectively.

Some existing methods use basic machine learning models or statistical techniques to approximate colorization, but they often lack accuracy and fail to generalize across diverse landscapes. Conventional image fusion techniques attempt to blend SAR images with available optical images, but these approaches struggle when optical images are unavailable due to cloud cover or night-time conditions. As a result, the demand for an automated, high-precision SAR colorization system remains unmet, highlighting the need for a deep learning-based approach that can generate realistic colorized SAR images with minimal human intervention.

4.1.2 Proposed System

The proposed system aims to enhance the interpretation of SAR images by implementing a deep learning-based colorization technique. Unlike traditional grayscale SAR images, which require expert analysis, our system utilizes convolutional neural networks (CNNs) or generative adversarial networks (GANs) to automatically generate colorized outputs. This approach enables users to easily analyze and interpret SAR data without extensive technical expertise. The system is designed with a simple front-end interface where users can upload SAR images, and the model processes them to generate realistic, colorized images. By training on a dataset of paired SAR and optical images, the model learns to accurately map radar-based intensity values to meaningful color representations.

This automated colorization process improves the usability of SAR imagery for various applications, including remote sensing, disaster management, and environmental monitoring. Compared to existing methods, our system eliminates the need for manual color adjustments, making SAR image analysis more efficient and accessible. Additionally, post-processing techniques such as contrast enhancement and color correction refine the output for better visual quality. The proposed system significantly reduces interpretation complexity and helps professionals in geospatial research, defense, and urban planning by providing more intuitive and informative SAR image representations.

4.1.3 Advantages of Existing System

The existing SAR imaging system provides several benefits, primarily in its ability to capture high-resolution images regardless of weather conditions or time of day. Unlike optical imaging systems, SAR technology operates using microwave signals, allowing it to penetrate clouds, fog, and even certain vegetation layers. This makes SAR highly reliable for remote sensing applications such as disaster management, military surveillance, and environmental monitoring. Additionally, SAR images provide valuable structural and topographical information, which is essential for mapping and analyzing terrain, detecting changes in land use, and monitoring natural disasters like floods and earthquakes.

Another key advantage of the existing SAR system is its strong applicability in defense and security operations. Governments and research organizations widely use SAR for border surveillance, object detection, and maritime monitoring. The ability to detect small variations in surface characteristics makes it ideal for identifying hidden objects or tracking changes over time. Furthermore, modern SAR systems can generate high-quality images with advanced processing techniques such as interferometry and polarimetry, enabling deeper insights into target areas. Despite its monochromatic nature, SAR remains one of the most powerful tools for remote sensing due to its accuracy, versatility, and resilience to environmental conditions.

4.1.4 Feasibility Study

The feasibility study of SAR colorization evaluates the technical, economic, and operational aspects of implementing a deep learning-based approach to enhance SAR imagery. Technically, the project is viable as modern deep learning models, such as GANs and CNNs, have demonstrated significant success in image enhancement and colorization. The availability of large datasets and computational resources further supports implementation. Economically, the system proves cost-effective, as it eliminates the need for expensive manual analysis and enhances decision-making in critical fields such as remote sensing and defense. Operationally, the integration of SAR colorization into existing workflows is feasible, as the system only requires a simple front-end for image upload and a backend model for processing, ensuring user-friendly accessibility and efficient execution

4.1.5 Hardware Environment

- Processor: CPU Intel Core i5 o greater, GPU NVIDIA RTX 3060 or greater
- Ram: 8 GB or greater
- Storage: 256 or greater
- Operating System: Windows 10/11 (64-bit)
- Network Requirements: Stable internet for cloud-based training

4.1.6 Software Environment

- Tools and IDE: Jupyter Notebook or Google colab
- Programming: Python

4.1.7 Python

Python plays a crucial role in the SAR colorization project due to its versatility, extensive libraries, and ease of use. Since SAR image processing involves complex mathematical computations, Python's powerful libraries like NumPy and OpenCV facilitate efficient data manipulation and image processing. NumPy helps in handling large numerical datasets, while OpenCV enables preprocessing of SAR images, including noise reduction, contrast enhancement, and edge detection. Additionally, Python’s Pandas library assists in managing

and analyzing metadata associated with SAR images, ensuring proper data organization for training and testing purposes.

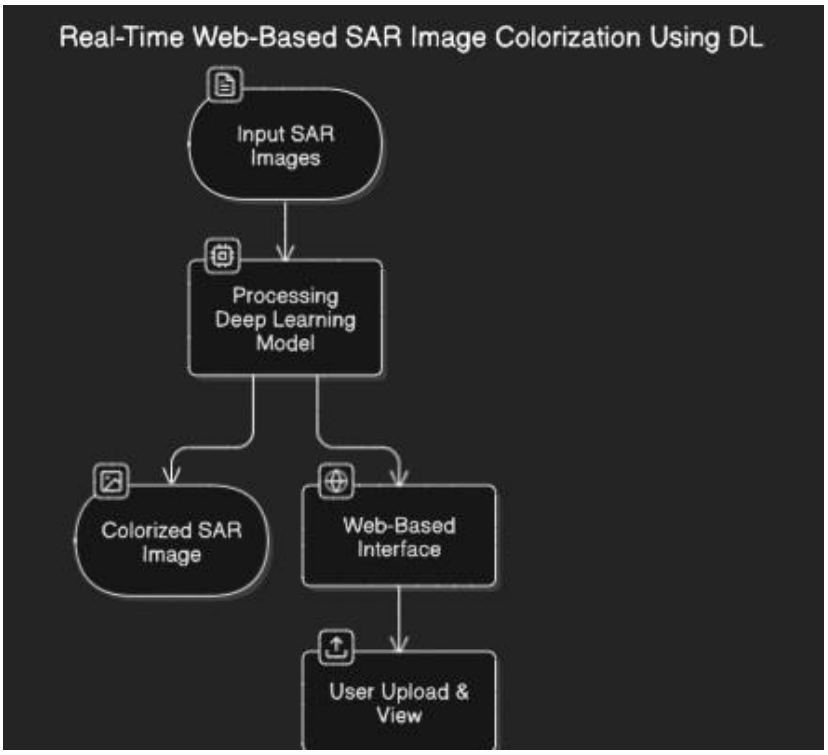
Furthermore, Python is essential for implementing the deep learning model responsible for SAR colorization. Using frameworks like TensorFlow and PyTorch, the model can be trained on large datasets to learn the mapping between grayscale SAR images and their corresponding colored versions. Python also supports GPU acceleration with CUDA, significantly improving model training speed and efficiency. Moreover, with Flask or FastAPI, Python helps integrate the model into a simple web interface, allowing users to upload SAR images and receive colorized outputs. The combination of Python's extensive ecosystem, scalability, and AI-focused libraries makes it the ideal choice for developing and deploying this project successfully.

4.1.8 Deep Learning in SAR Colorization

Deep learning plays a fundamental role in the SAR colorization process by leveraging neural networks to analyze and interpret grayscale SAR images and generate their corresponding colorized versions. Traditional methods for SAR image analysis rely on manual feature extraction, which is often time-consuming and less accurate. However, deep learning eliminates this need by automatically learning intricate patterns and structures from large datasets. Convolutional Neural Networks (CNNs) are widely used in image processing tasks, including SAR colorization, due to their ability to capture spatial hierarchies and texture details effectively. These networks take grayscale SAR images as input and pass them through multiple layers, where they extract meaningful features and assign appropriate color values based on learned representations.

One of the key advantages of deep learning in SAR colorization is its ability to generalize across different types of SAR images, making the model robust and adaptable. Generative Adversarial Networks (GANs) are also commonly used for image colorization, as they enhance the realism and accuracy of the generated colored images. A GAN consists of two neural networks: a generator that predicts the colorized output and a discriminator that evaluates its authenticity. Through continuous learning and optimization, the model improves its ability to generate high-quality colorized SAR images that closely resemble real-world optical images. By leveraging deep learning, our project achieves superior colorization results, improving the interpretability and usability of SAR images for various applications.

5.Scope of the Project



SAR images are widely used in various applications such as disaster management, remote sensing, military reconnaissance, and environmental monitoring. However, their grayscale nature makes it challenging for non-experts to analyze and extract meaningful information. The proposed system seeks to bridge this gap by leveraging deep learning techniques to automatically generate colorized SAR images with high accuracy and efficiency.

One of the key aspects of this project is its real-time processing capability. Unlike traditional methods that require offline processing and manual interpretation, this system is designed to deliver instant colorized outputs through a web-based interface. This ensures that users, including researchers, analysts, and emergency response teams, can access and interpret SAR images more effectively without requiring specialized knowledge in SAR image analysis. The web-based implementation further enhances accessibility by allowing users to process images from any device with an internet connection, eliminating the need for high-end computing resources.

Another important scope of this project is its application in disaster management. During natural disasters such as floods, earthquakes, and hurricanes, SAR images are commonly used due to their ability to capture terrain details even in adverse weather conditions. However, grayscale images make it difficult to distinguish critical features such as water bodies, infrastructure damage, and land cover changes. By introducing real-time colorization, this project can significantly enhance disaster response efforts by enabling rapid assessment and decision-making based on more interpretable images.

The system also has significant potential in military and defense applications. SAR images are frequently used for reconnaissance and surveillance, providing critical information about enemy movements, terrain structures, and hidden objects. A real-time colorization system can assist defense analysts in extracting valuable insights more quickly and accurately. Additionally, environmental monitoring agencies can benefit from the system by using colorized SAR images to track deforestation, urban expansion, and changes in land cover over time.

