Hyperspectral Image Denoising using Attention and Adjacent Features Extraction Hybrid Dense Network

Project Team

Muhammad Allah Rakha	19P-0006
Farkhanda Saleem	19P-0004
Mina Riaz	19P-0099

Session 2019-2023

Supervised by

Fazle Basit

Co-Supervised by

Dr. Muhammad Amin

Department of Computer Science

National University of Computer and Emerging Sciences
Peshawar, Pakistan

December, 2023

Student's Declaration

We declare that this project titled "*Hyperspectral Image Denoising using Attention and Adjacent Features Extraction Hybrid Dense Network*", submitted as requirement for the award of degree of Bachelors in Computer Science, does not contain any material previously submitted for a degree in any university; and that to the best of our knowledge, it does not contain any materials previously published or written by another person except where due reference is made in the text.

We understand that the management of Department of Computer Science, National University of Computer and Emerging Sciences, has a zero tolerance policy towards plagiarism. Therefore, We, as authors of the above-mentioned thesis, solemnly declare that no portion of our thesis has been plagiarized and any material used in the thesis from other sources is properly referenced.

We further understand that if we are found guilty of any form of plagiarism in the thesis work even after graduation, the University reserves the right to revoke our BS degree.

Muhammad Allah Rakha Signature: _____

Farkhanda Saleem Signature: _____

Mina Riaz Signature: _____

_____

Verified by Plagiarism Cell Officer

Dated:

Certificate of Approval

The Department of Computer Science, National University of Computer and Emerging Sciences, accepts this thesis titled *Hyperspectral Image Denoising using Attention and Adjacent Features Extraction Hybrid Dense Network*, submitted by Muhammad Allah Rakha (19P-0006), Farkhanda Saleem (19P-0004), and Mina Riaz (19P-0099), in its current form, and it is satisfying the dissertation requirements for the award of Bachelors Degree in Computer Science.

Supervisor

Fazle Basit Signature: _____

Co-Supervisor

Dr. Muhammad Amin Signature: _____

_____

Coordinator of FYP

FYP Coordinator
National University of Computer and Emerging Sciences, Peshawar

_____

Dr. Head of Computer Science Department

HoD of Department of Computer Science
National University of Computer and Emerging Sciences

Acknowledgements

We would identical to express our deepest gratitude and appreciation to all those who have contributed to the successful completion of this final year project on "Hyperspectral Image Denoising using Attention and Adjacent Features Extraction Hybrid Dense Network". First and foremost, We are appreciative to my project Fazle Basit (Supervisor) and Dr. Muhammad Amin (Co-Supervisor) for their invaluable guidance, support, and encouragement throughout the project. Their intuitive feedback and expertise in the field of Hyperspectral Image Denoising have contributed to configuring the direction of this research. We would also like to extend my gratitude to the faculty members of Computer Science for their valuable input and guidance in configuring my understanding of the subject matter. Their unwavering support has contributed to laying the foundation for this FYP project. We would like to gratitude FAST-NUCES Peshawar, for providing us with access to their computer system, which was indispensable for conducting this research. The cooperation and assistance from their staff were invaluable, and We are appreciative of their support. We are also thankful to my friends and family for their unwavering support and inspiration throughout the project. Their persistent motivation and belief in our abilities have been a source of strength for the team. Finally, We would identical to express our gratitude to the Almighty for his blessings, which have enabled the team to successfully complete this project.

Thank you all for your indispensable contributions and support, without which this project would not have been realizable.

Muhammad Allah Rakha

Farkhanda Saleem

Mina Riaz

Abstract

The performance of high-level semantic tasks in hyperspectral images (HSIs) is often unsatisfactory due to the complexity of the imaging environment, corruption, and degeneration. These factors result in different types of noise in HSIs. Although natural image denoising methods have been successful, existing CNN-based HSIs denoising methods continue to face the challenge of inadequate noise suppression and insufficient feature extraction. Therefore, there is still significant room for improvement in this field. To address these challenges, a novel HSIs denoising algorithm based on Attention and Adjacent Features - Hybrid Dense Network (AAFHDN) has been proposed. This algorithm is capable of decomposing high-frequency features, preserving geometrical characteristics as structure prior, and extracting band correlation of the adjacent spatial and multiscale separable spectral features. The effectiveness of the proposed method has been evaluated through experiments on simulated and real-world noisy images. The results demonstrate that the AAFHDN algorithm outperforms existing traditional methods in both quantitative evaluations and visual effects. The improved denoising performance of the proposed method can benefit subsequent classification and target detection tasks in HSIs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hyperspectral imagery has gained increasing significance across sectors such as agriculture, medicine, remote sensing, and military applications, owing to its ability to provide comprehensive spectral and spatial information for every scene block. Nevertheless, these images often encounter challenges in terms of noise, which hinders clear material identification. To tackle this issue, several denoising methods have been devised. These methods involve transforming the hyperspectral image into a matrix, where each column represents the spectral information of a pixel [1]. In contrast to conventional matrix-based techniques that lack spectral information, modern denoising methods employ nonlocal principles [2]. These methods calculate averages based on all the pixels in the image and group similar pixels together [3]. A multitude of denoising techniques have been specifically developed for hyperspectral images. These techniques utilize approaches like sparse representation and low-rank modeling to eliminate noise [17]. While many existing methods globally exploit the low-rank property of images, it is essential to address this property locally [4]. Spatially and spectrally adjacent pixels exhibit strong correlations, enabling the identification of noise-free pixel blocks [5].

Denoising plays a crucial role in hyperspectral image applications, including material identification, military operations, medicine, and agriculture. Denoising techniques continue to evolve to meet the specific requirements of these fields, thereby enhancing the quality and accuracy of hyperspectral data processing.

Chapter 2

Review of Literature

Hyperspectral imaging has attracted considerable interest across diverse domains for its capacity to capture comprehensive spectral data. Nonetheless, these images frequently contend with noise, impacting quality and subsequent analysis. Denoising methods strive to alleviate noise while maintaining critical spectral and spatial details. This literature review delves into cutting-edge approaches for hyperspectral image denoising, encompassing spectral and spatial filtering, wavelet transform, principal component analysis (PCA), sparse representation, deep learning, global low-rank representation, and local low-rank representation.

2.1 Hyperspectral Image Denoising Techniques

These mentioned approaches have use to Hyperspectral Image Denoising techniques.

- Spectral Filtering
- Spatial Filtering
- Wavelet Transform
- Principal Component Analysis
- Sparse Representation

- Sparse Representation
- Global low-rank representation
- Global low-rank representation

2.1.1 Spectral Filtering

These approaches utilize the spectral correlation inherent in hyperspectral data to remove noise [7]. Linear filters like the mean, median, and Gaussian filters exemplify this strategy. Spectral filters work on individual spectral bands without regard for spatial details. The disassembled components undergo spectral filtering for noise removal, enhancing the clarity of the denoised image [8].

2.1.2 Spatial Filtering

Methods for spatial filtering leverage the spatial correlation of adjacent pixels to enhance the clarity of hyperspectral images. The bilateral filter, total variation denoising, and nonlocal means filter exemplify this strategy, utilizing a nonlocal self-similarity measure in the spatial filtering process to grasp structural details and leverage redundancies within the image. These approaches take into account both spectral and spatial information [6].

2.1.3 Wavelet Transform

Methods based on wavelets break down the hyperspectral image into distinct frequency bands using wavelet transforms [11]. Denoising is achieved by applying thresholding to the wavelet coefficients within each frequency band [13]. Techniques for wavelet denoising encompass the wavelet shrinkage method and wavelet packet denoising.

2.1.4 Principal Component Analysis (PCA)

Denoising techniques based on Principal Component Analysis (PCA) aim to diminish noise by projecting the hyperspectral image onto a lower-dimensional subspace [16].

Components with lower eigenvalues, presumed to depict noise, are either eliminated or adjusted. The denoised image is subsequently reconstructed using the retained principal components. PCA is predominantly employed to dynamically choose and amalgamate principal components, boosting denoising effectiveness [18].

2.1.5 Sparse Representation

These approaches leverage the sparse nature of hyperspectral data in specific transform domains. Techniques rooted in sparse coding, dictionary learning, and compressed sensing exemplify this strategy. The objective of sparse representation methods is to express the hyperspectral image using a limited set of crucial coefficients, contributing to effective denoising [20].

2.1.6 Deep Learning

Recent progress in advanced learning techniques has demonstrated encouraging outcomes in the denoising of hyperspectral images. Convolutional neural networks (CNNs) can be instructed to understand the relationship between images with noise and those without [22]. Derivatives of CNNs, including U-Net [25] and residual networks [23], have been utilized in the denoising of hyperspectral images.

2.1.7 Global Low-Rank Representation

The global low-rank concept posits that image data can be expressed as a low-rank matrix, treating the entire image as a unified matrix [26]. The low-rank attribute suggests that the matrix contains a limited number of prominent singular values, signifying the primary components or patterns in the image. Through the decomposition of the image into low-rank and sparse elements, the low-rank component encapsulates the fundamental structure or key features of the image.

2.1.8 Local Low-Rank Representation

Conversely, local low-rank considers the spatial proximity or local configuration of the image, assuming distinct regions may have differing low-rank characteristics. Instead of treating the entire image as a singular matrix, it is segmented into smaller local patches or blocks. Each patch is regarded as possessing its unique low-rank structure [27]. This technique can be utilized for noise reduction through various iterative procedures, marking it as a cutting-edge strategy.

In conclusion, the intricate task of hyperspectral image denoising necessitates a nuanced fusion of spectral and spatial insights. Advanced techniques like spectral and spatial filtering, wavelet transform, PCA, sparse representation, deep learning, and global/local low-rank representation exhibit promise in noise reduction and enhancing hyperspectral image quality. Choosing the right denoising approach depends on specific data requirements, characteristics, and the desired balance between noise reduction and preservation of crucial spectral and spatial features. Future research is crucial for advancing hyperspectral image denoising techniques and expanding their applicability across diverse domains. The evolving nature of this field underscores the importance of continual advancements in addressing challenges associated with processing extensive hyperspectral data.

Chapter 3

Project Vision

Hyperspectral imaging (HSI) is a burgeoning technology applicable in agriculture, medicine, and remote sensing. Unfortunately, image quality is frequently marred by noise, compromising subsequent analysis accuracy. Our project endeavors to pioneer a novel HSI denoising network leveraging spectral band correlation, geometrical characteristics, and high-frequency feature decomposition to tackle this challenge.

3.1 Problem Statement

Various forms of noise frequently compromise the quality of hyperspectral images, posing challenges to accurate image analysis. Current HSI denoising methods often fall short in preserving the genuine spatial-spectral structure while eliminating noise. This underscores the necessity for an innovative HSI denoising network capable of adeptly extracting adjacent spatial and multiscale separable spectral features, mitigating global frequency noise, and upholding the authentic spatial-spectral structure of HSIs. This limitation adversely affects subsequent tasks like Target Detection and Classification, diminishing their overall performance.

3.2 Business Opportunity

The denoising network we suggest for hyperspectral images could open up novel prospects across industries like agriculture, medicine, and remote sensing. This capacity to precisely extract information from HSIs empowers companies to innovate and offer more advanced, efficient products and services, giving them a competitive edge.

3.3 Objectives

The core aim of this study is to create an advanced hyperspectral image denoising network, utilizing the distinctive aspects of spectral band correlation and geometric features, and proficiently breaking down high-frequency attributes. This envisioned network seeks to capture neighboring spatial and multiscale separable spectral features, effectively reducing global frequency noise and preserving the real HSI's spatial-spectral structure. The overarching objective is to outperform other proposed HSI methods, ensuring a balance of low runtime complexity, robustness, flexibility, and cost-effectiveness.

3.4 Project Scope

Our endeavor will center on constructing a network for denoising hyperspectral images, leveraging spectral band correlation and geometric traits for high-frequency feature decomposition. The envisaged network strives to capture adjacent spatial and multiscale separable spectral features, reduce overall frequency noise, and maintain the authentic spatial-spectral structure of hyperspectral images.

3.5 Constraints

The establishment of the hyperspectral image denoising network will be limited by the resources at hand, encompassing time, budget, and computational capabilities. Addition-

ally, we must acknowledge the constraints of current hardware and software frameworks to guarantee the network's effective deployment in diverse settings.

3.6 Stakeholders Description

Our initiative involves multiple parties, encompassing researchers, enterprises in hyperspectral imaging, and end-users. Researchers will find value in the novel denoising method, contributing to advancements. Hyperspectral imaging companies stand to gain enhanced precision in their offerings, while end-users in fields like agriculture, medicine, and remote sensing will experience improved efficiency in image analysis.

3.6.1 Stakeholders Summary

Our stakeholders encompass researchers, entities specializing in hyperspectral imaging, and end-users across diverse sectors like agriculture, medicine, and remote sensing. Researchers will find value in the novel hyperspectral image denoising method, while companies and end-users stand to gain from heightened accuracy and efficiency in subsequent image analysis endeavors.

3.6.2 Key High-Level Goals and Problems of Stakeholders

Our stakeholders' primary objectives are to enhance precision and efficiency in image analysis across sectors like agriculture, medicine, and remote sensing. The predominant challenge lies in the deterioration of hyperspectral image quality due to noise, impacting analysis accuracy. Our project targets this crucial concern, aspiring to deliver a solution that caters to the needs of all stakeholders.

Chapter 4

Software Requirements Specifications

This chapter will have the functional and non functional requirements of the project.

4.1 List of Features

Leverages spectral band correlation and geometric traits for high-frequency feature decomposition. Captures adjacent spatial and multiscale separable spectral features. Attenuates global frequency noise levels. Maintains the genuine spatial-spectral information structure of HSIs. Yields well-constructed outcomes surpassing other proposed HSI methods. Exhibits relatively low runtime complexity. Showcases a resilient, adaptable, and economical network.

4.2 Functional Requirements

The capability to intake hyperspectral images. The capability to derive spectral band correlation and geometric attributes. The capability to break down high-frequency features. The capability to obtain adjacent spatial and multiscale separable spectral features. The capability to diminish global frequency noise levels. The capability to maintain the authentic spatial-spectral information structure of HSIs. The capability to generate denoised hyperspectral images.

4.3 Quality Attributes

Efficient noise reduction. Retention of genuine spatial-spectral information structure. Minimal runtime complexity. Resilience, adaptability, and economical performance.

4.4 Non-Functional Requirements

Intuitive user interface. Adaptability to diverse operating systems. Suitability for various hardware platforms. Optimal memory usage. Effective processing of sizable hyperspectral images.

4.5 Use Cases/ Use Case Diagram

Introducing hyperspectral images. Executing the denoising process using the hyperspectral image network. Generating denoised hyperspectral images. The workflow diagram in our project aligns with this illustration.