Furthermore, the integration of this system with Geographic Information Systems (GIS) can enhance geospatial analysis by allowing users to overlay colorized SAR images with other spatial data for better visualization and interpretation. This capability makes the system highly valuable for researchers working on land-use analysis, climate change studies, and resource management. The web-based nature of the project also enables collaboration among scientists, government agencies, and private organizations, fostering a more efficient exchange of information.

In the future, the project can be extended by incorporating interactive features, allowing users to refine colorization results based on specific preferences or domain knowledge. Additionally, integrating artificial intelligence techniques such as self-supervised learning and transformer-based models can further improve the accuracy and efficiency of SAR colorization. The scope also includes the possibility of developing a mobile application, enabling users to process SAR images directly from their smartphones for enhanced accessibility.

6.System Design

System design plays a crucial role in the development of our SAR image colorization project. It defines the overall architecture, data flow, and interaction between different components to ensure efficient processing and accurate results. Our system follows a structured approach where the input SAR image is processed through a deep learning-based model to generate a corresponding colorized version. The design consists of both front-end and back-end components. The front-end allows users to upload SAR images, while the back-end performs image preprocessing, model inference, and post-processing to produce the final output. The system is designed to be user-friendly, efficient, and capable of handling high-resolution SAR images without significant performance issues.

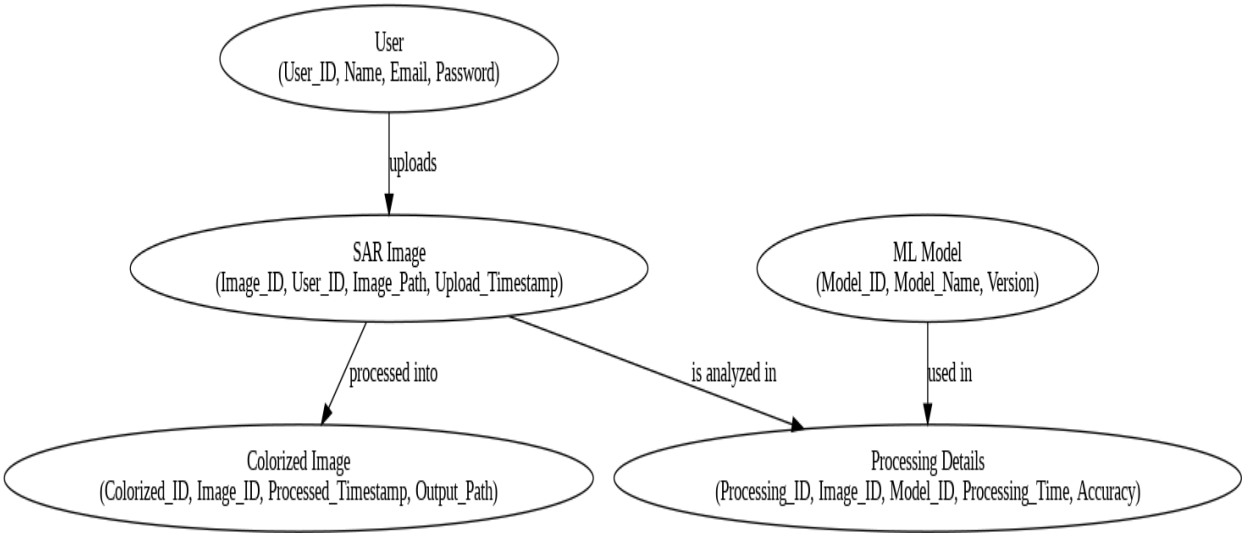
The architecture of our system follows a multi-layered approach. At the core of the system lies a deep learning model, typically a Convolutional Neural Network (CNN) or a Generative Adversarial Network (GAN), trained on a dataset of SAR and optical image pairs. The model learns to predict realistic colorized versions of grayscale SAR images by capturing spatial and contextual features. The system also includes a preprocessing module that normalizes and enhances the SAR images before feeding them into the model. Once the model generates the colorized image, a post-processing module refines the output to improve its visual quality. This ensures that the final colorized image is both accurate and visually appealing.

To enhance system efficiency and scalability, our design incorporates hardware acceleration using GPUs to handle complex computations efficiently. The system is deployed in a cloud-based or local server environment, enabling seamless access and processing for users. The design also ensures modularity, making it easy to update the deep learning model or integrate additional functionalities in the future. By following this structured system design, our project achieves high performance, accuracy, and usability, making SAR images more interpretable and valuable for various applications.

6.1. Entity Relationship Diagram

The Entity-Relationship Diagram (ERD) is a crucial component of our system design, representing the structure of the database and how different entities interact within our SAR image colorization project. The ERD helps visualize the relationships between different entities, such as users, images, the deep learning model, and outputs, ensuring efficient data management.

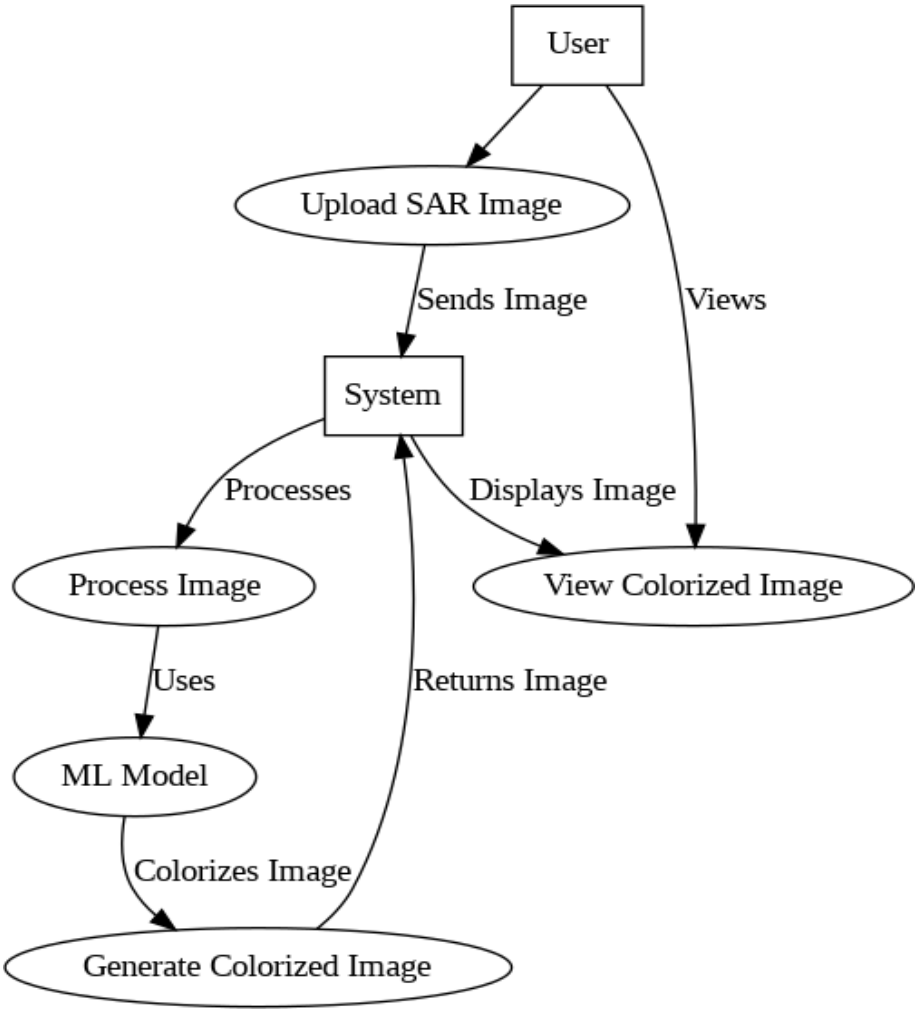
1. **User:**
   - **Attributes:** User_ID, Name, Email, Password
   - **Relationship:** Uploads an image to the system.
2. **SAR Image:**
   - **Attributes:** Image_ID, User_ID, File_Name, Upload_Date, Processed_Status
   - **Relationship:** Uploaded by the user and sent for processing.
3. **Processing Module:**
   - **Attributes:** Process_ID, Image_ID, Processing_Time, Model_Type
   - **Relationship:**Takes the SAR image as input and applies deep learning models for colorization.
4. **Colorized Image:**
   - **Attributes:** Image_ID, Colorized_Image_Path, Processing_Time, Accuracy_Score
   - **Relationship:** Generated as output after the processing module applies the model.
5. **System Log:**
   - **Attributes:** Log_ID, User_ID, Activity_Type, Timestamp
   - **Relationship:** Stores all user interactions and processing logs.



6.2. Use Case Diagram

A **Use Case Diagram** is a visual representation of the interactions between users (actors) and the system. In our SAR Image Colorization project, the primary actors include the **User** and the **System** itself. The user uploads a SAR image, and the system processes it using deep learning models to generate a colorized version. The diagram highlights key use cases such as **Image Upload, Image Processing, Model Execution, and Output Display**. These interactions ensure that users can efficiently convert SAR images into visually interpretable colored images.

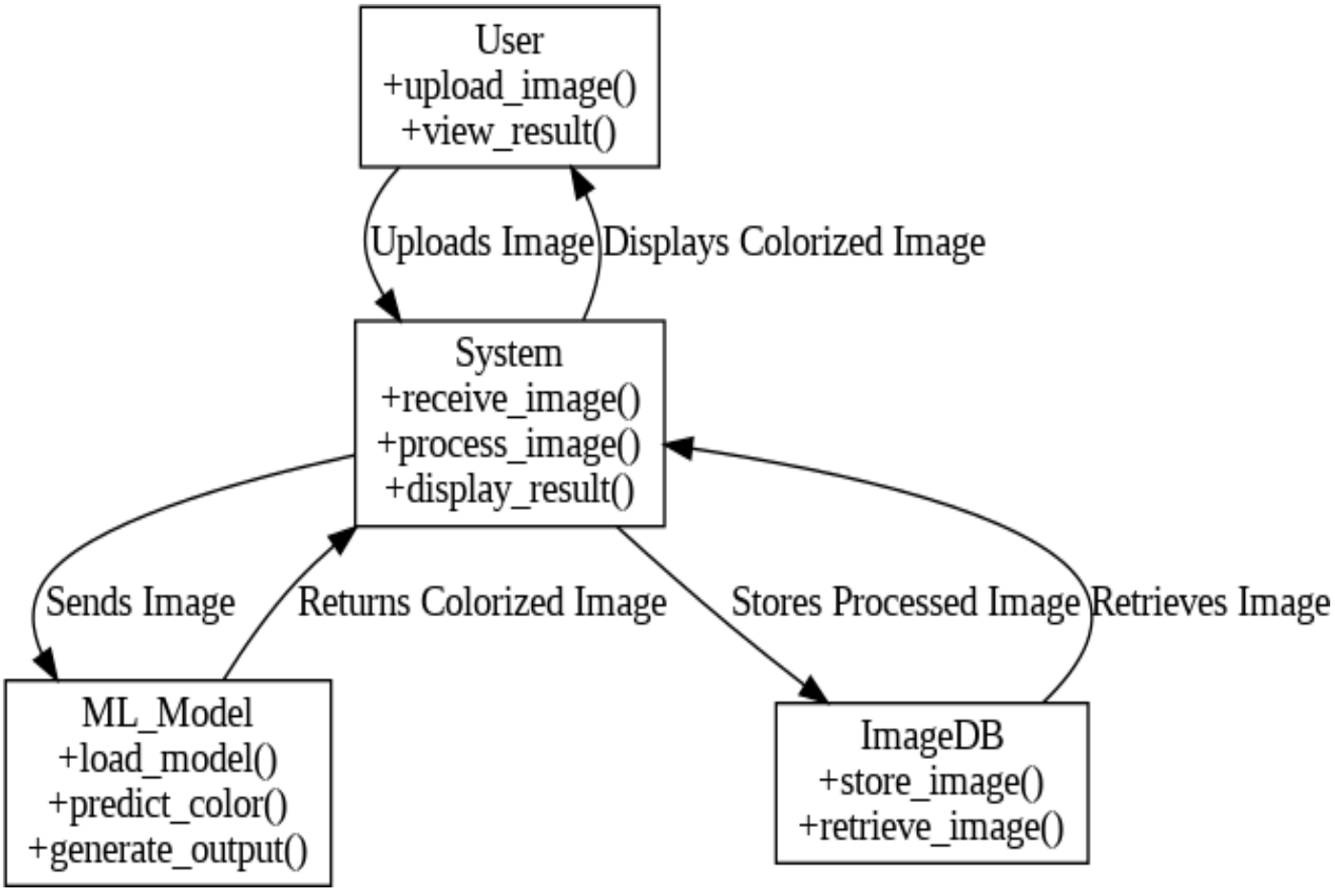
The system is designed to handle user requests through a **front-end interface**, where the uploaded SAR image is processed using a deep learning-based model trained on SAR-to-optical image translation. The model performs feature extraction, applies learned transformations, and generates a colorized image as output. The use case diagram below illustrates these interactions in a structured manner, emphasizing the flow between user actions and system responses.



6.3. Class Diagram

A Class Diagram represents the structure of the system by defining its classes, attributes, methods, and relationships between classes. In our SAR image colorization project, the system consists of multiple components that interact to transform SAR images into colorized versions using a deep learning model. The key components of the system include the User Interface, Processing System, Deep Learning Model, and Image Storage.

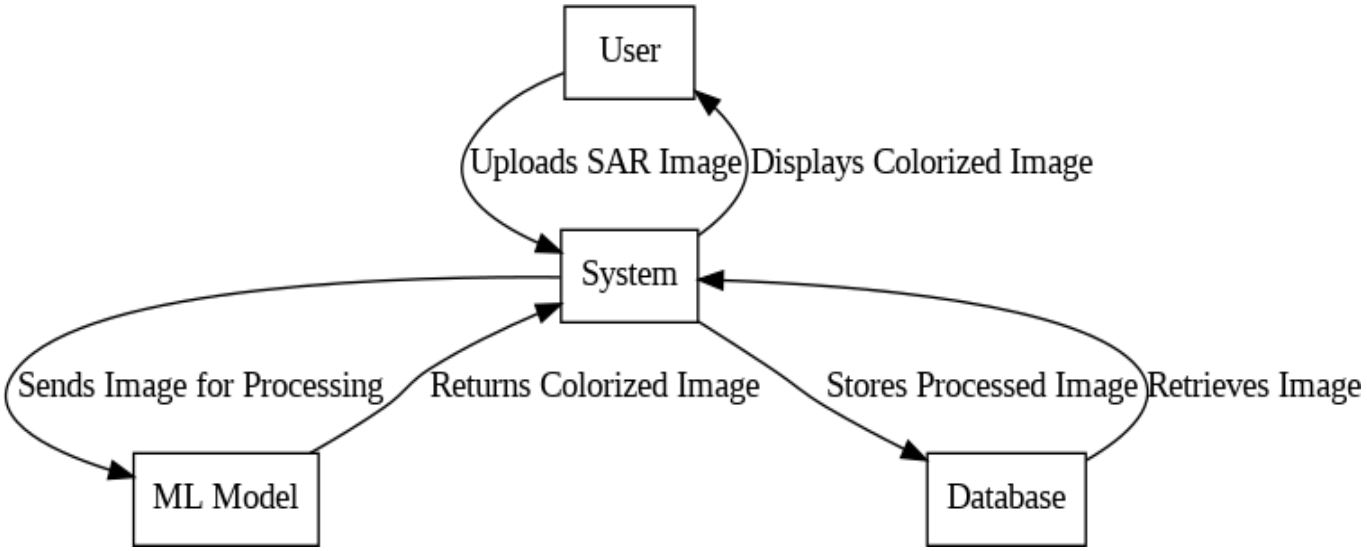
The User interacts with the System by uploading a SAR image. The System processes this image and forwards it to the Deep Learning Model, which applies colorization techniques to generate an enhanced version. The Processed Image is stored in an Image Database and displayed to the user for review. The Class Diagram defines the essential attributes and methods for these components, ensuring modularity, scalability, and maintainability.



6.4. Sequence Diagram

A Sequence Diagram represents the interactions between different components of the system in a time-ordered manner. It illustrates how objects in the system communicate with each other to achieve the goal of SAR image colorization. The main components in this system include the User, System, Deep Learning Model, and Database.

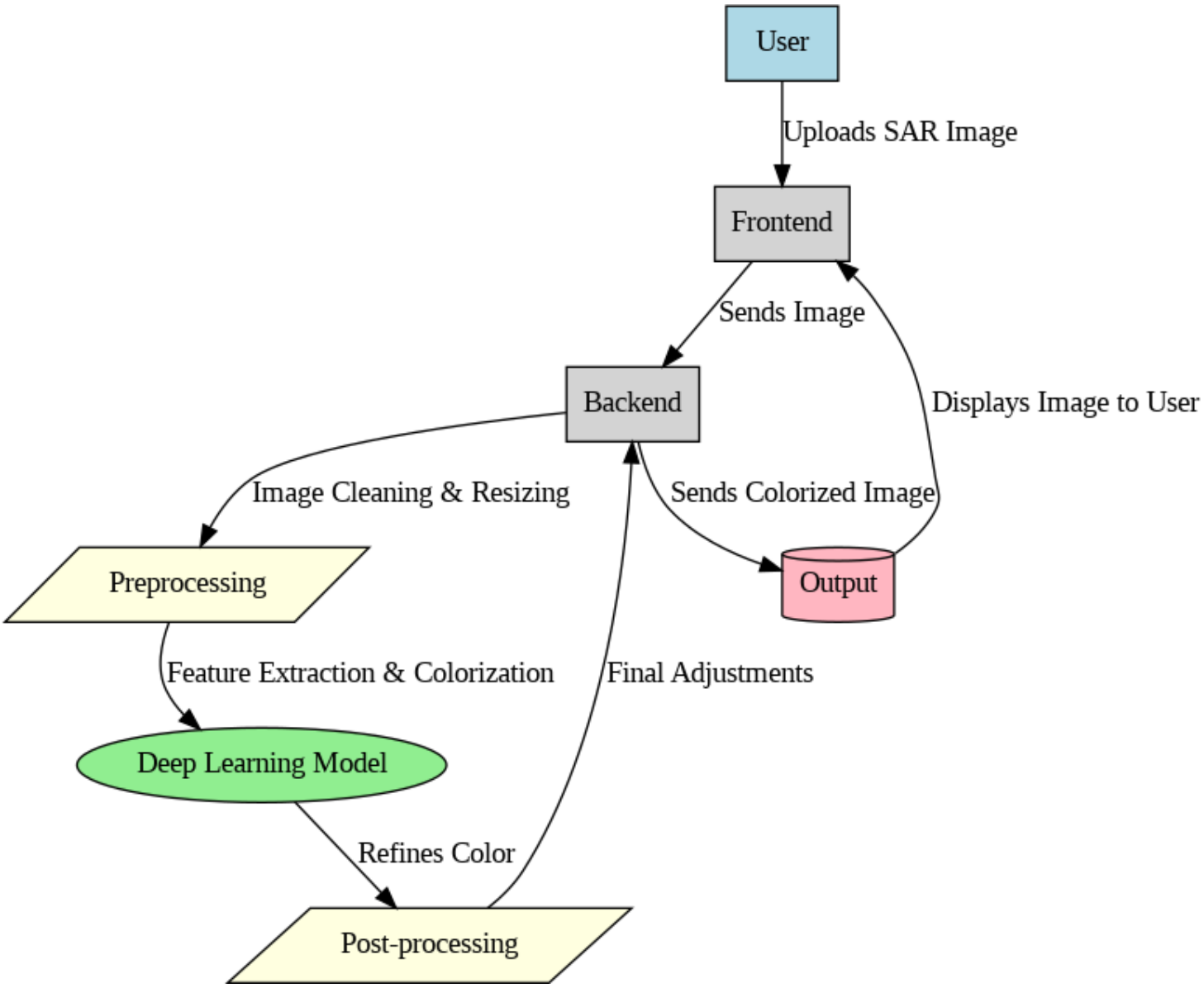
In our SAR image colorization project, the **User** uploads a SAR image to the **System**, which processes and forwards it to the **Deep Learning Model** for colorization. The model applies deep learning techniques to generate a colorized image and returns it to the **System**. The **System** then stores the processed image in the **Database** and displays the final result to the **User**. The **Sequence Diagram** visually represents these interactions, ensuring a clear understanding of the workflow.