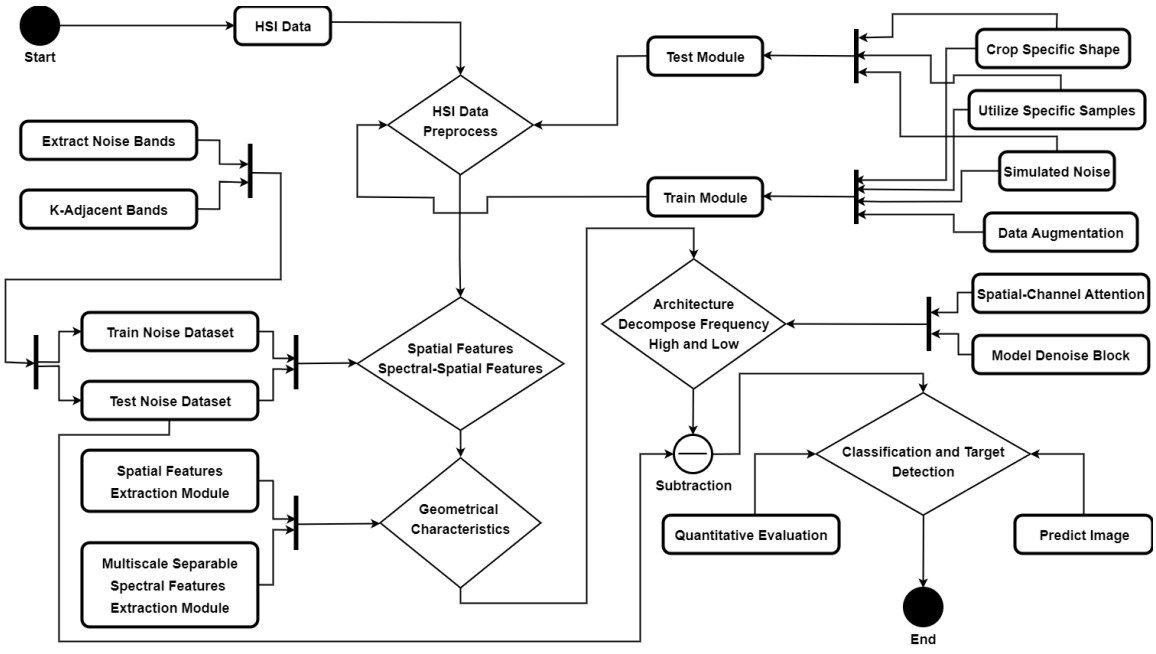


Figure 4.1: The generalized activity method of hyperspectral image denoising.

4.6 Sequence Diagrams/System Sequence Diagram

Feeding hyperspectral images into the network, initiating the decomposition of high-frequency features, and extracting adjacent spatial and multiscale separable spectral features. This process reduces global frequency noise while maintaining the inherent spatial-spectral information structure in actual hyperspectral images.

These steps have used to process HSI Denoising.

- Hyperspectral Image (HSI) Data: The refined dataset is generated through the application of preprocessing methods like cropping, simulated noise, data augmentation, and specified batch size samples. The dataset is organized into training, testing, and matric shapes.
- Spectral Band Correlation: The utilization of spectral bands in which correlation through K-Adjacent noisy bands.
- Geometrical Characteristics: The utilization of feature extraction modules on both case of spatial and spectral to preserve geometrical characteristics in structured prior information.
- Decompose Frequency (High and Low): The hybrid dense network based on attention modules (spatial and channel) is used to decompose frequency of higher and lower noisy features.
- Classification and Target Detection: The predicted denoise results (loss) are substituted by noisy band features.

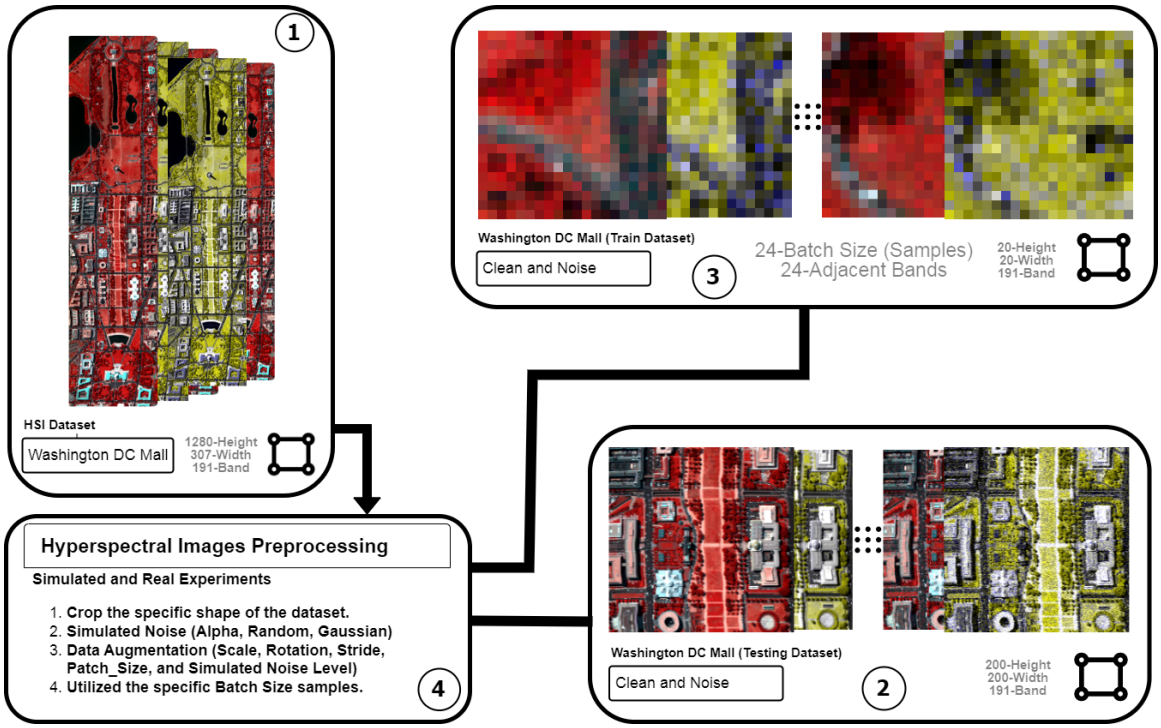


Figure 4.2: The generalized preprocessing method of hyperspectral image denoising.

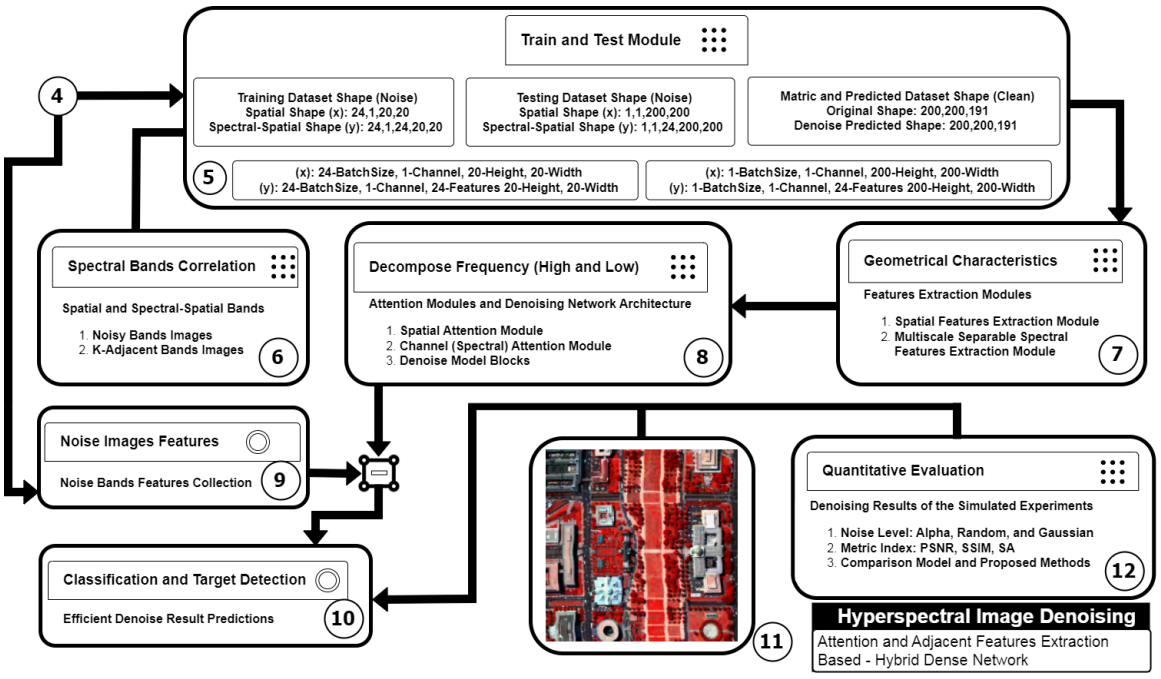


Figure 4.3: The generalized architecture method of hyperspectral image denoising.

4.7 Test Plan (Test Level, Testing Techniques)

Component testing: Examining individual elements of the network. Interconnection testing: Evaluating the amalgamation of various network components. System-wide testing: Assessing the entire system's compliance with specifications. Efficiency evaluation: Appraising the denoising effectiveness and computational complexity of the network. Validation testing: Verifying the network's performance with authentic hyperspectral images to meet stakeholder expectations.

4.8 Software Development Plan

Analysis of requirements: Collecting and evaluating the specifications for the network. Structuring: Creating the network's architecture and algorithms. Execution: Putting the network into practice using appropriate programming languages and tools. Verification: Ensuring the network complies with the specified requirements. Rollout: Introducing the network across different hardware and software platforms. Sustainment: Sustaining and updating the network as required.

Chapter 5

Iteration Plan

This chapter outlines the project's iteration plan, detailing the progression to fulfill project requirements. It provides guidance on module development and execution phases. Students need to discuss the project's execution plan in this chapter:

- Midterm FYP 1: During our Midterm FYP 1, we conducted thorough research into the intricate realm of Hyperspectral Image denoising. This involved delving into the complexities of HSI spectral data, examining nuanced noise reduction techniques. Our exploration covered a detailed review of 40 research papers, the investigation of diverse HSI datasets, and the identification of a precise problem statement.

- Final FYP 1: In the scope of our concluding FYP 1, our investigative pursuits delved deeper into the nuanced realm of Hyperspectral Image denoising. This involved an extensive examination of the fundamental principles guiding the HSI spectral model and the intricate architectural paradigms for denoising. This phase of our work encompassed a thorough analysis of approximately 20 research papers, providing us with a profound grasp of the complexities inherent in HSI denoising methodologies. Concurrently, we developed a sophisticated framework for Hyperspectral Images Preprocessing. Within this framework, we intricately structured datasets to support both simulated and real-world experiments.

  The preprocessing stage involved a sequence of intricate procedures outlined as follows: 1. Precision Dataset Cropping: The dataset was meticulously shaped to exact

specifications. 2. Simulated Noise Generation: We introduced diverse types of simulated noise, including Alpha, Random, and Gaussian noise, replicating real-world conditions. 3. Data Augmentation: Our approach employed a comprehensive data augmentation strategy, covering scaling, rotation, stride modification, patch size variations, and adjustments to simulated noise levels. 4. Optimized Batch Size Selection: We systematically determined the most efficient batch size for processing, ensuring optimal utilization of computing resources.

This thorough preprocessing stage established the groundwork for our subsequent experiments and analysis, elevating the caliber and resilience of our research results.

- Midterm FYP 2: During the Midterm FYP 2 phase, our research delved extensively into the domain of Hyperspectral Image denoising. This involved a thorough exploration of the HSI spectral model, its practical implementation in Pytorch, and the complexities of a denoising architecture enriched with Feature and Geometrical characteristics. This stage of our work included a detailed review of up to 10 research papers, fostering a deep understanding of the diverse methodologies driving HSI denoising. Furthermore, we dedicated our efforts to constructing the architectural modules for Hyperspectral Image denoising.

  In this undertaking, we executed a series of simulated trials, subjecting our study to the stringency of Sigma-50 noise levels. These trials were dedicated to honing HSI noise reduction procedures, distinguishing between High and Low frequency noise treatment, and embracing a spectrum of approaches as delineated below: 1. Training and Testing Module: A comprehensive framework for training and evaluating the denoising model. 2. Spectral Bands Correlation: Investigating the interaction of spectral bands to improve denoising efficiency. 3. Geometric Characteristics: Harnessing geometric attributes to further enhance the denoising process. 4. Frequency Decomposition (High and Low): The systematic breakdown of frequencies into High and Low bands for precise denoising. 5. Classification and Target Detection: Utilizing classification methods and target detection strategies to enhance denoising results. 6. Quantitative Assessment: A thorough evaluation methodology to measure the effectiveness of our denoising procedures.

This extensive research endeavor signifies a notable stride in pushing forward the domain of Hyperspectral Image denoising, demonstrating our dedication to investigating state-of-the-art methodologies and approaches.

- Final FYP 2: In the scope of our concluding FYP 2, we broadened our research scope to include a set of thorough simulated experiments, incorporating various noise models like Random and Gaussian. This exhaustive comparison involved benchmarking against cutting-edge models like MemNet and DeNet, enabling a meticulous assessment of the proposed HSI denoising approach.

  Our study concluded with the finalization of an exhaustive report, synthesizing our discoveries and the innovative HSI denoising approach crafted to tackle three common issues. The elaborated method, underpinned by a nuanced grasp of Spectral Band Correlation, Geometrical Characteristics, and the systematic Breakdown of Frequency into High and Low bands, presents a compelling resolution to these issues.

Chapter 6

Iteration 1

The first stage is anticipated to finish before the midpoint of FYP-1. Throughout the Midterm FYP 1 phase, our focus was committed to a thorough exploration in the complex realm of Hyperspectral Image denoising. Our research involved a detailed examination of HSI spectral data and refined techniques for noise reduction. This leg of our research journey encompassed the following intricate steps:

1. A thorough examination of a substantial collection of 40 research papers, providing a comprehensive insight into the dynamic developments in hyperspectral image denoising.

2. Exploring a variety of HSI datasets that formed the basis for practical examinations.

3. A thorough exploration of the vast knowledge within the domain, focusing on extracting essential insights and leading to the precise articulation of a clearly defined problem statement.

6.1 Introduction

Hyperspectral imagery, acquired through aerial or satellite sensors, delivers comprehensive information about target area objects across multiple spectral bands. It integrates both spatial and spectral details into pixel-based vectors, indicating feature strength at specific

coordinates. Refer to the HSI process in Figure-6.1 for a visual representation, and delve into further details [28].

- Hyperspectral images are produced using sensors on airborne or satellite platforms, capturing data across a defined geographic region.
- These images contain information about diverse objects in the target area, spanning continuous and segmented spectral bands from visible light to the infrared.
- Hyperspectral Images (HSIs) uniquely acquire both spatial and spectral data simultaneously, creating a comprehensive dataset.
- Pixels play a crucial role in hyperspectral data analysis, with spatial and spectral information organized into these fundamental units.
- Each pixel is a multi-dimensional vector, capturing the intensity across different spectral bands at specific spatial coordinates (x, y). This vectorization represents the rich multi-spectral information in hyperspectral data.