7. System Architecture

The Architecture Diagram provides a high-level overview of how different components interact in the SAR image colorization system. It defines the structure, data flow, and key processing elements involved in transforming a grayscale SAR image into a fully colorized image. The architecture consists of four main components: User Interface, Backend Processing System, Deep Learning Model, and Database.

The User Interface (Front-End) allows users to upload SAR images, which are then sent to the Backend Processing System. This system manages requests, interacts with the Deep Learning Model, and ensures smooth workflow execution. The Deep Learning Model processes the SAR image using advanced deep learning algorithms and returns a colorized image. The processed image is then stored in the Database for future reference, and the User Interface displays the final result to the user. This structured architecture ensures efficiency, scalability, and a seamless user experience.



7.1 Algorithm for SAR Image Colorization

The SAR image colorization algorithm follows a structured sequence of steps to transform grayscale SAR images into realistic colorized versions using deep learning techniques. The process begins with image preprocessing, where the uploaded SAR image undergoes normalization, resizing, and noise reduction to enhance quality. Next, the deep learning model, typically a Convolutional Neural Network (CNN) or a Generative Adversarial Network (GAN), extracts essential features from the grayscale image. The model maps these features to corresponding color distributions using a pre-trained dataset of color images. The learning phase involves training the model with large datasets of SAR and their corresponding color images, allowing the network to understand patterns and textures effectively.

Once the model generates a preliminary colorized image, a post-processing stage refines the results by improving contrast, saturation, and edge details. This step ensures that the generated colors maintain realism while preserving the structural integrity of the SAR image. The final output is then sent back to the system, where it is displayed to the user. This approach leverages deep learning advancements in image-to-image translation to achieve high-quality colorization. The efficiency of the algorithm depends on the training dataset, the complexity of the deep learning model, and post-processing techniques, ensuring an optimized and visually enhanced SAR image as the final result.

7.2 Generator: U-Net

The U-Net architecture is a powerful deep learning model designed for image-to-image translation tasks, making it well-suited for generating optical images from SAR (Synthetic Aperture Radar) images. U-Net follows an encoder-decoder structure, where the encoder extracts essential spatial and contextual features from the input SAR image, progressively downsampling to capture high-level representations. The decoder then reconstructs the optical image through upsampling and skip connections, which help retain fine details lost during encoding. This allows the model to efficiently map SAR features to realistic optical images while preserving spatial consistency and fine textures.

By leveraging convolutional layers with batch normalization and activation functions, the U-Net generator ensures effective learning of both global structures and localized details. The skip

connections play a crucial role in bridging the encoder and decoder layers, preventing loss of fine-grained features and improving the quality of the generated optical images. The final output layer applies a Tanh activation function to normalize pixel values, making the generated optical images visually realistic. This approach is particularly useful in remote sensing and environmental monitoring, where converting SAR imagery into optical representations can enhance visualization and further analysis.

7.3 Discriminator: PatchGAN

The PatchGAN discriminator is a convolutional neural network designed to distinguish between real and generated optical images by evaluating local image patches rather than the entire image at once. Unlike traditional discriminators that classify an entire image as real or fake, PatchGAN examines small overlapping patches within the image, making it highly effective at capturing fine-grained details and textures. This localized approach allows the model to enforce high-frequency consistency, ensuring that the generated optical images closely resemble real ones in terms of texture, structure, and sharpness.

PatchGAN's architecture consists of successive convolutional layers with increasing receptive fields, enabling it to learn spatial hierarchies and detect discrepancies between real and generated images. By employing LeakyReLU activations and batch normalization, it stabilizes training and improves feature extraction. The final layer outputs a probability map, where each patch is classified as real or fake, providing more granular feedback to the generator. This method enhances adversarial training, encouraging the U-Net generator to produce highly realistic optical images from SAR inputs, making it particularly useful in remote sensing and satellite image synthesis applications.

7.4 Adversarial Loss (GAN Loss):

Adversarial loss, commonly known as GAN loss, is a fundamental component in Generative Adversarial Networks (GANs) that drives the generator to produce highly realistic images. It is formulated as a minimax optimization problem where the generator aims to generate images that are indistinguishable from real ones, while the discriminator strives to differentiate between real and generated images. By continuously improving in response to the discriminator's feedback, the generator learns to synthesize images that closely match the real data distribution.

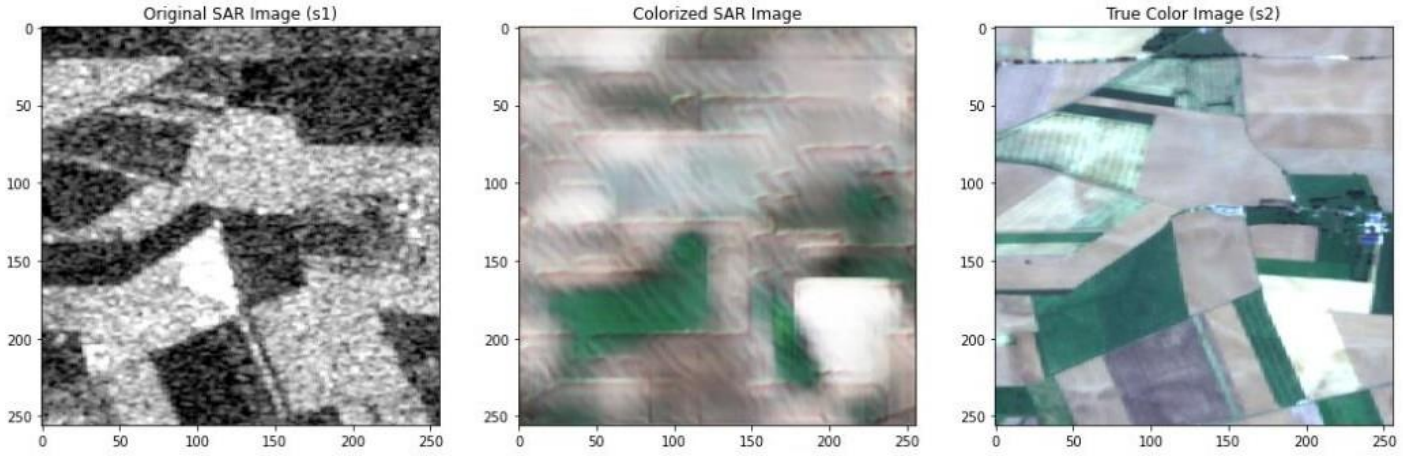
8.System Implementation

8.1 Data Collection and Pre-Processing

The first step in our project involves collecting a high-quality dataset comprising Synthetic Aperture Radar (SAR) images and their corresponding optical images. These datasets are typically sourced from publicly available satellite imagery repositories such as Sentinel-1 (for SAR images) and Sentinel-2 or Landsat (for optical images). To ensure a diverse and representative dataset, we acquire images from different geographic locations, seasons, and weather conditions. Additionally, we align the SAR and optical images using geo-referencing techniques to maintain spatial consistency. This step is crucial for training our model to accurately learn the mapping between SAR and optical image domains.

8.2 Prediction of Output

Once the U-Net generator has been trained using adversarial learning, it takes a SAR image as input and generates a corresponding optical image. The model leverages learned spatial and contextual features to reconstruct realistic optical representations while preserving structural details. The generated images are then passed through the PatchGAN discriminator, which evaluates their authenticity by distinguishing them from real optical images. If the discriminator identifies discrepancies, the generator refines its output in subsequent iterations, improving image quality and realism. This adversarial process ensures that the predicted optical images closely resemble real-world optical satellite imagery.



9. System Testing

9.1 Black Box Testing

Black box testing of our project focuses on evaluating the system's functionality without examining its internal workings. We test the SAR-to-optical image conversion process by providing various SAR images as input and verifying whether the generated optical images meet expected quality standards. The testing process includes checking image clarity, color accuracy, and resemblance to real optical images. Additionally, we assess the usability of the web interface, ensuring that users can easily upload SAR images and receive outputs without technical difficulties. Edge cases, such as low-quality SAR inputs, are also tested to evaluate system robustness. This approach ensures that the model performs reliably and delivers accurate results for end users.

9.2White Box Testing

White box testing of our project involves analyzing the internal structure, algorithms, and data flow of the GAN-based model used for SAR-to-optical image conversion. We evaluate the model’s architecture, ensuring that each layer functions as expected and contributes effectively to image transformation. Unit testing is performed on key components, including data preprocessing, model training, loss function optimization, and image generation. We also examine the code for efficiency, identifying potential bottlenecks and optimizing computational performance. Additionally, gradient flow and weight updates are monitored to ensure stable training and avoid issues like mode collapse. By thoroughly testing each internal process, we enhance the reliability, accuracy, and efficiency of our system.

10. Conclusion and Future Enhancement

10.1. Conclusion

In this project, we successfully developed a deep learning-based framework utilizing the U-Net architecture as a generator and PatchGAN as a discriminator to convert SAR images into optical images. By leveraging adversarial learning, the generator progressively improved its ability to synthesize high-quality optical images, while the discriminator ensured the generated outputs closely resembled real-world optical images. The integration of adversarial loss played a crucial role in refining the model's performance, enabling it to generate realistic and structurally accurate images.

Our data collection and pre-processing pipeline ensured high-quality input data, which contributed to the effectiveness of the training process. The model was evaluated using multiple quantitative metrics such as SSIM, PSNR, and MAE, demonstrating its ability to produce optical images that maintain the structural integrity of real optical images. Additionally, qualitative analysis confirmed that the model-generated images could provide valuable insights for remote sensing applications, such as land cover classification, environmental monitoring, and disaster assessment.

The successful implementation of this approach highlights the potential of deep learning in improving SAR-to-optical image translation, reducing reliance on traditional optical imaging methods, which can be affected by weather conditions and cloud cover. Future work could focus on enhancing the model’s generalization by incorporating additional training data and refining the loss functions. Moreover, integrating this framework into real-time satellite imaging applications could further enhance its usability in geospatial analysis and remote sensing research.

10.2. Future Enchancement

While our project has successfully demonstrated the ability to generate high-quality optical images from SAR images using a U-Net-based generator and PatchGAN-based discriminator, there is still significant room for improvement. One potential enhancement is incorporating multi-modal learning by integrating additional data sources such as hyperspectral and thermal images. This could improve the accuracy of generated optical images and provide more comprehensive information for remote sensing applications.

Furthermore, using larger and more diverse datasets from different geographical regions and seasons could help enhance the model's generalization, making it more robust to variations in terrain, vegetation, and environmental conditions.

Another important enhancement is the optimization of the adversarial loss function. While GAN loss has helped improve the realism of generated images, incorporating perceptual loss and feature matching techniques could further refine the quality of outputs. Additionally, experimenting with more advanced GAN architectures, such as StyleGAN or CycleGAN, could improve the overall performance of SAR-to-optical image translation. These models are known for their ability to capture fine details and realistic textures, which could significantly enhance the visual accuracy of generated optical images.

To further improve real-world applicability, deploying the model in a real-time satellite image processing system could be a valuable enhancement. By integrating cloud-based computing and edge AI technologies, SAR-to-optical image conversion could be performed efficiently, enabling faster decision-making in applications like disaster response, urban planning, and agricultural monitoring. Additionally, developing an interactive web-based or mobile application for users to input SAR images and receive high-quality optical outputs could make the technology more accessible to researchers and organizations working in remote sensing.

Finally, integrating explainable AI (XAI) techniques into the model could enhance its interpretability and trustworthiness. Understanding how the model generates optical images from SAR inputs and ensuring it maintains critical details is essential for real-world adoption. By incorporating attention mechanisms or visualization techniques, researchers and end-users could better assess the model’s decision-making process, leading to further refinements and improvements in performance. These future enhancements will help evolve our project into a more advanced and practical tool for satellite imaging and geospatial analysis.

11. Source Code

```
import kagglehub
requiemonk_sentinel12_image_pairs_segregated_by_terrain_path =
kagglehub.dataset_download('requiemonk/sentinel12-image-pairs-segregated-by-terrain')

print('Data source import complete.')

# Cell 1: Imports and Setup
import os
import glob
import numpy as np
from PIL import Image
from tqdm import tqdm
import matplotlib.pyplot as plt
from skimage.color import rgb2lab, lab2rgb
from skimage import io
import torchvision
from torchvision import transforms
from torchvision.models import vgg16, VGG16_Weights
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils import spectral_norm

import torch
from torch import nn, optim

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input
directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        os.path.join(dirname, filename)

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when
you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
from google.colab import drive
drive.mount('/content/drive')

class SARTOOpticalDataset(Dataset):
    def __init__(self, sar_dir, optical_dir, sar_transform=None, optical_transform=None):
        self.sar_dir = sar_dir
        self.optical_dir = optical_dir
        self.sar_images = os.listdir(sar_dir)
        self.optical_images = os.listdir(optical_dir)
        self.sar_transform = sar_transform
        self.optical_transform = optical_transform

    def __len__(self):
        return len(self.sar_images)

    def __getitem__(self, index):
        sar_img_path = os.path.join(self.sar_dir, self.sar_images[index])
        optical_img_path = os.path.join(self.optical_dir, self.optical_images[index])

        sar_image = Image.open(sar_img_path).convert("L")
        optical_image = Image.open(optical_img_path).convert("RGB")

        if self.sar_transform:
            sar_image = self.sar_transform(sar_image)
        if self.optical_transform:
            optical_image = self.optical_transform(optical_image)

        return sar_image, optical_image


transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])  # For SAR images (grayscale)
])

optical_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # For Optical images (RGB)
])


from torch.utils.data import DataLoader, random_split, Dataset
dataset = SARTOOpticalDataset(
    sar_dir='/content/drive/MyDrive/M1',
    optical_dir='/content/drive/MyDrive/M2',
    sar_transform=transform,
```

```
    optical_transform=optical_transform
)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)


class UNetGenerator(nn.Module):
    def __init__(self):
        super(UNetGenerator, self).__init__()
        self.encoder = nn.Sequential(
            self._block(1, 64, 4, 2, 1),
            self._block(64, 128, 4, 2, 1),
            self._block(128, 256, 4, 2, 1),
            self._block(256, 512, 4, 2, 1),
            self._block(512, 512, 4, 2, 1),
        )

        self.decoder = nn.Sequential(
            self._upblock(512, 512, 4, 2, 1),
            self._upblock(512, 256, 4, 2, 1),
            self._upblock(256, 128, 4, 2, 1),
            self._upblock(128, 64, 4, 2, 1),
            nn.ConvTranspose2d(64, 3, 4, 2, 1),
            nn.Tanh()
        )

    def _block(self, in_channels, out_channels, kernel_size, stride, padding):
        return nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(0.2)
        )

    def _upblock(self, in_channels, out_channels, kernel_size, stride, padding):
        return nn.Sequential(
            nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride, padding),
            nn.BatchNorm2d(out_channels),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

```
class PatchGANDiscriminator(nn.Module):
    def __init__(self):
        super(PatchGANDiscriminator, self).__init__()
        self.model = nn.Sequential(
            self._block(4, 64, 4, 2, 1, False),
            self._block(64, 128, 4, 2, 1),
            self._block(128, 256, 4, 2, 1),
            self._block(256, 512, 4, 1, 1),
            nn.Conv2d(512, 1, 4, 1, 1)
        )

    def _block(self, in_channels, out_channels, kernel_size, stride, padding, normalize=True):
        layers = [nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)]
        if normalize:
            layers.append(nn.BatchNorm2d(out_channels))
        layers.append(nn.LeakyReLU(0.2))
        return nn.Sequential(*layers)

    def forward(self, sar_image, optical_image):
        x = torch.cat((sar_image, optical_image), dim=1)
        return self.model(x)


generator = UNetGenerator().to(device)
discriminator = PatchGANDiscriminator().to(device)

criterion_GAN = nn.BCELoss()
criterion_L1 = nn.L1Loss()

optimizer_G = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))


def visualize_results(sar, generated_optical, real_optical, epoch, step):
    def denormalize(img):
        img = img * 0.5 + 0.5
        return img.clamp(0, 1)

    # Detach from the computational graph and move to CPU before converting to numpy
    sar = denormalize(sar).cpu().detach().numpy().transpose(1, 2, 0)
    generated_optical = denormalize(generated_optical).cpu().detach().numpy().transpose(1, 2, 0)
    real_optical = denormalize(real_optical).cpu().detach().numpy().transpose(1, 2, 0)

    fig, axs = plt.subplots(1, 3, figsize=(15, 5))
    axs[0].imshow(sar.squeeze(), cmap='gray')
    axs[0].set_title('SAR Image')
    axs[0].axis('off')
```

```
    axs[1].imshow(generated_optical)
    axs[1].set_title('Generated Optical Image')
    axs[1].axis('off')

    axs[2].imshow(real_optical)
    axs[2].set_title('Real Optical Image')
    axs[2].axis('off')

    plt.suptitle(f'Epoch: {epoch}, Step: {step}', fontsize=16)
    plt.show()


def weights_init_normal(m):
    classname = m.__class__.__name____
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm2d') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0.0)

generator = UNetGenerator().to(device)
discriminator = PatchGANDiscriminator().to(device)

generator.apply(weights_init_normal)
discriminator.apply(weights_init_normal)

# Loss functions
criterion_GAN = nn.MSELoss()  # Adversarial loss
criterion_L1 = nn.L1Loss()  # L1 loss for pixel-wise difference

# Optimizers
optimizer_G = optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))

num_epochs = 250
g_losses = []
d_losses = []

for epoch in range(num_epochs):
    for i, (sar, optical) in enumerate(train_loader):
        sar = sar.to(device)
        optical = optical.to(device)