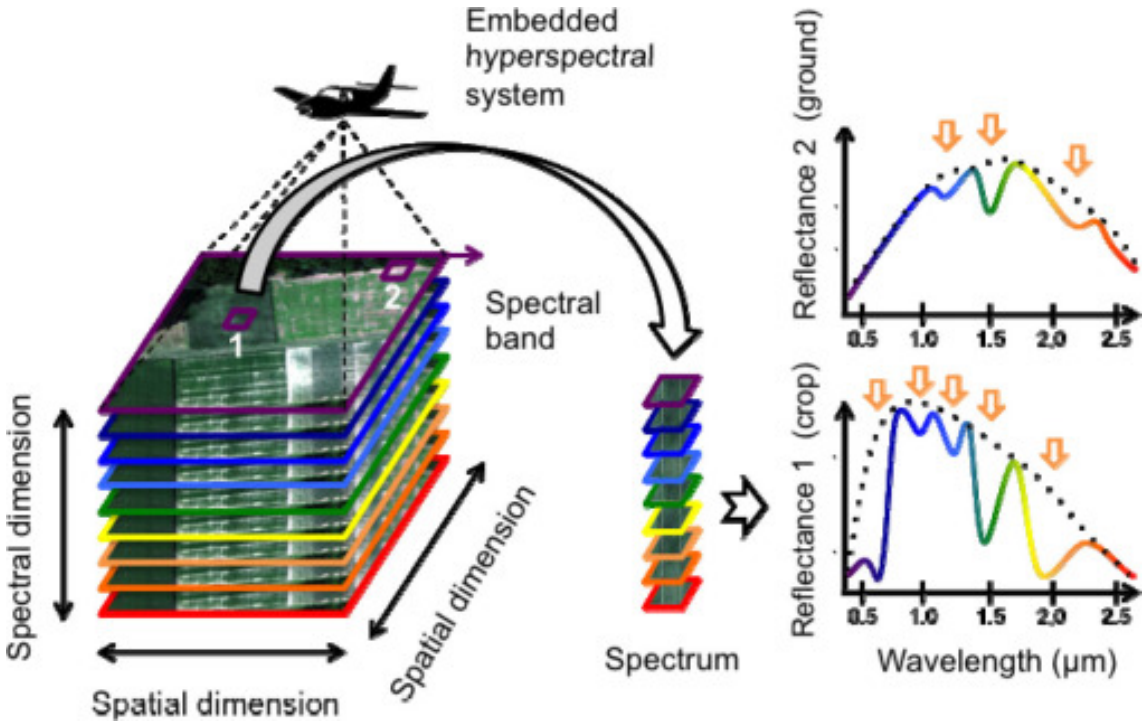


Figure 6.1: The structure of a Hyperspectral Image, in which the spatial and spectral dimensions and Wavelength with different reflectances.

6.2 Related Work

- A Comprehensive Attention-Based Network for Hyperspectral Image Denoising, adept at handling both inter- and intra-spectral correlations, excelling in noise experiments, boasting minimal parameters and runtime, and effectively integrating inter-spectral features, is detailed in the 2023 reference paper [29].

- The Deep Spatial-Spectral Global Reasoning Network for Hyperspectral Image Denoising innovatively incorporates global contextual insights, efficiently reducing hyperspectral noise by combining local and global data, surpassing existing denoising methods, and mastering complex noise challenges using multiple representations, as outlined in the 2022 reference paper [30].

- Our research introduces a novel Hyperspectral Image Denoising approach, merging a Weighted Nonlocal Low-Rank Model with Adaptive Total Variation Regularization. This technique utilizes non-independent and nonlocal similarity, edge-preserving total variation regularization, and extensive ADMM network experiments, demonstrating superior performance over state-of-the-art methods, as detailed in the 2022 reference paper [31].

- Described in the 2022 paper [32], this work focuses on the rapid denoising and inpainting of hyperspectral images, addressing Gaussian and Poissonian noise, and missing observations. It achieves lower computational complexity by fully leveraging compact and sparse HSI representations, particularly emphasizing low-rank and self-similarity features.

- LR-Net leverages low-rank features to capture latent semantic connections in hyperspectral images, exhibiting superior denoising performance validated through extensive experiments. Atrous blocks explore spatial-spectral features, aggregated via a multi-atrous block, and combined with spatial-spectral attributes in a low-rank module (Reference: [33], 2021).

- Utilizing a 3D attention denoising network, we achieve hyperspectral image denoising that concurrently processes spatial and spectral information, managing global

dependence and correlation. Experimental results on both simulated and real data support the superior quality of our method, incorporating position and channel attention modules for spatial and spectral features, and employing multiscale feature extraction and fusion. Referenced from [34] in 2021.

- Hyperspectral Image Denoising integrates sparse-based low-rank representation, exploring global spatial and spectral correlations, with a CNN-based denoiser representing deep prior restoration models. This leads to superior denoising results in both simulated and real data experiments within a flexible and extensible framework, as detailed in the 2021 reference paper [35].

- Spatial-spectral weighted nuclear norm minimization is applied in hyperspectral image denoising, involving weighted nuclear norm minimization to recover spectral and spatial LR structures. Validated for denoising efficiency and visual quality through experiments on both simulated and real HSIs datasets, the approach incorporates spatial domain nonlocal similar cubic patches stacked into an LR matrix containing local spatial texture information. Detailed in reference paper [36] from the year 2020.

- The Enhanced Non-Local Cascading Network with Attention Mechanism (ENCAM) excels in extracting combined spatial-spectral features for Hyperspectral Image Denoising. This surpasses other cutting-edge methods, backed by both theoretical analysis and experiments. ENCAM extends non-local structures and incorporates multi-scale convolutions with channel attention modules, as outlined in the 2020 reference paper [37].

- HSISDeCNN, a singular CNN model, effectively utilizes spatial and spectral data for hyperspectral image denoising. It demonstrates superior results in synthetic and real experiments, capitalizing on the high spectral correlation in adjacent bands of HSIs. This work is detailed in the 2020 paper [38].

- The innovative TenSRDe approach employs tensor subspace representation, reducing computational complexity and efficiently denoising hyperspectral images. Supported by experiments on simulated and real datasets, it affirms its denoising effi-

cacy, particularly for tensors with low tubal rankness and nonlocal self-similarity. Details are available in the 2020 reference paper [39].

- Employing a Spatial-Spectral Deep Residual Convolutional Neural Network, this approach establishes a nonlinear mapping from noisy to clean HSIs. It demonstrates superior performance in both simulated and real-data experiments across various evaluation metrics, including visual effect and classification accuracy. This is outlined in the 2018 reference [40].

6.3 Problem Statement

Cutting-edge models for Hyperspectral Images (HSIs) presently cannot leverage the properties of spectral band correlation and geometric features, constraining their ability to efficiently break down high-frequency features. This drawback adversely affects subsequent processing tasks associated with Target Detection and Classification.

6.4 System Design

- Hyperspectral Image (HSI) Data Preparation: The dataset undergoes meticulous filtering through various preprocessing techniques, including cropping, introducing simulated noise, data augmentation, and specifying batch sizes. The structured dataset comprises Training, Testing, and Matric components.

- Spectral Band Connectivity: Our methodology integrates spectral bands, concentrating on their correlation within K-Adjacent noisy bands. This analysis of correlation plays a crucial role in our HSI data processing approach.

- Preservation of Geometric Features: A core principle involves maintaining intricate geometric characteristics within the data. To achieve this, we employ feature extraction modules operating on both spatial and spectral aspects, ensuring the retention of underlying geometric features and structured prior information critical for subsequent analysis.

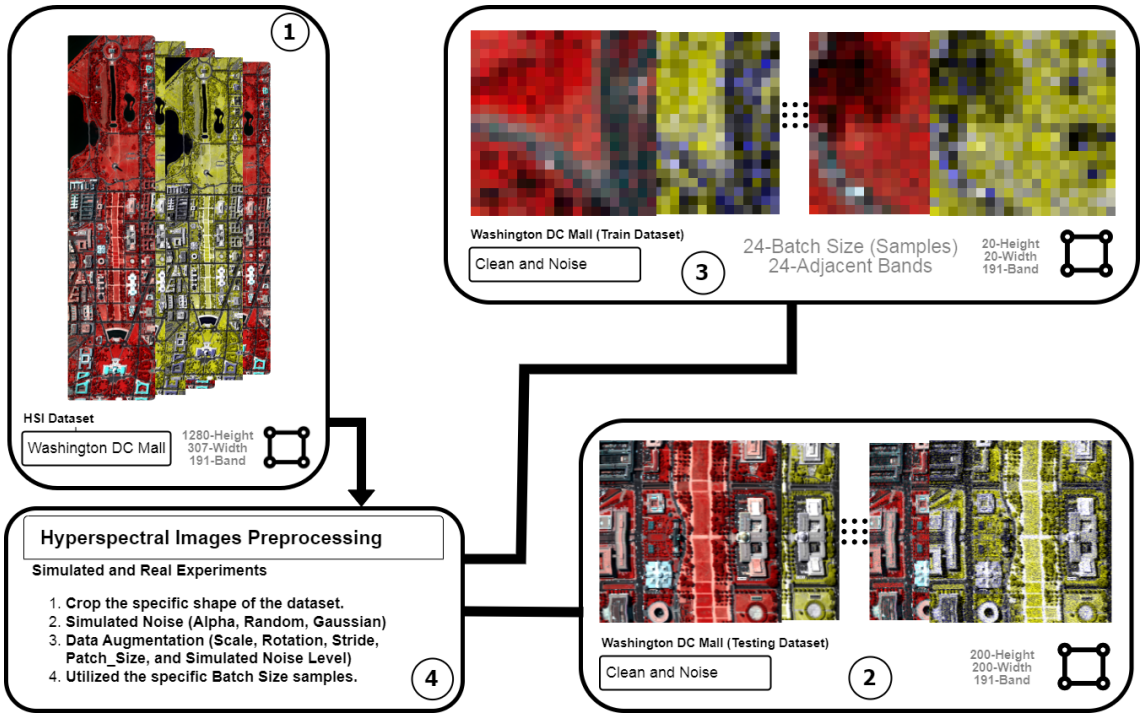
- Frequency Decomposition (High and Low): Our strategy involves using a hybrid dense network with attention modules across spatial and channel domains. This configuration effectively decomposes frequency components within the data, specifically targeting the separation of higher and lower frequency noisy features.

- Image Classification and Target Detection: The denoising process results in predicting clean data (loss), utilized in image classification and target detection. Substituting the predicted denoised results for noisy band features facilitates more accurate downstream analysis, enhancing outcomes in image classification and target detection.

6.5 Component Diagram Illustration

The diagram of components presents a detailed representation of how individual modules sequentially operate, outlining the complex processes they undergo. Each component provides insight into the structure of functionalities, illuminating the conventional denoising procedure applied to Hyperspectral Images (HSI). This diagram acts as a visual guide, leading us through the step-by-step execution of various modules contributing to the HSI denoising process. By dissecting and encapsulating each module's behavior, we develop a comprehensive understanding of the procedures involved, unraveling the complexities of traditional HSI denoising methods. Moreover, the component diagram plays a crucial role in elucidating the interdependencies among these modules, clarifying how they interact and complement one another. It serves as a valuable tool for visualizing and understanding the structural foundations of the HSI denoising procedure, providing insights into the seamless execution of these processes.

Essentially, this graphical depiction encapsulates the core of HSI denoising, elucidating its complexities and rendering the typically intricate process more understandable for analysis and comprehension. It stands as a fundamental reference for grasping the sequential and structural elements of the conventional HSI denoising methodology.



6.6 Activity Diagram Illustration

The activity diagram vividly portrays the systematic progression of each module in the system, elucidating the orchestrated flow of functionalities within Hyperspectral Images' (HSI) traditional denoising process. It intricately details the functional structure, offering a profound insight into quantitative evaluation and denoised image prediction.

This diagram provides a dynamic narrative, visually articulating how modules interact with the larger system, depicting the actions they perform and their sequential order. It paints a vivid picture of the operational choreography within HSI's traditional denoising framework, showcasing the collaborative contributions of these components.

Beyond outlining the steps, the diagram explores the functional flow structure, serving as a guide through the intricate landscape of traditional HSI denoising. It reveals the intricate paths of various modules, each playing a crucial role in the overall process.

Moreover, the activity diagram incorporates quantitative evaluation and denoised image prediction, emphasizing these vital aspects of HSI denoising. By illustrating activity flows, it elucidates how quantitative assessment and denoised image prediction seamlessly integrate into the process, shedding light on the denoising goals and outcomes.

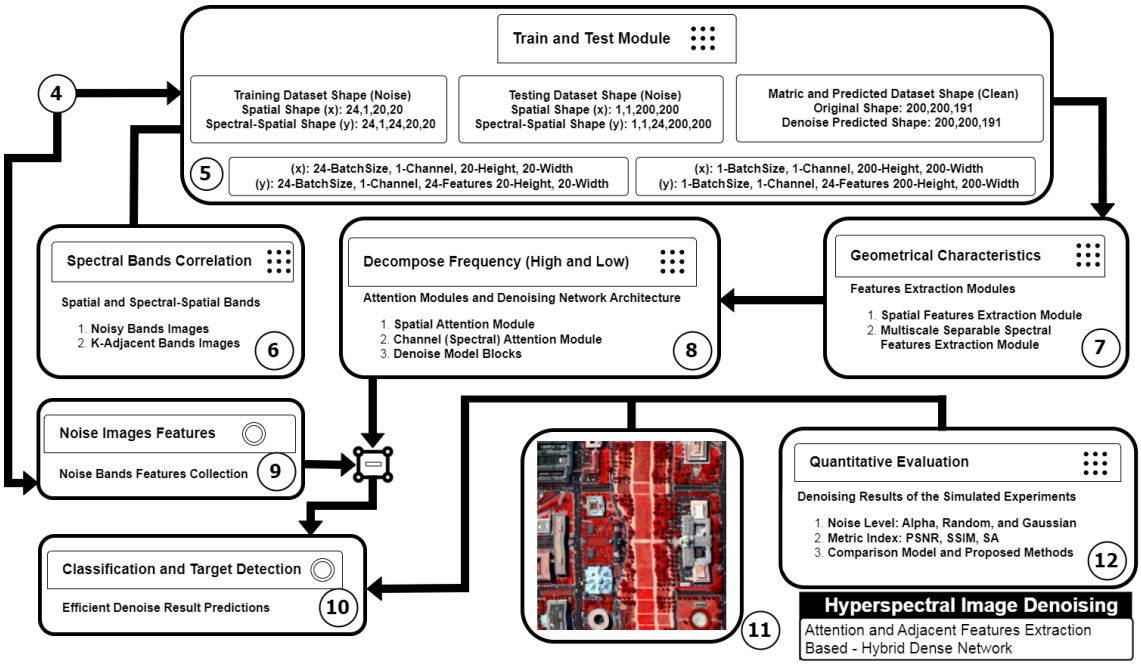


Figure 6.2: The component diagram offers a meticulous breakdown of each module's sequential behavior, serving as an illustrative guide through HSI's traditional denoising procedure. Each module's depiction sheds light on the underlying structure of functionalities.

Now, let's through the intricate details of each component's functionalities.

The process of training and testing Hyperspectral Images (HSI) involves a systematic approach to data preparation, guided by two crucial modules. These components go beyond managing raw data; they play a vital role in crafting features that navigate the terrain of prior information, a crucial step towards meaningful HSI analysis.

- Precision through Cropping: The module "Crop Specific Shape" ensures targeted extraction, precisely selecting an area within the extensive HSI dataset tailored to specific analytical requirements.

- Enhancing with Specific Samples: The module "Utilize Specific Samples" takes prominence, amplifying data richness by focusing on identified band samples. This augmentation deepens the dataset, offering a more comprehensive resource for training and testing.

- Experimenting with Simulated Noise: Vital for assessing HSI data robustness, the "Simulated Noise" module introduces various noise types for a thorough evaluation,

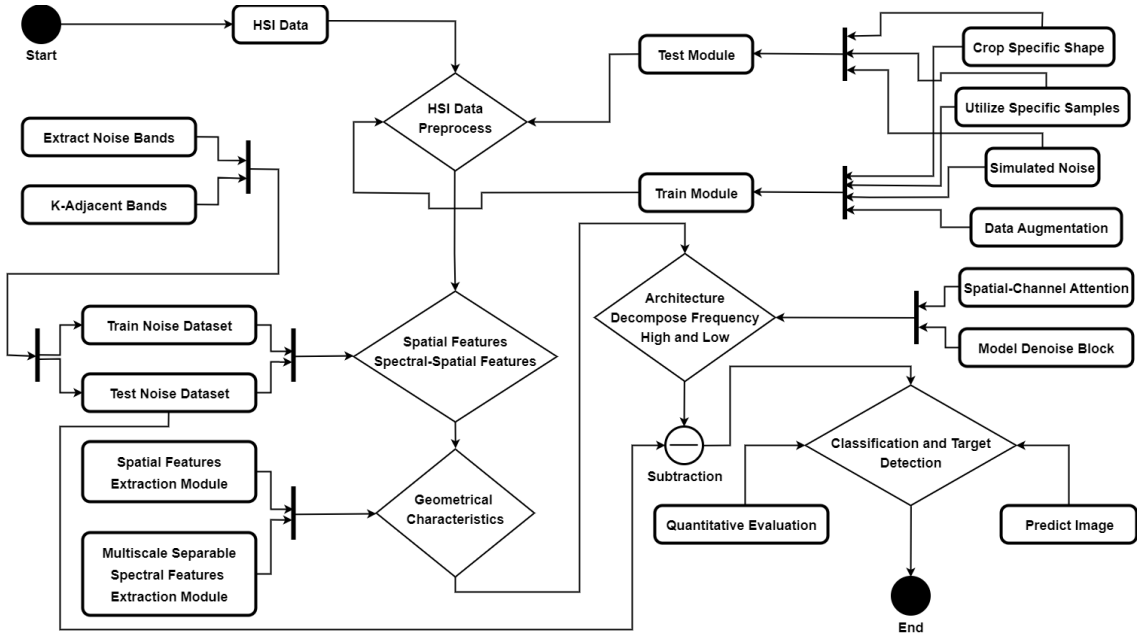


Figure 6.3: The activity diagram provides a detailed, step-by-step depiction of module flow, class interactions, and functional operations within HSI's denoising procedure, including crucial elements like quantitative assessment and denoised image prediction.

particularly in terms of Peak Signal-to-Noise Ratio (PSNR). This not only evaluates data quality but also sets the stage for crucial noise removal in HSI analysis.

- Enrichment through Data Augmentation: The "Data Augmentation" module adds complexity to the dataset by introducing diverse features, often derived from noise. This diversity enhances dataset robustness for effective model training.

Preparing data for training and testing in the domain of Hyperspectral Images (HSI) involves a multifaceted process, employing two crucial modules to define the dataset's content and structure. These modules go beyond simple data preparation, adopting a sophisticated approach to enhance the dataset for thorough analysis, playing a vital role in exploring HSI datasets effectively.

- Revealing Band Connections: Central to this procedure is the "Extraction Noise Bands" unit, intricately crafted to expose the subtle connections among bands. This task, executed on designated channels, unveils the inherent band correlations in the dataset. By isolating the band correlation of specific channels, this unit establishes a profound comprehension of the interrelationships between spectral bands, a crucial

aspect of HSI examination.

- K-Adjacent Band Associations: Conversely, the "K-adjacent bands" unit delves into the spectral domain, constructing a resilient spectral correlation framework. It constructs an intricate network of associations, methodically linking the initial and terminal bands across channels. This elaborate network of spectral correlations unfolds fresh perspectives for channel feature scrutiny, establishing the groundwork for deeper exploration into the complexities of the HSI dataset.

Within the domain of Hyperspectral Image (HSI) denoising, the groundwork is laid through data preparation. This elaborate procedure involves the synergistic operation of two crucial modules. These modules collaborate not only in managing data but also in actively shaping the underlying characteristics essential for the denoising model.

- Module for Extracting Spatial Features: The "Spatial Features Extraction Module" assumes a pivotal role in the data preparation phase. Its primary function is to retrieve spatial samples from HSI, crucial for understanding the spatial intricacies of the data. What distinguishes this module is its flexibility, extracting samples using a diverse range of kernel filter sizes. This adaptability facilitates a nuanced analysis of spatial information, accommodating variations in spatial scale.

- Module for Extracting Multiscale Separable Spectral Features: In tandem with spatial considerations, the "Multiscale Separable Spectral Features Extraction Module" introduces a versatile approach. It specializes in isolating separable features, with a particular emphasis on K-adjacent band correlations and spectral-spatial aspects. This module's distinctiveness lies in its ability to function across various kernel filter dimensions, fostering a multiscale comprehension of the data. It encapsulates the intricacies of spectral-spatial features across different scales, enriching the HSI denoising model with a diverse information source.

The complexities of denoising Hyperspectral Images (HSI) are significantly heightened during the data preparation phase, where two indispensable modules play a central role. These modules go beyond mere data management, playing a pivotal role in preparing

essential High and Low features critical for the Classification and Target Detection phases of the HSI denoising model. Essentially, these modules pave the way for the model's ability to discern and classify intricate details within the HSI dataset.

- Spatial-Spectral Attention for Data Monetization: The "Spatial-Spectral Attention" module is a critical component in the data preparation process. Its function is to meticulously examine each band, extracting spatial and channel samples crucial for monetizing the specified HSI area. This detailed operation captures intrinsic details often concealed within HSI data. By utilizing spatial and channel samples, it lays the foundation for comprehensive analysis, ultimately enhancing the denoising model's classification and target detection capabilities.

- Model Denoise Block for Frequency Decomposition: Simultaneously, the "Model Denoise Block" steps in to address the decomposition of High and Low frequencies in the data. This block employs various techniques, including the "Additive Skip Connection" and "Progressive Spectral Channel Attention," integral components of the Model Blocks. These techniques are pivotal in accomplishing the intricate task of decomposing frequency features. The "Additive Skip Connection" enhances the model's ability to capture fine-grained details by adding the output of one layer to another. Meanwhile, "Progressive Spectral Channel Attention" strategically guides the model's focus, emphasizing significant areas within the spectral channels.

Essentially, this all-encompassing portrayal not only displays the series of steps but also furnishes an elaborate and detailed account of the traditional denoising process for HSI. It serves as an essential instrument for both visualization and analysis, delivering a deep insight into the complex interaction among modules and the overall progression of the denoising procedure.

6.7 Data Representation

In the pursuit of unraveling the complexities of Hyperspectral Image (HSI) analysis, the dataset assumes a crucial role, contextualized within the framework of two well-regarded

hyperspectral datasets: the Indian Pines (IP) and the Washington DC Mall (WDCM).

1. Indian Pines (IP): Acquired in 1992 using the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) sensor [43], this dataset captures the Northwestern region of Indiana. It showcases a diverse landscape with agricultural fields, geometric structures, and untouched forests. Comprising 145 x 145 pixels and 224 spectral bands (400 to 2500 nm), it boasts a spatial resolution of 20 meters per pixel. However, atmospheric water absorption affected four null bands and 20 others, leaving 200 bands for experimentation, as illustrated in Figure-6.5.

2. Washington DC Mall (WDCM): Collected under the Spectral Information Technology Application Center of Virginia [44], this dataset features a sensor pixel system with 210 bands spanning 0.4 to 2.4 µm. Rigorous selection excluded bands in the 0.9 and 1.4 µm regions, resulting in 191 bands for detailed spectral analysis. With 1208 scan lines, each containing 307 pixels, it forms a dataset of (1208-W x 307-H) shape, ready for diverse analysis and experimentation, as depicted in Figure-6.4.

These datasets, each exhibiting distinctive features and spectral composition, act as the platform for HSI analysis. They present a diverse array of data, each carrying its nuances and difficulties. The Indian Pines dataset, with its atmospheric complexities and varied terrain, provides a thorough collection of spectral bands. Conversely, the Washington DC Mall dataset encompasses extensive spectral range, deliberately excluding less informative bands. Combined, they epitomize the core of hyperspectral data analysis, teeming with potential and complexities awaiting exploration.

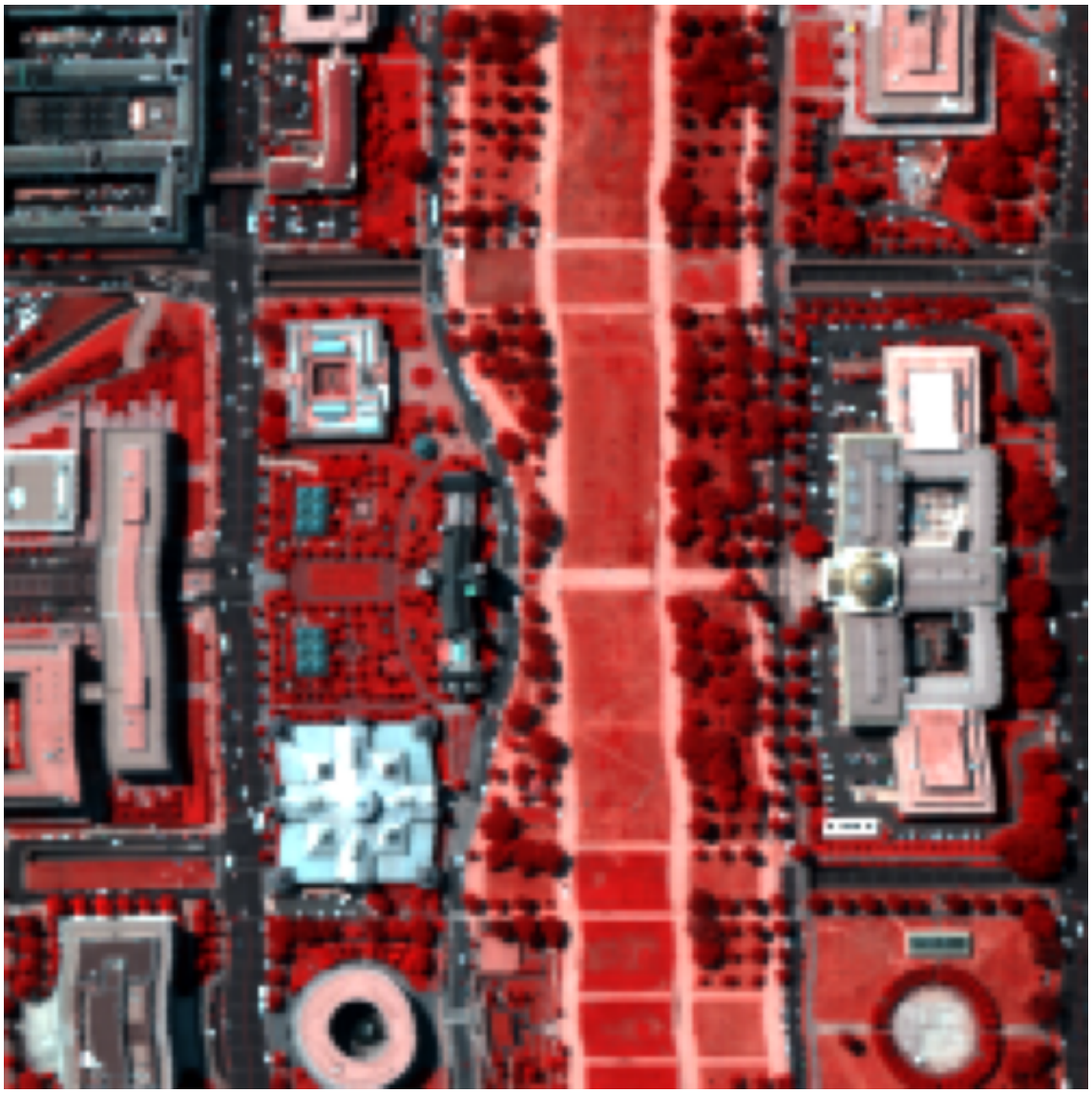


Figure 6.4: Our analysis is anchored in two prominent hyperspectral datasets, Washington DC Mall (WDCM) with its unique spectral attributes and data collection details.