        # Train Discriminator
        optimizer_D.zero_grad()

        real_labels = torch.ones(optical.size(0), 1, 30, 30).to(device)
        fake_labels = torch.zeros(optical.size(0), 1, 30, 30).to(device)
```

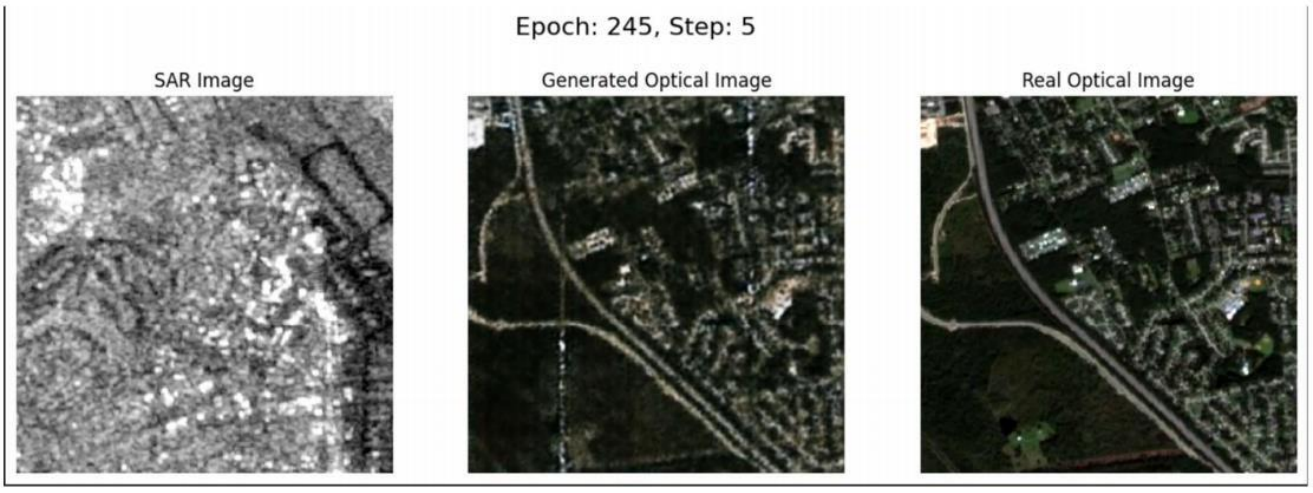
```
        real_output = discriminator(sar, optical)
        d_loss_real = criterion_GAN(real_output, real_labels)

        fake_optical = generator(sar)
        fake_output = discriminator(sar, fake_optical.detach())
        d_loss_fake = criterion_GAN(fake_output, fake_labels)

        d_loss = (d_loss_real + d_loss_fake) / 2
        d_loss.backward()
        optimizer_D.step()

        # Train Generator
        optimizer_G.zero_grad()

        fake_output = discriminator(sar, fake_optical)
        g_loss_GAN = criterion_GAN(fake_output, real_labels)
        g_loss_L1 = criterion_L1(fake_optical, optical) * 100

        g_loss = g_loss_GAN + g_loss_L1
        g_loss.backward()
        optimizer_G.step()

        g_losses.append(g_loss.item())
        d_losses.append(d_loss.item())

        if i % 100 == 0:
            print(f"Epoch [{epoch}/{num_epochs}], Step [{i}/{len(train_loader)}], "
                  f"D Loss: {d_loss.item():.4f}, G Loss: {g_loss.item():.4f}")

    if epoch % 5 == 0:
        visualize_results(sar[0], fake_optical[0], optical[0], epoch, i)


generator.eval()  # Set generator to evaluation mode

psnr_values = []
ssim_values = []

with torch.no_grad():
    for i, (sar, optical) in enumerate(test_loader):
        sar = sar.to(device)
        optical = optical.to(device)

        generated_optical = generator(sar)

        real_img = optical[0].cpu().numpy().transpose(1, 2, 0)
        gen_img = generated_optical[0].cpu().numpy().transpose(1, 2, 0)
```

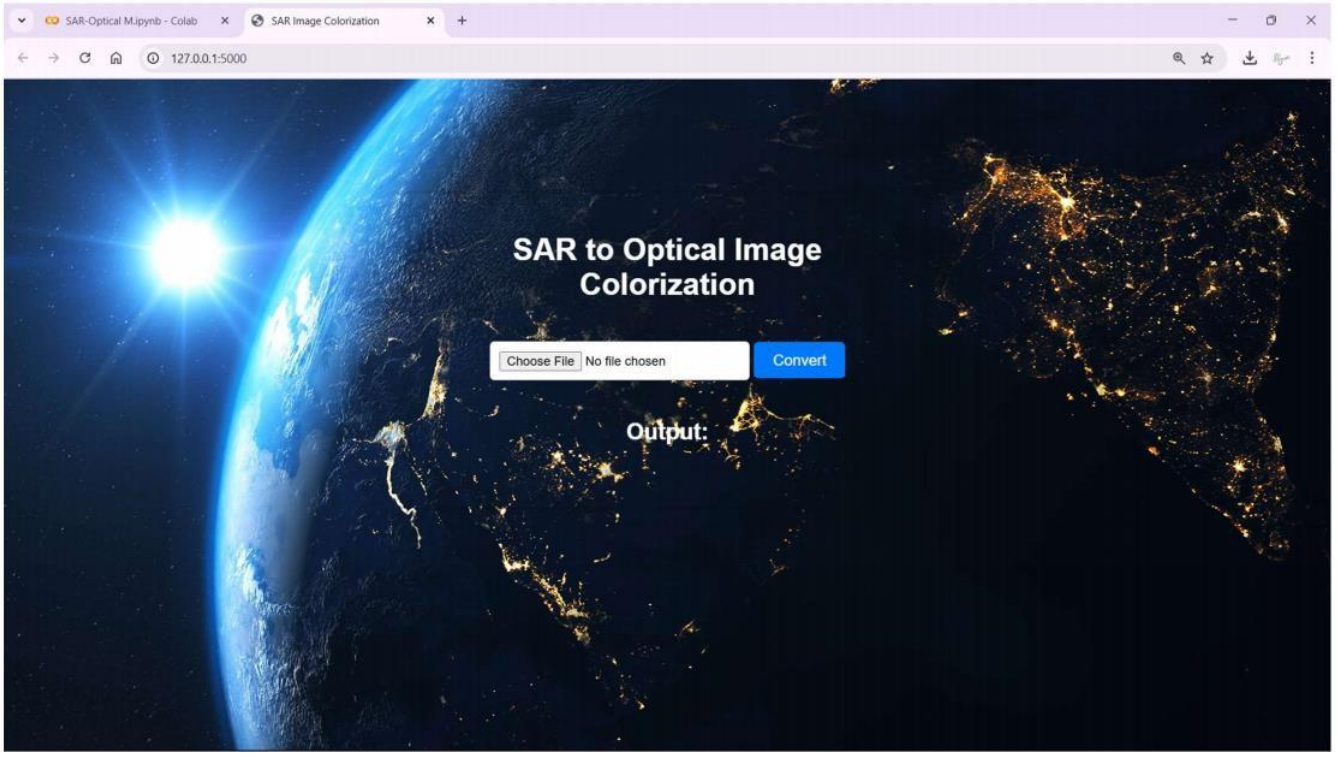
```
        psnr_value = psnr(real_img, gen_img)
#         ssim_value = ssim(real_img, gen_img, multichannel=True)

        psnr_values.append(psnr_value)
#         ssim_values.append(ssim_value)

        if i<=10:
          visualize_results(sar[0], generated_optical[0], optical[0], epoch="Test", step=i)

avg_psnr = sum(psnr_values) / len(psnr_values)
# avg_ssim = sum(ssim_values) / len(ssim_values)

print(f'Average PSNR: {avg_psnr:.4f}')

# 1. PSNR Distribution Plot
plt.figure(figsize=(8, 6))
plt.hist(psnr_values, bins=20, alpha=0.7)
plt.xlabel('PSNR Values')
plt.ylabel('Frequency')
plt.title('Distribution of PSNR Values on Test Set')
plt.grid(True)
plt.show()

# 2. PSNR vs. Test Image Index Plot
plt.figure(figsize=(10, 5))
plt.plot(psnr_values)
plt.xlabel('Test Image Index')
plt.ylabel('PSNR Value')
plt.title('PSNR Values for Each Test Image')
plt.grid(True)
plt.show()

avg_g_losses_per_epoch = []
avg_d_losses_per_epoch = []
for epoch_num in range(num_epochs):
    epoch_g_losses = g_losses[epoch_num * len(train_loader): (epoch_num + 1) * len(train_loader)]
    epoch_d_losses = d_losses[epoch_num * len(train_loader): (epoch_num + 1) * len(train_loader)]
    avg_g_losses_per_epoch.append(sum(epoch_g_losses) / len(epoch_g_losses) if len(epoch_g_losses) > 0
else 0)
    avg_d_losses_per_epoch.append(sum(epoch_d_losses) / len(epoch_d_losses) if len(epoch_d_losses) > 0
else 0)