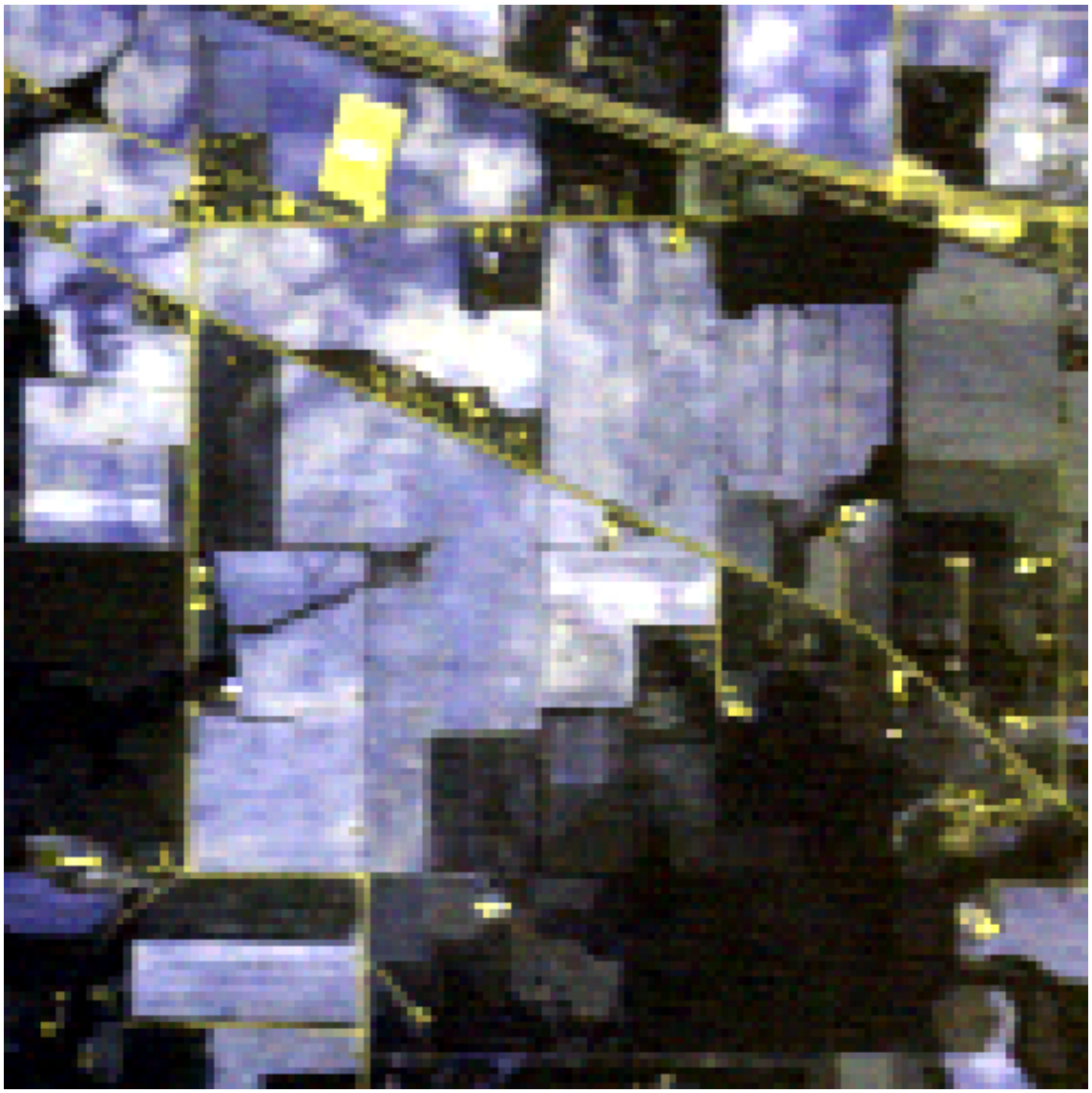


Figure 6.5: Our analysis is anchored in two prominent hyperspectral datasets, Indian Pines (IP) with its unique spectral attributes and data collection details.

Chapter 7

Iteration 2

In the concluding stage of Final Year Project 1 (FYP-1), a crucial achievement marked the progression in Hyperspectral Image (HSI) data processing. This phase involved developing a detailed procedure, integrating a module tailored for efficient data management during intensive training and testing. Utilizing benchmark datasets, namely Washington DC Mall and Indian Pines, the procedure underwent a series of experiments, encompassing four key modules adept at processing both simulated and real HSI data.

7.1 Hyperspectral Data Preprocessing

In summary, the conclusive phase of the Final FYP-1 term played a crucial role in shaping the landscape of HSI data processing. It introduced a spectrum of transformative modules and techniques, ranging from precise cropping and simulated noise incorporation to flexible data augmentation and meticulous batch size adjustments. This comprehensive strategy laid the groundwork for enhanced HSI data processing, mirroring the intricate facets of HSI analysis.

7.1.1 Crop Specific Shape

1. Precision through Cropping: A fundamental step in this process involves accurately cropping HSI data to meet the specific requirements of model training and testing.

The original dimensions of the Washington DC Mall dataset were (1280, 307, 191). During training, the dataset was cropped to a defined shape, encompassing heights from 0 to 600 and widths from 800 to 1280, retaining all 191 bands. Concatenation of the (0, 600) and (800, 1280) scales resulted in a final shape of (1080, 307, 191). For the testing dataset, a height range of (600, 800) and a width range of (50, 250) were applied, also with 191 bands. This precise cropping ensured focused model training and testing, maintaining data conformity to desired dimensions. The figure-7.1 illustrates the Washington DC Mall (a), Training Dataset (b), and Testing Dataset (c).

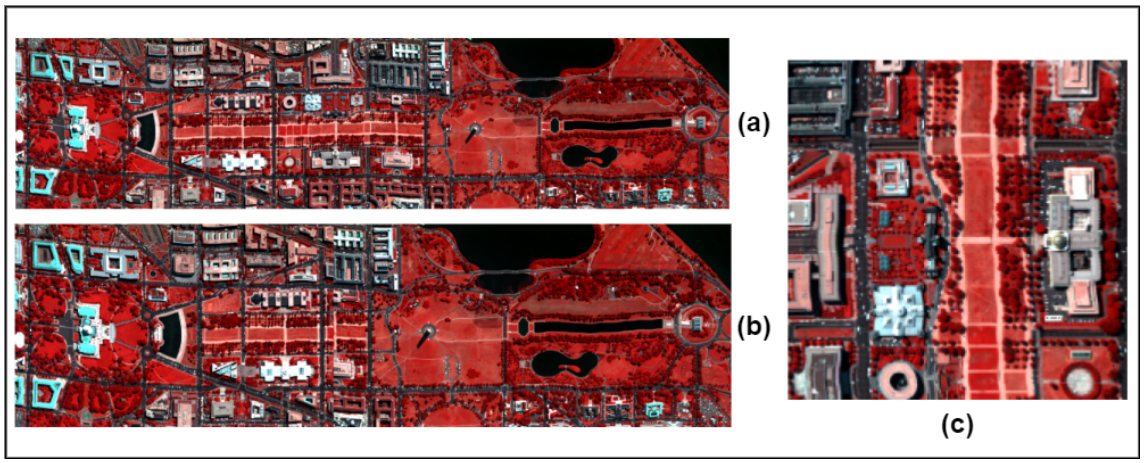


Figure 7.1: An exploration of the multifaceted data augmentation process, where parameters like Scale, Rotation, Stride, Patch Size, and Simulated Noise Level play a crucial role in molding the dataset for diverse scenarios.

7.1.2 Simulated Noise

1. Simulated Noise Experimentation: Elevating the HSI denoising process involved introducing Alpha Sigma, Random, and Gaussian noises. In this trial, the application of Alpha Sigma noise at a 50 percent intensity level added a distribution aspect to the HSI data. The test dataset retained its dimensions as (200, 200, 191). This variation in noise not only enhanced the dataset but also facilitated the investigation of denoising approaches. Figure-7.2 (a) displays the test image with added noise.

7.1.3 Data Augmentation

1. Enhancing Data Versatility: Data augmentation involved manipulating specific feature parameters such as Scale, Rotation, Stride, Patch Size, and Simulated Noise Level. For example, Scale was used to control multiscale feature dimensionality, resulting in predicted shapes of (1080, 307, 191) for Scale (1) and (2160, 614, 191) for Scale (2). Stride and Patch Size governed kernel feature search size, with 20 levels employed. Rotation managed angular dimensions with a 90-degree rotation. Simulated Noise played a vital role in denoising and enhancing HSI classification and target detection. This comprehensive data augmentation approach increased dataset versatility, enabling exploration across diverse scenarios. The code for Versatile Data Augmentation can be found in Figure 7.2 (b and c), displaying the noisy training images.

```
def data_aug(img, rot_time, filp_mode):
  if filp_mode == -1:
    return np.rot90(img, k=rot_time)
    # Rotate an array by 90 degrees in the plane specified by axes.
    # Output same as the given input, in case of k = -1
  else:
    return np.flip(np.rot90(img, k=rot_time), axis=filp_mode)
    # Reverse the order of elements in an array along the given
                                        axis.
    # filp_mode: vertically (axis=0), horizontally (axis=1).

clean = []
noise = []
aug_times = 1
scales = [1]
patch_size, stride = 20, 20

def augmentation(data_train, count = 0):
  for s in scales:
    print(f"Per number of total: {s}/{scales}")
    data_scaled = scipy.ndimage.zoom(data_train, (s, s, 1)).astype(
```

```
                                        np.float32)
    data_scaled[data_scaled < 0] = 0
    data_scaled[data_scaled > 1] = 1

    print("data scaled shape of per number (",s,") : ", data_scaled
                                        .shape)
    h_scaled, w_scaled, band_scaled = data_scaled.shape
    for i in range(0, h_scaled - patch_size + 1, stride):
        for j in range(0, w_scaled - patch_size + 1, stride):
            for k in range(0, aug_times):
                count += 1
                x = data_scaled[i:i + patch_size, j:j + patch_size,
                                                  :]

                rot_time = np.random.randint(0, 4) # Generate: 0, 1
                                                  , 2, 3
                filp_mode = np.random.randint(-1, 2) # Generate: -1
                                                  , 0, 1
                x_aug = data_aug(x, rot_time, filp_mode) # Data
                                                  Augmented

                y_aug = noise_add_by_bands(x_aug) # Add the
                                                  Simulated Noise
                x_np = np.array(x_aug, dtype='float32')
                y_np = np.array(y_aug, dtype='float32')
                clean.append(x_np)
                noise.append(y_np)
  print("Total number of augmention : ",count)

augmentation(data_train_DC, count = 0)

original = np.array(clean)
noise = np.array(noise)
print("Original Dataset Shape : ", original.shape, noise.shape)

# Used specific batch_size = 24 = K
batch_size = 24  # K = 24
```

```
original_ = original[:batch_size]
noise_ = noise[:batch_size]
print("Batch-Size Dataset Shape : ", original_.shape, noise_.shape)
```

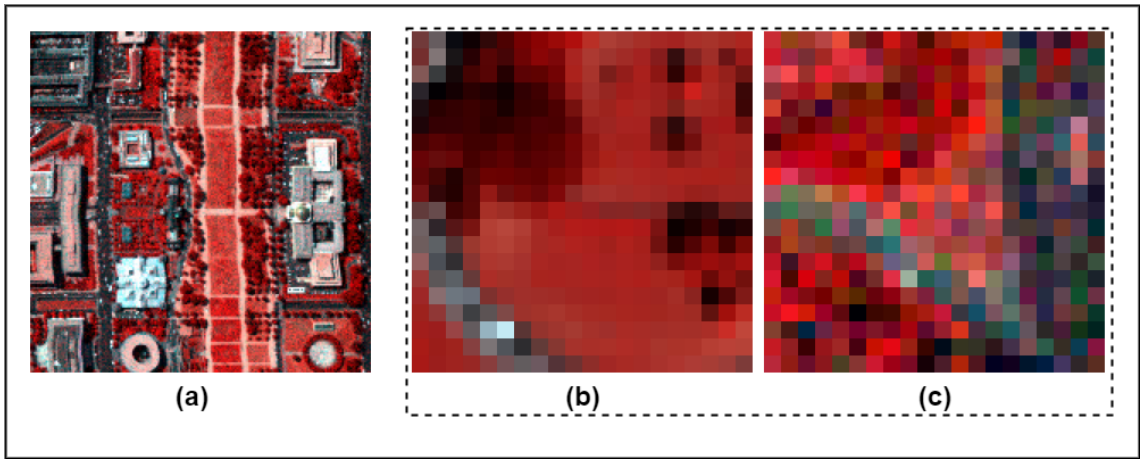


Figure 7.2: Elevating HSI Denoising with Simulated Noise

7.1.4 Batch Size Samples

1. Precision in Batch Size: The selective use of specific batch size samples significantly influenced the sample choice for model training. In this case, 24 samples were chosen. Before this selection, the dataset dimensions were (810, 20, 20, 191). After applying the batch size feature, the dimensions were streamlined to (24, 20, 20, 191). This meticulous batch size precision ensured a focused and efficient training process, maximizing computational resources and enhancing model performance.

7.2 Simulated Noise Methods

7.2.1 Gaussian Noise Illustration

Noise intensity demonstrates diversity across spectral bands, with the sigma noise level (σ_n) applied along the spectral axis, exhibiting Gaussian distribution characteristics. The

β noise intensity and η standard deviation parameters significantly influence this distribution, with σ_n derived from the Gaussian distribution using $\beta = 200$ and $\eta = 30$, where B represents the band of the training or testing dataset. The mathematical representation of this process is concisely articulated in Equation-7.1.

$$\sigma_n = \beta \sqrt{\frac{exp-(n-B/2)^2/2\eta^2}{\sum_{i=1}^{B} exp-(n-B/2)^2/2\eta^2}} \tag{7.1}$$

7.2.2 Random Noise Illustration

At the intensity of Random noise, we employed the probability density function of the standard Gaussian distribution to augment the existing Gaussian level. Here, μ denotes the mean, σ represents the standard deviation, and the square of σ denotes the variance. This phenomenon is precisely formulated in Equation-7.2.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{7.2}$$

7.2.3 Data Truncate: Washington DC Mall

In the field of image processing, the "Truncate Image" feature stands out as a powerful and versatile tool. It possesses the capability to apply linear stretch, providing precise control over image scales. This sophisticated function goes beyond technicality, serving as an artful process to effectively handle and improve spatial features within the image. In this exploration, we uncover the diverse advantages and applications of the truncate module, illuminating its multifaceted utility. The code elucidates the process of Truncate Image Manipulation.

1. Precision Range Adjustment: The truncate module, with its meticulous and targeted refinements, assumes a crucial role in adjusting the dynamic range of an image. This involves strategically isolating and removing extreme values at both ends of

the data spectrum, followed by a careful rescaling of the remaining values to cover the full display range. This method not only enhances clarity but also revitalizes the image, offering a broader spectrum of nuances.

2. Intensified Contrast: Going beyond dynamic range adjustments, the truncate module proves to be a powerful tool for image enhancement. Through precise trimming and rescaling, it amplifies the image's contrast. This heightened contrast acts like a spotlight, revealing previously obscured intricacies, textures, and subtleties. The image undergoes a transformation, with enhanced clarity and context for each highlighted element.

```
def trim_image_in_linear_stretch(image, q_sequence_of_percentiles=2
                                 , maxout=1, min_out=0):

  def image_ojbect_process(image_ojbect, maxout=maxout, minout=
                                   min_out):
    # Truncate the HSI Image. # 2% of pixels to be truncated from
                                     both ends
    trim_down = np.percentile(a=image_ojbect, q=
                                     q_sequence_of_percentiles)
    trim_up = np.percentile(a=image_ojbect, q=100 -
                                     q_sequence_of_percentiles)
    # q = between 0 and 100 inclusive. If q is a single percentile
                                     and axis=None,
    # then the result is a scalar. The return output is specified
                                     in array.

    # Rescale the pixel values. Just like Numpy Clip image function
                                     . In ranage of b/w 0 and 1.
    image_ojbect_new = (image_ojbect - trim_down) / ((trim_up -
                                     trim_down) / (maxout - minout
                                     ))
    # Formula: ((Old Image - Low Percentile) / ((Max Percentile -
                                     Low Percentile) / (Max Edge -
                                      Min Edge))
    image_ojbect_new[image_ojbect_new < minout] = minout # Replace
                                     0.
```

```
    image_ojbect_new[image_ojbect_new > maxout] = maxout # Replace
                                                1a.

    return np.float32(image_ojbect_new)

image = np.float32(image)
height, width, band = image.shape
new_image = np.zeros((height, width, band))

for b in range(band):
    new_image[:, :, b] = image_ojbect_process(image[:, :, b])
return new_image
```

Chapter 8

Iteration 3

Navigating the complex terrain of the Final Year Project-2 (FYP-2), the midpoint stands out as a crucial juncture. This initial phase marks the achievement of key milestones, navigating through the complexities of spectral intricacies, geometrical subtleties, high and low-frequency decomposition, meticulous quantitative assessments, and the coordination of train/test modules.

1. Revealing Spectral Bands Connections: At the core of our exploration lies the introduction of spectral bands correlation. It's not a mere concept but an extensive component crafted to unveil the concealed insights within hyperspectral image (HSI) data. Spectral bands correlation provides a nuanced perspective on the intricate interrelations between bands, offering a detailed understanding of spectral nuances. This module enriches our investigation by uncovering the elaborate connections that underscore the complexity of hyperspectral data.

2. Emphasizing Geometrical Characteristics: Geometrical characteristics, another fundamental aspect, come into play. The module dedicated to extracting geometrical features transcends being just a tool; it serves as a key to unlock the structured prior information embedded within HSI data. It systematically extracts spatial and spectral intricacies, molding them into a format that encapsulates the geometrical essence of the data. Through this approach, we gain a deeper understanding of the spatial and structural complexities inherent in the data.

3. Clarifying Frequency Components: This stage involves a comprehensive examination of high and low-frequency breakdown, offering a robust method for untangling intricate frequency patterns in HSI data. The dissection enhances our ability to address noise and signals with precision, laying the foundation for advanced denoising and analysis.

4. Precision in Quantitative Assessment: A crucial aspect of our exploration is the meticulous pursuit of quantitative evaluation. This goes beyond simple metrics, involving a detailed process to objectively assess outcomes from both simulated and real experiments. This evaluation yields vital insights, enabling informed, data-driven decisions and elevating the overall quality of results.

5. Striving for Benchmark Excellence: As we reach the midpoint, our focus turns to benchmarking. Through purposeful experiments, we compare our methodologies with state-of-the-art techniques, establishing a benchmark for assessing our approaches in the realm of HSI denoising. This process not only enriches our insights but also contextualizes our work within the broader landscape.

6. Matrix of Test Cases: The commitment to detail extends to meticulous testing of each component module. We methodically scrutinize every aspect, from the dataset's dimensions and structure to the type of tensors and instances. This thorough examination ensures the optimal functioning of each component, eliminating the possibility of unnoticed errors or inconsistencies.

As we reach the midpoint of the FYP-2 odyssey, this comprehensive exploration assures a deeper comprehension of hyperspectral data and HSI denoising. From spectral correlations to geometrical insights, high-frequency decomposition to quantitative assessments, benchmarking against premier methods, and thorough component testing, each aspect of the journey contributes intricacy and accuracy to our efforts. In this pursuit of excellence, each module, experiment, and evaluation enhances our comprehension and advances our approaches to unprecedented levels.

8.1 Decompose Frequency (Architecture) Network: Navigating the AAFEHDN Framework

The architecture of Decompose Frequency assumes a pivotal role in our quest to denoise hyperspectral datasets. It encapsulates the core of AAFEHDN (Adaptive Attention-Based Fully End-to-End Hyperspectral Denoising Network), our intricately designed method aimed at bringing clarity and precision to the intricate realm of hyperspectral data. In this segment of our journey, we present the six-block AAFEHDN, a complex mechanism that guides the conversion of Spatial-Spectral (SS) data, culminating in a clear and denoised hyperspectral image. Let's delve into the exploration of the AAFEHDN block, unveiling its internal mechanisms layer by layer. The figure-8.1 elucidates the network structure of the AAFEHDN framework.

- Initial Input: Our journey commences with the Initial Input, the fundamental raw material supporting our denoising initiative. This foundational input acts as the gateway to spatial attention modules, nurturing profound spatial sample information. The Updated Input, a product of this stage, accumulates and processes samples in each iteration, thus refining features of the Initial Input.

- Upgraded Input: The Upgraded Input seamlessly follows the trail of the Initial Input, inheriting the responsibility of data transformation. This component carries updated samples of the Initial Input for each iteration, continuing its journey. In this phase, the input is directed towards the ASC (Adaptive Spatial Correlation) module, diligently amalgamating the output of Spatial Attention with Initial Input features.

- ASC Module (Adaptive Spatial Correlation): The ASC module stands as a central element in the AAFEHDN block, where data evolution takes on novel dimensions. Input from the Upgraded Input undergoes processing, and its output is transmitted to both Spectral Attention and the PSCA (Pixel-Wise Spatial Channel Attention) Module, all within each iteration. The ASC module serves as the core where spatial correlations are meticulously crafted, laying the groundwork for subsequent steps.

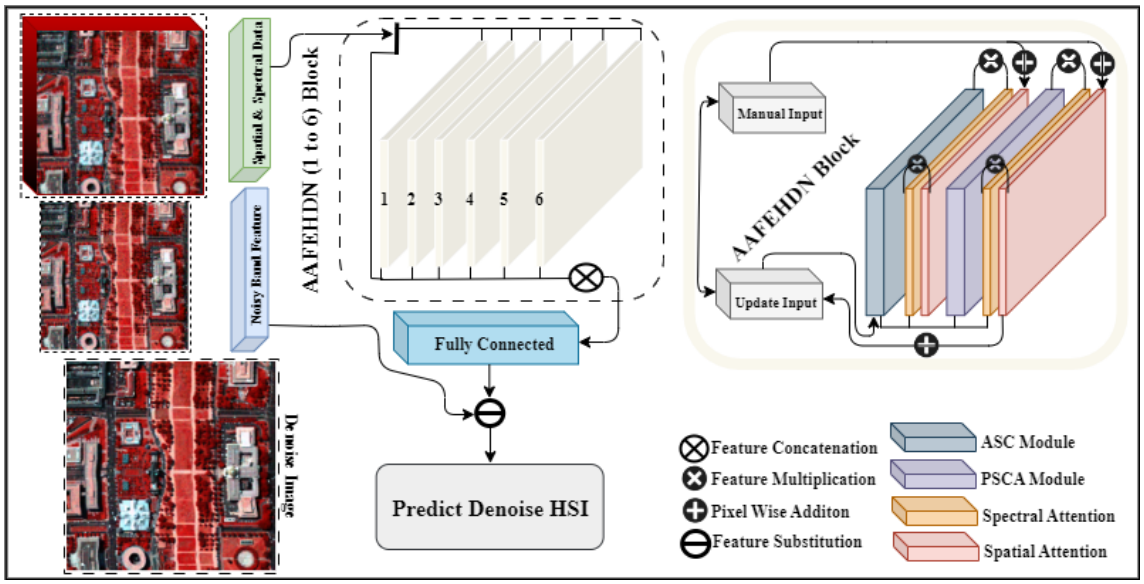


Figure 8.1: Denoising HSI Dataset: AAFEHDN's Decompose Frequency Architecture Unraveled, Featuring Spatial-Spectral Blocks, ASC, PSCA, Spectral and Spatial Attention Modules.

- Pixel-Wise Spatial Channel Attention (PSCA) Module: The PSCA module, intricately linked with the ASC module, takes input from both ASC and Spectral Modules. In this stage, data undergoes detailed processing, and the output becomes intertwined with the Spectral Attention element. This step is crucial in the broader channel attention strategy, further refining hyperspectral data.

- Spectral Attention: An integral element in the AAFEHDN framework, the Spectral Attention component influences the data. It receives input from both ASC and PSCA Modules, enriching it with intricate spectral details. The result is spatial attention enhanced with feature multiplication, a vital aspect shaping the denoising process.

- Spatial Attention: In the final phase of the AAFEHDN block, the Spatial Attention component refines hyperspectral data. It takes input from multiple features, adding spatial finesse by incorporating samples from Spectral Attention and Manual Input. The output represents the culmination of the denoising process, delivering a pristine hyperspectral image. The pixel-wise addition of Manual Input features in each iteration contributes to the precision of this final step.

The Decompose Frequency structure embedded in the AAFEHDN framework transcends being a mere network; it's a sophisticated interplay of data transformations, spectral and spatial focus, and adaptive correlations. It unfolds as a meticulous journey, refining input at every stage to yield a denoised hyperspectral image. In this elaborate choreography of data, each step assumes significance, collectively enhancing the accuracy and lucidity of the ultimate output. As our exploration persists, we welcome the intricacies of hyperspectral data, aiming to unveil its profound intricacies, layer by layer.

8.2 Geometrical Characteristics: Feature Extraction

In our quest for superior HSI denoising, attention shifts to the Geometrical Characteristics component, a crucial element in the complex hyperspectral data analysis framework. This module, integral to the broader structure, holds a key role in extracting spatial-spectral features that form the foundation of spectral bands correlation. To comprehend its intricacy, let's delve into its layers, each marked by precision and purpose. Figure-8.2 delineates the design of the geometrical features extraction module.

1. Module for Extracting Spatial Features: At the core of Geometrical Characteristics lies the Spatial Features Extraction Module. This unit is dedicated to capturing the subtleties of spatial data, delving into the spatial arrangement, patterns, and relationships that characterize the hyperspectral landscape. It embarks on a journey into the complexities of how objects and elements are positioned within the spatial dimension. Through thorough analysis, it unravels the spatial intricacies that often hold the key to understanding the intricate interplay within hyperspectral data.

2. Module for Extracting Multiscale Separable Spectral Features: Spectral features, often present in the frequency domain, play a pivotal role in hyperspectral data analysis. The Multiscale Separable Spectral Features Extraction Module is a dynamic component that unveils the mysteries of spectral data across different scales. The term "separable" signifies a unique capability — these features can be independently analyzed at various scales. This autonomy allows for a detailed examination

of spectral intricacies, elucidating the subtleties of hyperspectral data across different frequency scales.

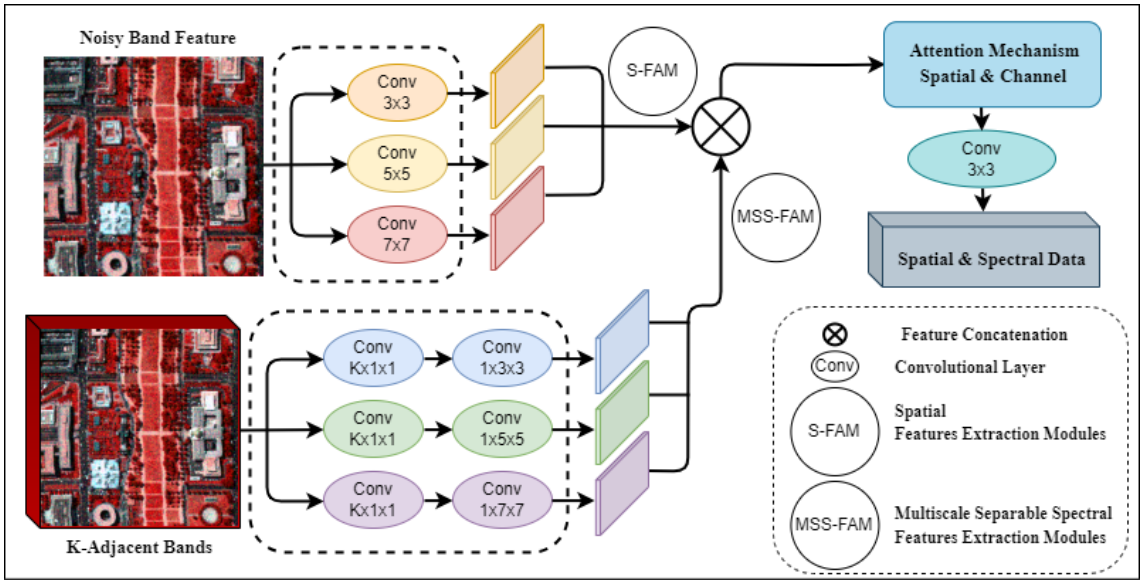


Figure 8.2: Unveiling Geometrical Characteristics: Extracting Spatial-Spectral Features for Enhanced Understanding of Spectral Bands Correlation in Hyperspectral Data Analysis.

8.3 Spatial and Spectral: Attention Module

In the complex realm of hyperspectral data examination, the Spatial and Spectral Features Decompose Frequency Attention Module emerges as a paragon of meticulous intricacy. This module, intricately crafted to unveil the profound intricacies within the spatial and spectral facets of hyperspectral images, initiates a nuanced and sophisticated data decomposition journey. Let's explore its internal mechanisms to grasp its intricacy. The figure-8.3 depicts the configuration of the attention module for Spatial and Spectral features.

1. Spatial Focus Unit: An integral element of the Frequency Decomposition Focus Unit, this module is committed to amplifying spatial attributes in hyperspectral images. Spatial attributes encompass the spatial organization of elements in the data, critical for precise analysis. To realize this, the input data undergoes several transformations, involving Max and Average pooling layers to discern spatial subtleties.

Additionally, a Convolution layer, activated by sigmoid, concentrates on spatial features in individual spectral bands. This intricate procedure culminates in enhanced spatial noise reduction, a crucial refinement for hyperspectral data.