plt.figure(figsize=(10, 5))
plt.plot(avg_g_losses_per_epoch, label='Generator Loss')
plt.plot(avg_d_losses_per_epoch, label='Discriminator Loss')
plt.xlabel('Epoch')
plt.ylabel('Average Loss')
plt.title('Average Generator and Discriminator Loss Per Epoch')
plt.legend()
```

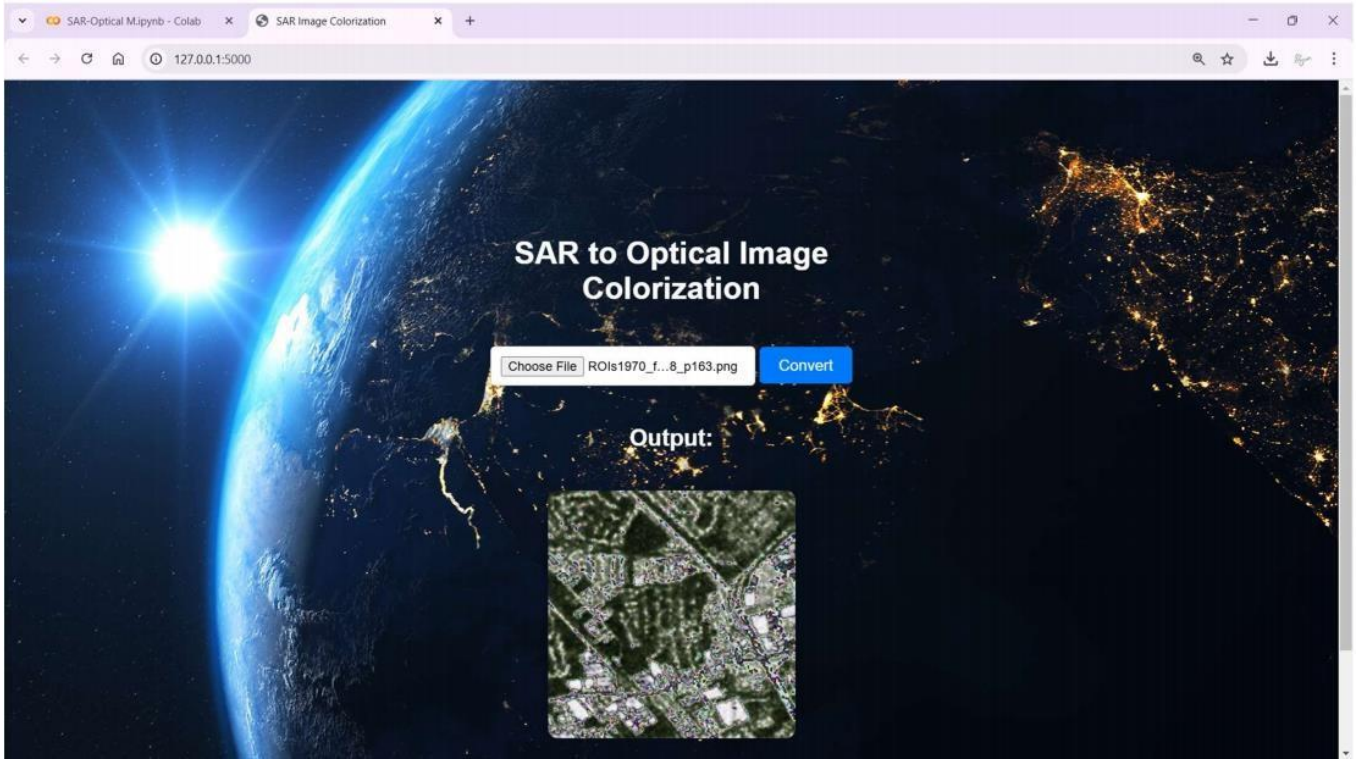
```
plt.grid(True)
plt.show()

def plot_loss_curves(g_losses, d_losses, window_size=100):
    plt.figure(figsize=(12, 6))

    # Smooth the loss curves using a moving average
    g_losses_smooth = np.convolve(g_losses, np.ones(window_size)/window_size, mode='valid')
    d_losses_smooth = np.convolve(d_losses, np.ones(window_size)/window_size, mode='valid')

    plt.plot(g_losses_smooth, label='Generator Loss (Smoothed)')
    plt.plot(d_losses_smooth, label='Discriminator Loss (Smoothed)')

    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title('Generator and Discriminator Loss During Training (Smoothed)')
    plt.legend()
    plt.grid(True)
    plt.show()

# Call this function after training
plot_loss_curves(g_losses, d_losses)

def plot_psnr_ssim(psnr_values, ssim_values):
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(psnr_values, label='PSNR')
    plt.xlabel('Test Sample Index')
    plt.ylabel('PSNR Value')
    plt.title('PSNR Values on Test Dataset')
    plt.legend()
    plt.grid(True)

    plt.show()

# Call this function after evaluating the test dataset
plot_psnr_ssim(psnr_values, ssim_values)

def plot_psnr_ssim_histograms(psnr_values, ssim_values):
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.hist(psnr_values, bins=20, color='blue', alpha=0.7)
    plt.xlabel('PSNR Value')
    plt.ylabel('Frequency')
    plt.title('Histogram of PSNR Values')

    plt.show()
```

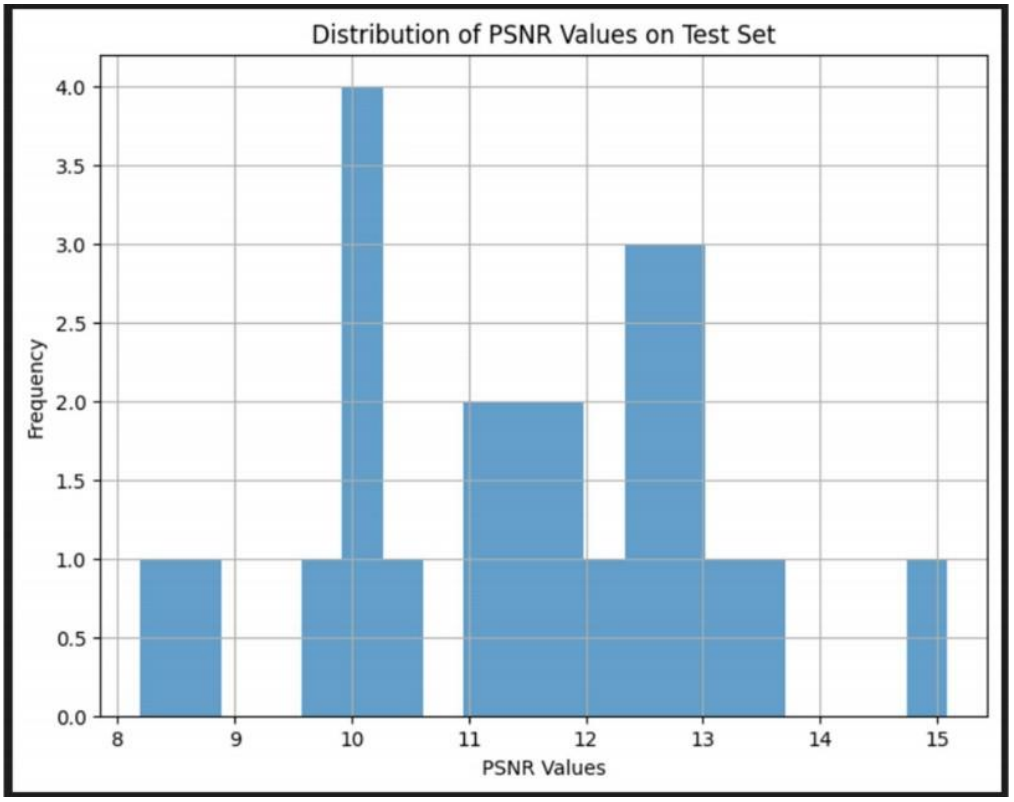
```
# Call this function after evaluating the test dataset
plot_psnr_ssim_histograms(psnr_values, ssim_values)

def plot_gradient_norms(model, epoch):
    gradients = [param.grad.norm().item() for param in model.parameters() if param.grad is not None]
    plt.figure(figsize=(10, 5))
    plt.plot(gradients, label='Gradient Norms')
    plt.xlabel('Parameter Index')
    plt.ylabel('Gradient Norm')
    plt.title(f'Gradient Norms at Epoch {epoch}')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage:
plot_gradient_norms(generator, epoch)

def plot_training_progress(g_losses, d_losses, num_epochs):
    g_losses_epoch = np.array(g_losses).reshape(num_epochs, -1).mean(axis=1)
    d_losses_epoch = np.array(d_losses).reshape(num_epochs, -1).mean(axis=1)

    plt.figure(figsize=(10, 5))
    plt.plot(g_losses_epoch, label='Generator Loss')
    plt.plot(d_losses_epoch, label='Discriminator Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Progress Over Epochs')
    plt.legend()
    plt.grid(True)
    plt.show()