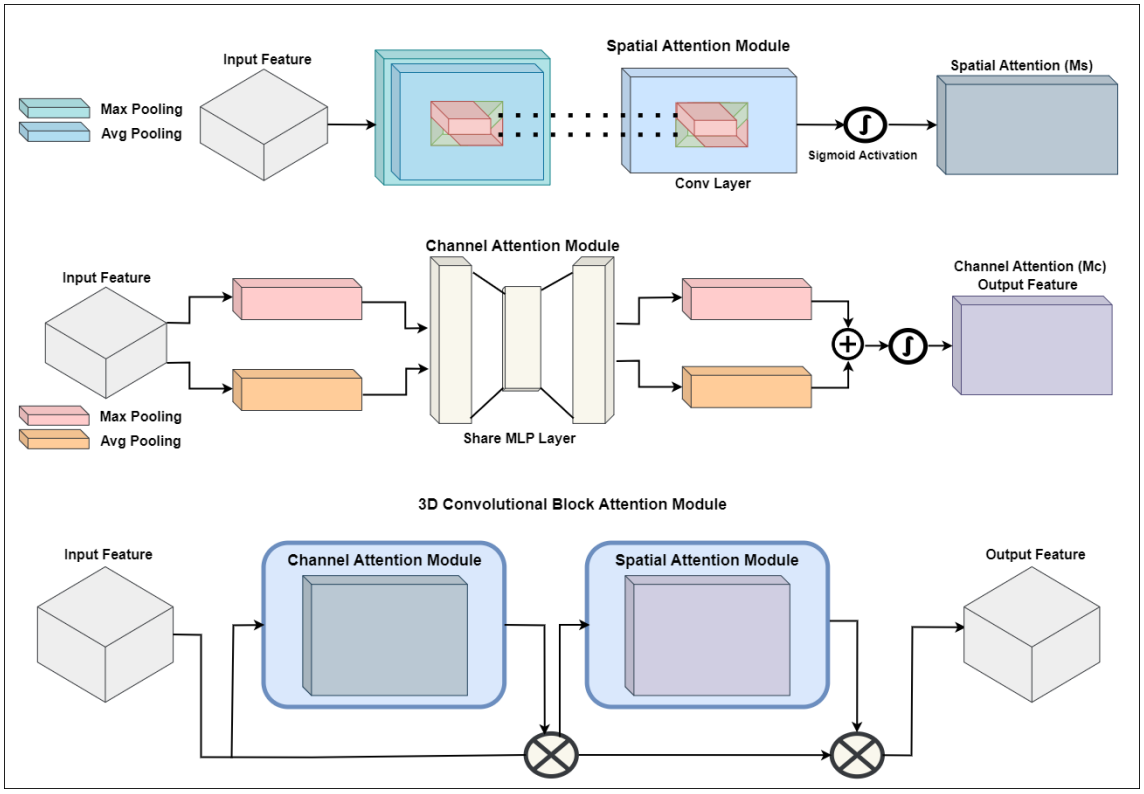


Figure 8.3: Decompose Frequency Attention Module: A Symphony of Spatial and Spectral Precision

2. Spectral (Channel) Attention Module: Complementing the Spatial Attention Module, the Spectral Attention Module focuses on enhancing features in the spectral or channel domain within hyperspectral images. These features play a critical role in understanding the quality and uniqueness of spectral band correlations. Similar to its spatial counterpart, the Spectral Attention Module utilizes Max and Average pooling layers to discern spectral nuances, which are then merged to preserve the integrity of spectral features. To explore channel correlation complexities further, multi-layer perceptron layers are employed. The resulting output undergoes Max and Average pooling layers again before sigmoid activation, yielding a set of channel samples that encapsulate the spectral intricacies within the hyperspectral data.

8.4 ASC and PSCA: Network Modules

The Attentive Skip Connection (ASC) utilizes high-frequency elements for detailed intricacies, and low-frequency components for broader patterns, enhancing contextual understanding and nuanced predictions. The Progressive Spectral Channel Attention (PSCA) dynamically focuses on specific spectral channels, facilitating adaptation to diverse frequency patterns for optimal task performance. The diagram in Figure 8.4 elucidates the architecture of the ASC and PSCA network. **Meticulous Integration of High and Low**

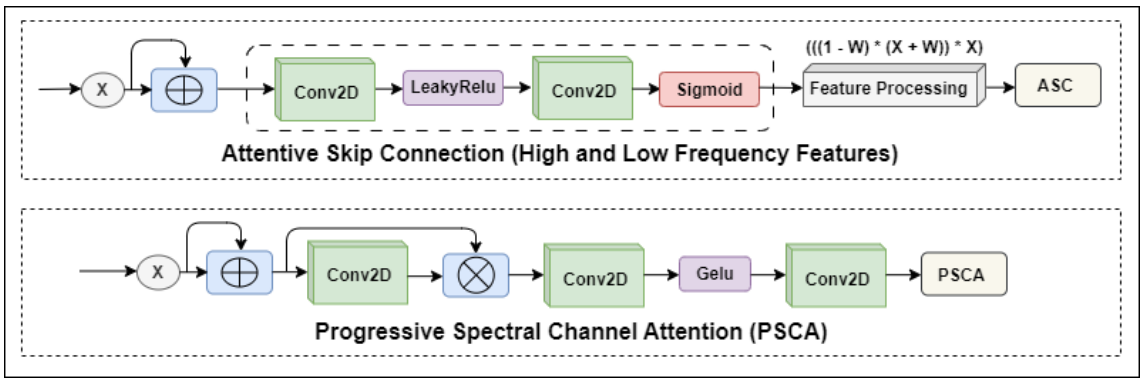


Figure 8.4: Unifying Detail and Context: Attentive Fusion of High and Low Frequencies, Empowered by Dynamic Focus in Progressive Spectral Channel Attention.

Frequencies: Navigating Through Delicate Details and Comprehensive Patterns: In the domain of hyperspectral data analysis, the Thoughtful Link Connection emerges as a symbol of refinement. It seamlessly blends the intricate interplay of high and low-frequency characteristics, each capturing a distinct aspect of the data. High-frequency attributes master fine details, while their low-frequency counterparts outline the canvas with broader, comprehensive patterns. By harmonizing these varied elements, the model embarks on a journey of predictions tuned to context and well-informed decision-making. Let's explore further into this intricately woven narrative of integration.

- High-Frequency Characteristics: These attributes act as the maestros of intricacy, precisely capturing the finer subtleties inherent in the data. They represent the minute details, delineating the most delicate patterns.

- Low-Frequency Characteristics: Conversely, low-frequency characteristics act as

the designers of overarching patterns. They involve the broader aspects of information, shaping the foundational structure of the data's portrayal.

The Attentive Skip Connection adeptly combines these components, granting the model a holistic perspective on the intricacies of the data. This integration enables the model to formulate predictions and decisions with a deep comprehension of both the details and the broader context, ensuring thorough consideration of all aspects of the data.

Progressive Evolution of Spectral Focus (PSCA): A Shifting Embrace of Attention: In the realm of hyperspectral data scrutiny, the Progressive Spectral Channel Attention (PSCA) module stands out as a vibrant influence. It embodies a philosophy of evolving attention, where the model dynamically adjusts its focus on distinct spectral channels in real-time. This flexibility mirrors a chameleon's adaptability, conforming to the varied frequency patterns within the input data, customized for the task at hand.

- Adaptive Attention Refinement: PSCA utilizes a flexible attention mechanism, dynamically adjusting its focus based on the inherent frequency patterns in the input data. It doesn't adhere to a rigid lens but fine-tunes attention to match unique patterns.

- Dynamic Evolution in Real Time: This ability for real-time adaptation ensures the model remains dynamic, constantly refining its comprehension of the data. This mechanism mirrors the fluid nature of real-world scenarios encountered in hyperspectral data analysis.

The Progressive Spectral Channel Attention (PSCA) module exemplifies the intricacy in hyperspectral data analysis, ensuring the model adapts to the dynamic frequency variations in the input data. It serves as a responsive lens, enabling context-aware decisions and accurate predictions.

8.5 Quantitative Evaluation: Comparison Models

In the dynamic realm of image restoration, standout structures like MemNet [45] and DeNet [46] have claimed prominence. These networks are not just creations but elaborate arrangements of memory and spatial correlation. Join us as we delve into their mechanics and unravel the intricacies that characterize them. The comparison experiments have conducted in this table [8.1, 8.2 and 8.3] and figure [8.5 and 8.6].

Table 8.1: The Sigma-50 benchmark experiments of MemNet and DeNet with the proposed network have been illustrated in the matrix index of PSNR, SSIM, and SAM.

Model (Sigma-50)	PSNR	SSIM	SAM	Train Time Average %	Test Time Average %
AAFEHDN	28.92651.5490	0.95550.0279	0.10890.0285	3.6936	11.5924
MemNet	19.39026.4411	0.77370.1128	0.20280.1560	13.3993	14.9050
DeNet	22.572511.6716	0.76850.0919	0.31220.1487	10.3661	11.3272

Table 8.2: The Gaussian benchmark experiments of MemNet and DeNet with the proposed network have been illustrated in the matrix index of PSNR, SSIM, and SAM.

Model (Sigma-50)	PSNR	SSIM	SAM	Train Time Average %	Test Time Average %
AAFEHDN	34.03781.7848	0.98240.0143	0.07480.0226	3.8477	11.6141
MemNet	22.351515.4515	0.86960.1196	0.15510.1639	13.3232	14.7361
DeNet	28.925712.0762	0.92170.1058	0.17240.2174	10.2759	10.6780

Table 8.3: The Random benchmark experiments of MemNet and DeNet with the proposed network have been illustrated in the matrix index of PSNR, SSIM, and SAM.

Model (Sigma-50)	PSNR	SSIM	SAM	Train Time Average %	Test Time Average %
AAFEHDN	34.56451.2742	0.99410.0029	0.06490.0097	3.9187	11.7451
MemNet	22.51636.1887	0.90250.1246	0.15980.1719	13.4377	14.8680
DeNet	16.53978.3215	0.91220.0922	0.19180.1646	10.3118	11.0173

MemNet: The Symphony of Adaptive Memory: At the core of MemNet resides a memory block orchestrating a sophisticated interplay between a recursive unit and a gate unit. This architectural masterpiece goes beyond being a mere block; it exemplifies adaptive learning and the skill of retaining enduring memory.

- Recursive Module: This module acts as the creator, crafting varied representations of the present state with a range of receptive fields. It serves as the director, adding depth and subtlety to the network's comprehension.

- Gate Module: Serving as a custodian, this module determines which memories from the past to preserve and how much of the current state to store. It acts as the guardian of the network's memory, ensuring only the most valuable fragments from the past are retained.

The utilization of MemNet in three distinct image enhancement tasks, namely image denoising, super-resolution, and JPEG deblocking, showcases the network's adaptability and its proficiency in effectively leveraging memory for improved restoration outcomes.

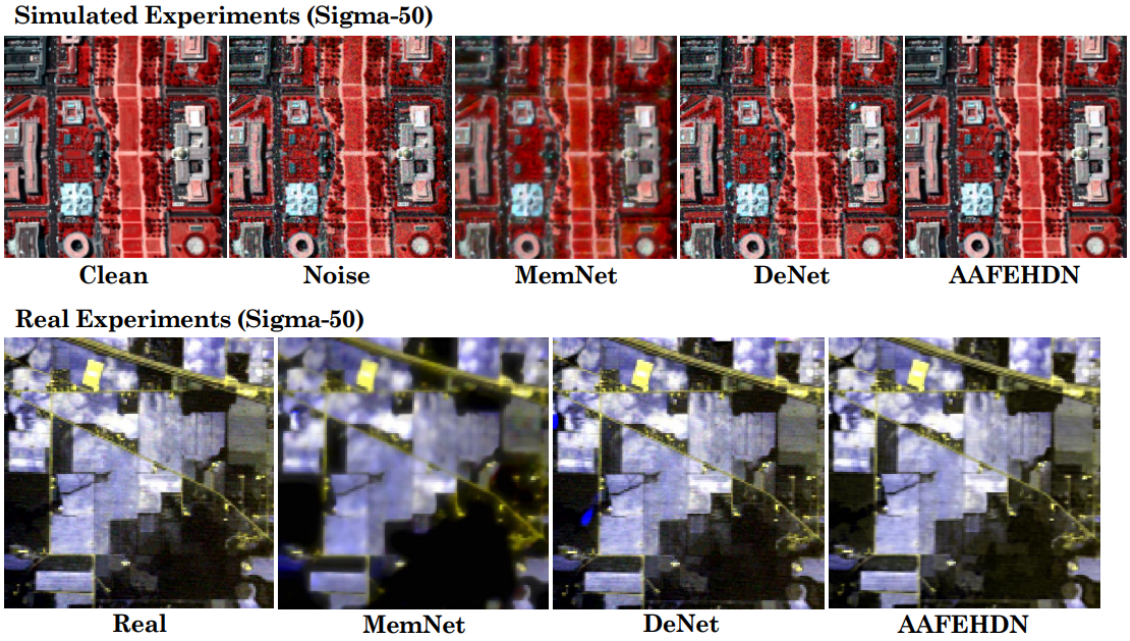


Figure 8.5: Unifying predict benchmark experiments of MemNet and DeNet with the proposed network based on Alpha Sigma 50 noise level in both Simulated and Real.

DeNet: The Architect of Spatial and Spectral Symphony: On the flip side, DeNet is a network that unveils the intricacies of spatial details and spectral correlation. Its architectural finesse is evident in its capability to extract these crucial elements using acquired filters and numerous filter channels.

- Maintaining Spectral-Spatial Structures: DeNet serves as a protector of the intricate interplay between spectral and spatial elements in images, guaranteeing the preservation of these essential structures for more accurate restorations.

- Versatility and Swiftness: DeNet, a highly adaptable performer, excels in managing diverse noise types and proves its versatility in processing both single and multiple

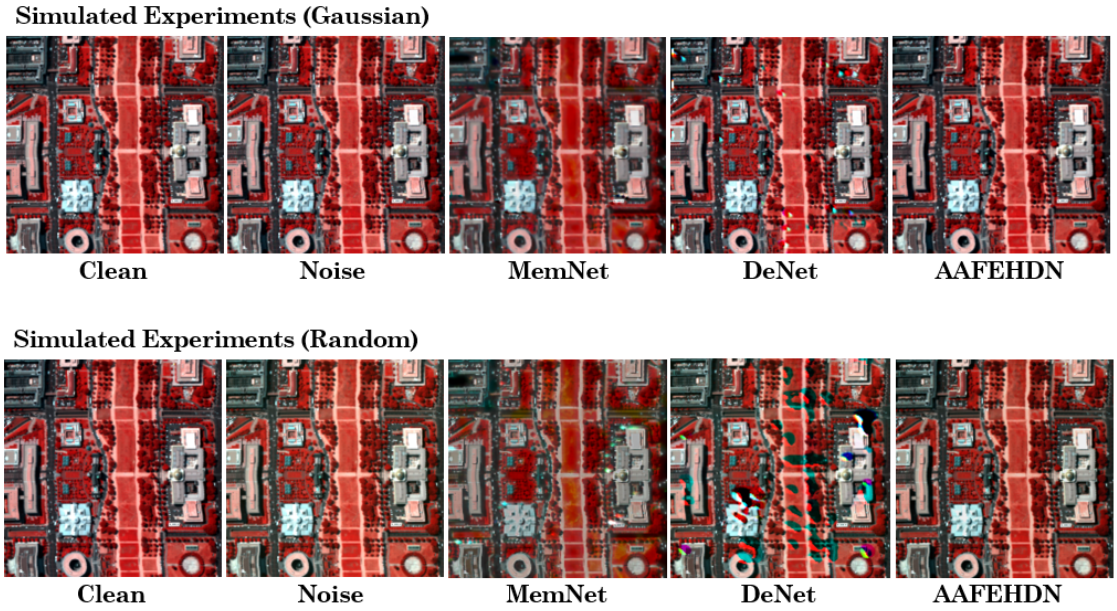


Figure 8.6: Unifying predict benchmark experiments of MemNet and DeNet with the proposed network based on Gaussian and Random noise level in both Simulated and Real.

images across various real-world scenarios. Additionally, its rapid performance during testing makes it invaluable for practical applications.