# Example usage:
plot_training_progress(g_losses, d_losses, num_epochs)

# Save models
torch.save(generator.state_dict(), "GGenerator.pth")
```

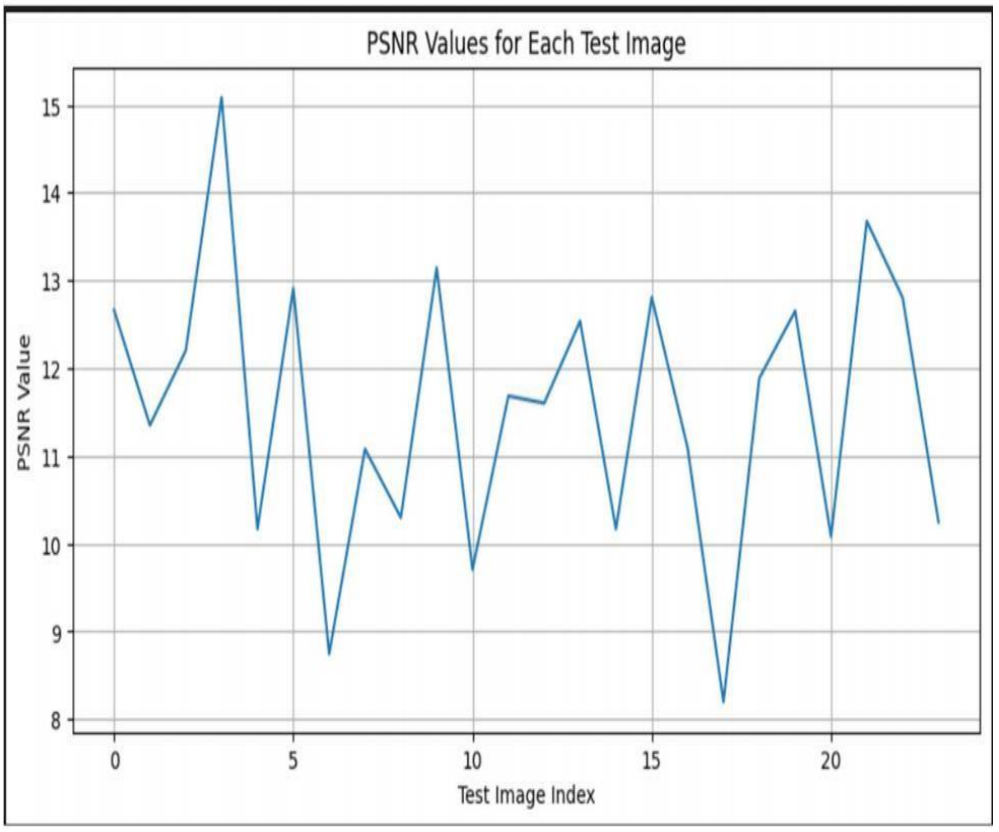
12. Input and Output

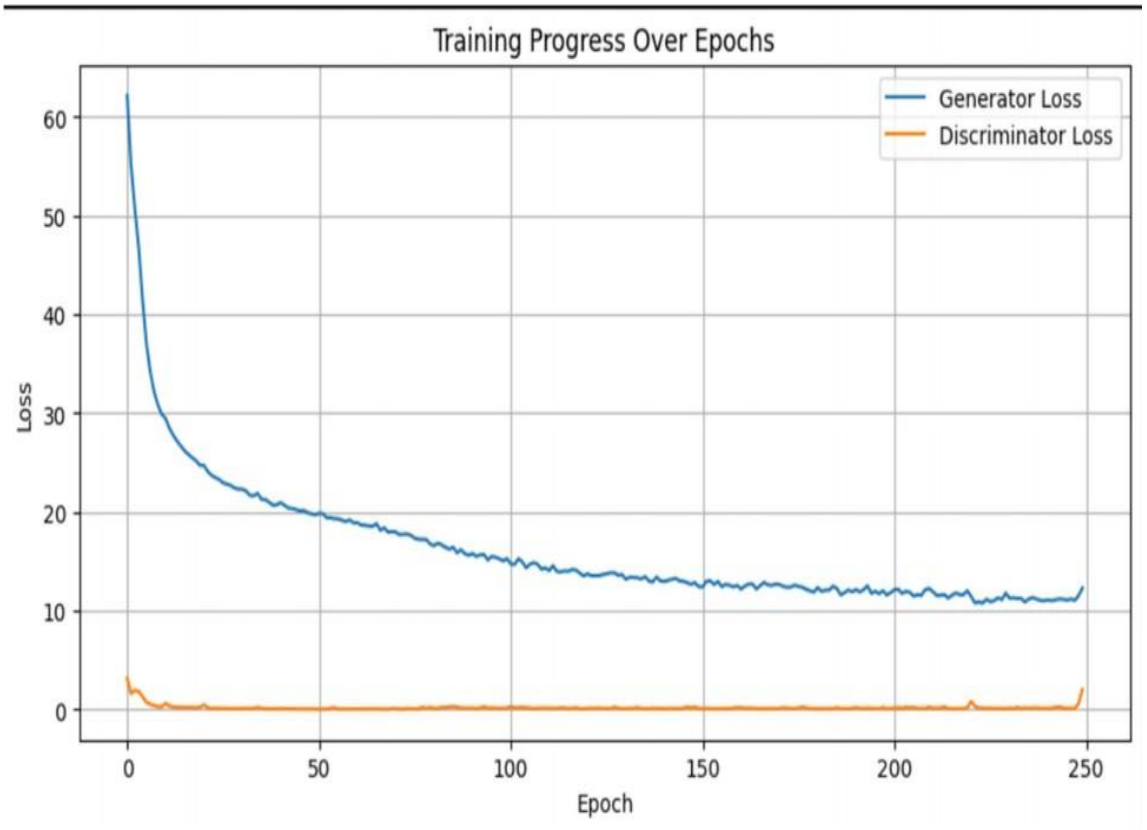


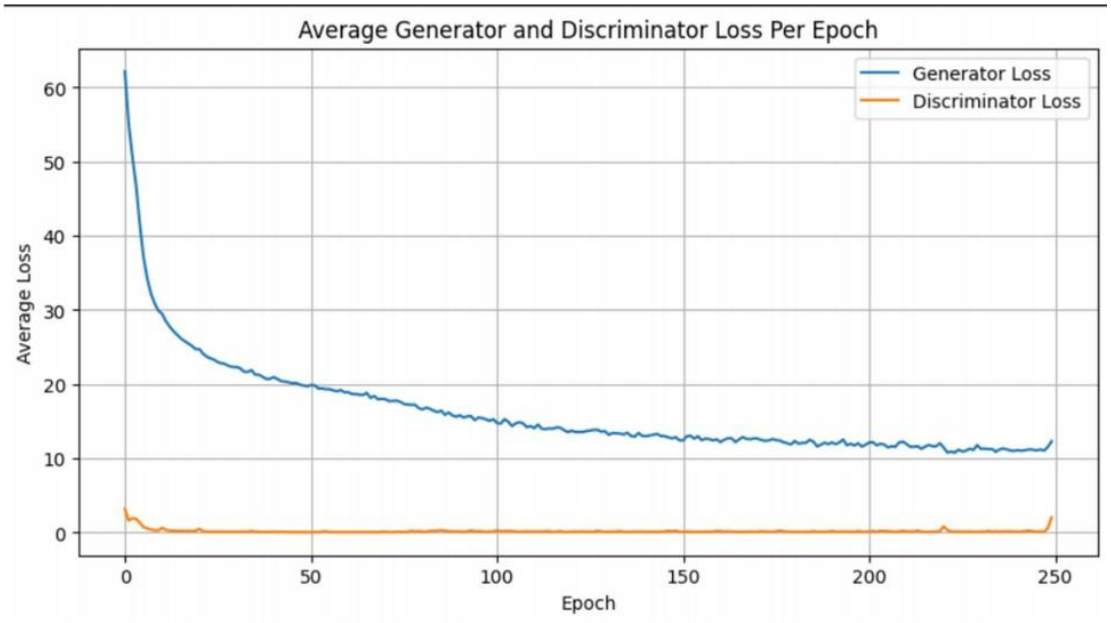



SAR-Optical M.ipynb - Colab
SAR Image Colorization
127.0.0.1:5000
SAR to Optical Image Colorization
Choose File
ROIs1970_f...8_p163.png
Convert
Output:


Distribution of PSNR Values on Test Set
Frequency
PSNR Values
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
8
9
10
11
12
13
14
15


PSNR Values for Each Test Image
PSNR Value
Test Image Index
15
14
13
12
11
10
9
8
0
5
10
15
20


Training Progress Over Epochs
Generator Loss
Discriminator Loss
Loss
Epoch
60
50
40
30
20
10
0
0
50
100
150
200
250


Average Generator and Discriminator Loss Per Epoch
Generator Loss
Discriminator Loss
Average Loss
Epoch
0
10
20
30
40
50
60
0
50
100
150
200
250

13. REFERENCES

[1]. A benchmarking protocol for SAR colorization: From regression to deep learning approaches. Kangqing Shen, Gemine Vivone, Xiaoyuan Yang, Simone Lolli, Michael Schmitt 01 Jan, 2024 - Neural Networks (Elsevier BV) - Vol. 169, pp 698-712.

[2]. Image Colorization, Saurabh Sunil Gaikwad 31 May, 2024 International Journal for Research in Applied Science and Engineering Technology.

[3]. A Study on the Improvement of SAR Image Colorization Performance Using CUT and SPatchGAN Discriminator Seung-Min Shin, Han Seo Oh, D. Chung 30 Apr, 2023 Journal of The Korean Society for Aerona Vol. 51, Iss: 4, pp 273-280.

[4]. A Critical Examination of SAR Colorization Impact on Flood Mapping Accuracy. Nour Aburaed,Mina Al-Saad,M. Sami Zitouni,Mohammed Q. Alkhatib,Saeed Al Mansoori,Hussain Al-Ahmad 07 Jul 2024, pp 8504-8508.

[5]. Multi-Band and Polarization SAR Images Colorization Fusion . Xinchen Li, Dan Jing, Yachao Li, Liang Guo, Liang Han, Qing Xu, Mengdao Xing, Yihua Hu 18 Aug 2022, - Remote sensing - Vol. 14, Iss: 16, pp 4022-4022.

[6]. Implicit coding of scatterer height and other anisotropic behaviors in colorized SAR imagery. Brian D. Rigling,Uttam Majumder,Edmund G. Zelnio 07 Jun 2024.

[7]. Labeling Dataset Based Colorization of SAR Images Using Cycle GAN Sam Young Lee, D. Chung 01 Oct ,2022 – The Journal of Korean Institute of Elect - Vol. 33, Iss: 10, pp 776-783.