MemNet and DeNet transcend mere networks; they epitomize a harmony of technology and artistic finesse. Within the domain of image restoration, they encapsulate the intricacies of memory, spatial correlation, and adaptability. Through their applications and architectural prowess, they elevate the landscape of image restoration, expanding the horizons of digital visual possibilities.

8.6 Quantitative Evaluation: Metrics Measurement

In the constantly evolving realm of image processing and assessment, a triumvirate of metrics serves as custodians of image quality and similarity. Each metric crafts a distinct narrative about the visual data it assesses, offering a holistic view of the components that define image excellence. Let's explore the nuances of these metrics and decipher the narratives they unfold. The comparative experiments depicted in figures [8.7, 8.8, and 8.9] present graphical representations of metric measurements.

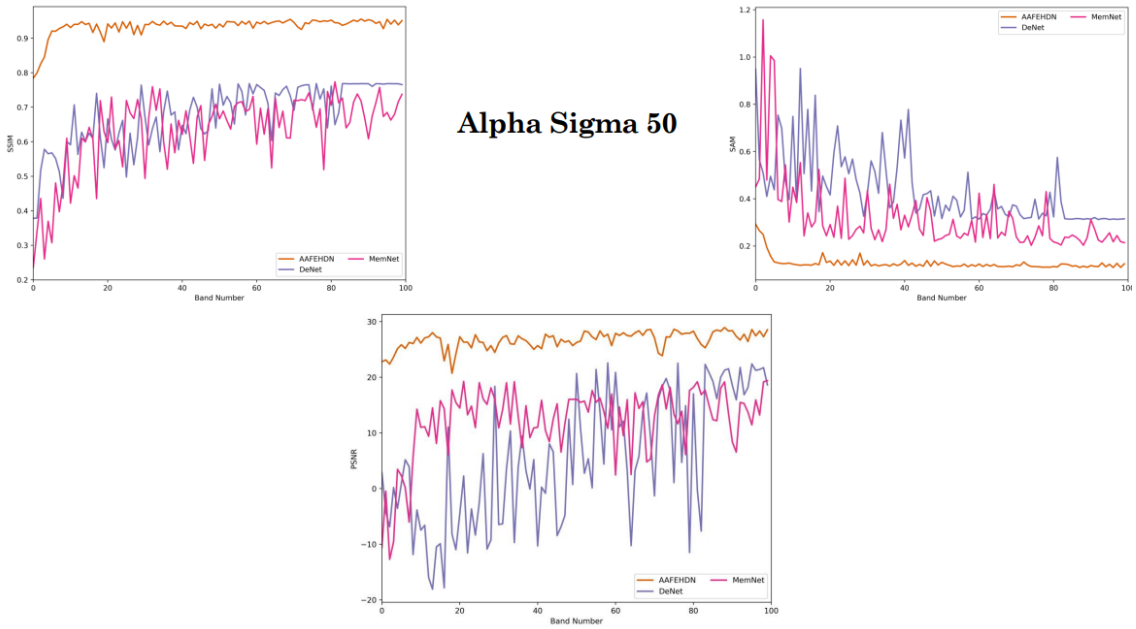


Figure 8.7: Unifying predict graphs of metrics measurement in benchmark experiments of MemNet and DeNet with the proposed network based on Alpha Sigma 50 noise level in both Simulated and Real.

Peak Signal-to-Noise Ratio (PSNR): PSNR serves as the virtuoso in evaluating image quality, expressing itself through ratios. This metric assesses image quality by comparing it to a reference image, focusing on the interplay between peak signal power and noise power. Essentially, it gauges how distinctly the desired signal stands out against background noise, akin to a harmonious symphony. Elevated PSNR values signify superior image quality, akin to a melodious chorus where the desired signal prevails over noise.

Spectral Angle Mapper (SAM): SAM serves as the expert in contrasting the spectral fingerprints of image pixels. Its canvas is the angular resemblance among pixel spectra, an exclusive metric embodying the craft of similarity evaluation. SAM's proficiency proves invaluable in endeavors like image classification and change detection, where discerning the nuanced distinctions in spectral signatures holds utmost importance. It functions as the tool that adeptly captures the angular intricacies in spectral data, crafting a vibrant portrayal of pixel relationships.

Structural Similarity Index Measure (SSIM): SSIM stands as a discerning judge of structural finesse, assessing luminance, contrast, and structure like an art critic scrutinizing a masterpiece. SSIM values, resembling brushstrokes, convey the similarity in

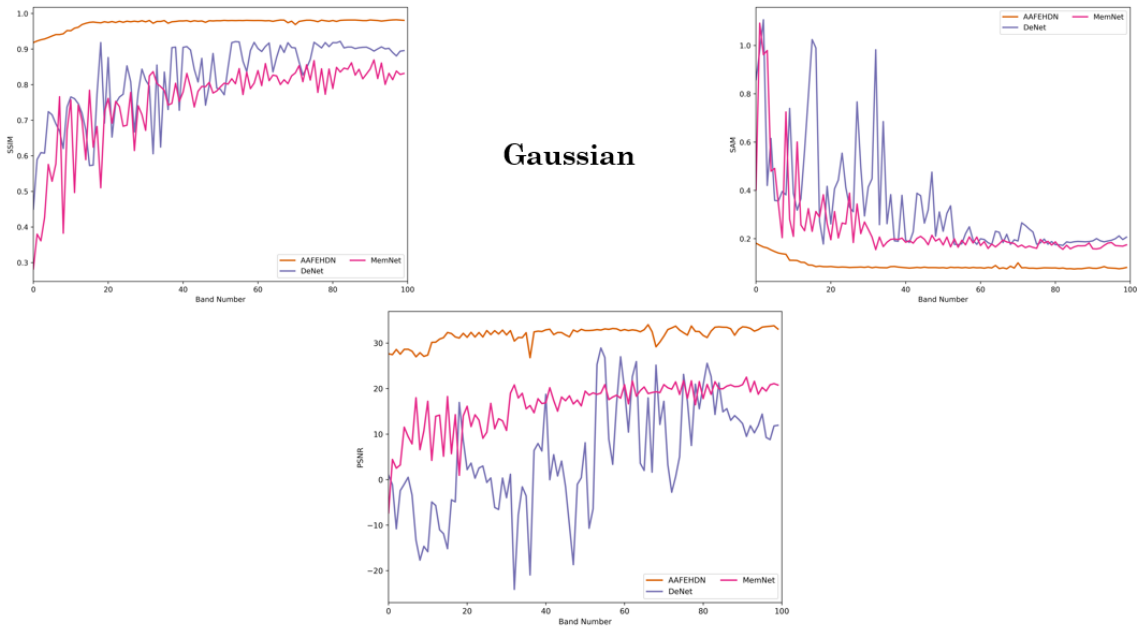


Figure 8.8: Unifying predict graphs of metrics measurement in benchmark experiments of MemNet and DeNet with the proposed network based on Gaussian noise level in both Simulated and Real.

structural and visual aspects between two images. This metric scrutinizes images, considering luminance, contrast, and structure, showcasing their structural harmony through higher SSIM values.

In the realm of image assessment, PSNR, SAM, and SSIM serve as the storytellers elucidating the narrative of image excellence, spectral congruence, and structural finesse. Collectively, they craft a vibrant portrayal of visual quality, fostering comprehension, admiration, and refinement of the visual landscape.

8.7 Test Cases: AAFEHDN Network

In the domain of detailed examinations, we showcase our commitment to accuracy by meticulously examining each module. Our methodical review involves a thorough assessment, encompassing all aspects, from the dataset's size and configuration to the tensor type and instantiation. This exhaustive evaluation eliminates any room for unnoticed errors or inconsistencies, ensuring that each component functions exactly as intended.

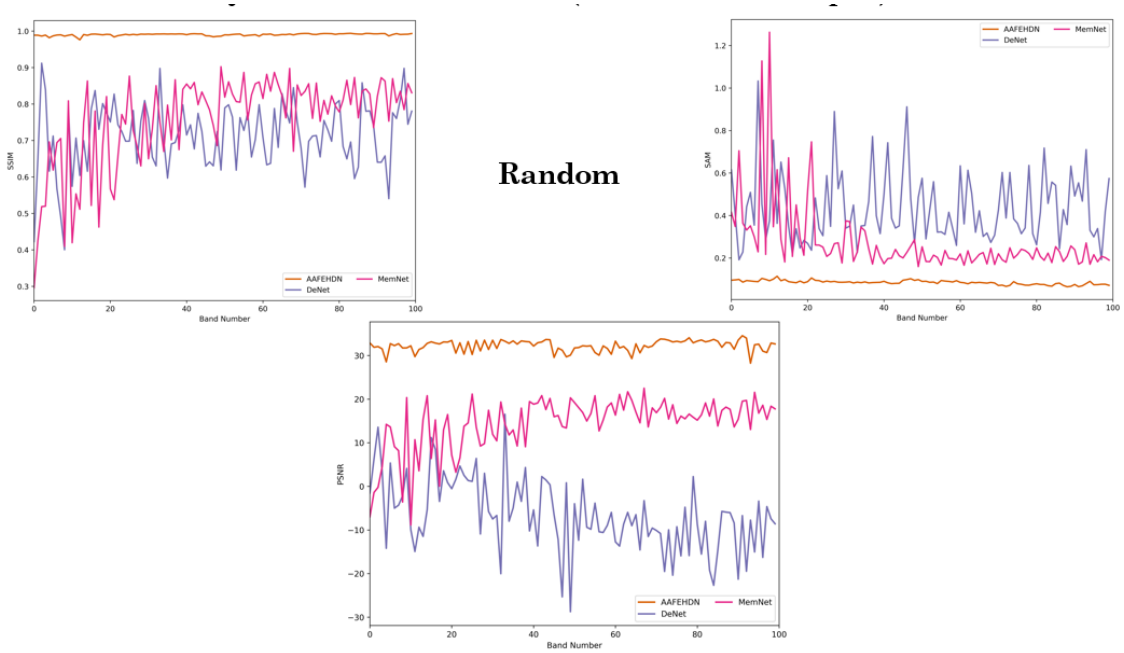


Figure 8.9: Unifying predict graphs of metrics measurement in benchmark experiments of MemNet and DeNet with the proposed network based on Random noise level in both Simulated and Real.

8.7.1 Spatial and Spectral: Attention Modules

Modules for Spatial and Spectral Attention: Configuring parameters for spatial and spectral attention modules, transmitting information between channel and spatial modules for feature attention. Test cases cover various data types, output shapes, and return types.

```
def spatial_channel_attention_testcase():
  data = torch.randn(1, 80, 200, 200)

  output_ca = ChannelAttention(in_planes=data.shape[1])
  output_sa = SpatialAttention(kernel_size=7)

  output = output_ca(data) * data
  output = output_sa(output) * output + data

  assert isinstance(output, torch.Tensor), "Spatial and Channel
                                    Attention Test Failed! -> List"
  print("Spatial and Channel Attention Test Passed! -> List")

  assert (len(output) == 1) and (output.shape == torch.Size([1, 80,
```

```
                                    200, 200])), "Spatial and
                                    Channel Attention Test Failed!
                                    -> Length (1) and Shape (1,80,
                                    200,200)"
  print("Spatial and Channel Attention Test Passed! -> Length (1)
                                    and Shape (1,80,200,200)")
  for value in output:
    assert isinstance(value, torch.FloatTensor), "Spatial and
                                      Channel Attention Test Failed
                                      ! -> FloatTensor"
  print("Spatial and Channel Attention Test Passed! -> FloatTensor"
                                    )

spatial_channel_attention_testcase()

# Output:
Spatial and Channel Attention Test Passed! -> List
Spatial and Channel Attention Test Passed! -> Length (1) and Shape
                                  (1,80,200,200)
Spatial and Channel Attention Test Passed! -> FloatTensor
```

8.7.2 Features Extraction Modules

Spatial and Multiscale Separable Spectral Features Extraction Modules: Configuring parameters for the extraction modules related to spatial and spectral characteristics, involving channel and spatial feature information. Testing various data types, output shapes, and return types.

```
def spatial_channel_attention_testcase():
  spatial_data = torch.randn(1, 1, 200, 200)
  spectral_data = torch.randn(1, 1, 24, 200, 200)

  output_geo = Geometrical_Characteristics(input_Channel=1,
                                    feature_Channel=20, K=24)
  output = output_geo(spatial_data, spectral_data)
```

```
  assert isinstance(output, torch.Tensor), "Geometrical
                                    Characteristics Test Failed! ->
                                     List"
  print("Geometrical Characteristics Test Passed! -> List")

  assert (len(output) == 1) and (output.shape == torch.Size([1, 120
                                    , 200, 200])), "Geometrical
                                    Characteristics Test Failed! ->
                                     Length (1) and Shape (1,120,
                                    200,200)"
  print("Geometrical Characteristics Test Passed! -> Length (1) and
                                     Shape (1,120,200,200)")

  for value in output:
    assert isinstance(value, torch.FloatTensor), "Geometrical
                                      Characteristics Test Failed!
                                      -> FloatTensor"
  print("Geometrical Characteristics Test Passed! -> FloatTensor")

spatial_channel_attention_testcase()

#Output:
Geometrical Characteristics Test Passed! -> List
Geometrical Characteristics Test Passed! -> Length (1) and Shape (1
                                    ,120,200,200)
Geometrical Characteristics Test Passed! -> FloatTensor
```

8.7.3 Denoise Model Blocks

Breakdown of Frequency (High and Low): Configuring ASC and PSCA module parameters and transmitting information from ASC to PSCA modules for feature decomposition. Testing various data types, output shapes, and return types.

```
def asc_psca_decompose_frequency_testcase():
  spatial_spectral_data = torch.randn(1, 80, 200, 200)
```

```
  channel_get = spatial_spectral_data.shape[1]
  output_asc = ASC(channel=channel_get)
  output_psca = PSCA(channel=channel_get, channel_half=channel_get
                                    //2)

  output = output_asc(spatial_spectral_data, spatial_spectral_data)
  output = output_psca(output)

  assert isinstance(output, torch.Tensor), "Decompose Frequency (
                                    High and Low) Test Failed! ->
                                    List"
  print("Decompose Frequency (High and Low) Test Passed! -> List")

  assert (len(output) == 1) and (output.shape == torch.Size([1, 80,
                                     200, 200])), "Decompose
                                    Frequency (High and Low) Test
                                    Failed! -> Length (1) and Shape
                                     (1,80,200,200)"
  print("Decompose Frequency (High and Low) Test Passed! -> Length
                                    (1) and Shape (1,80,200,200)")

  for value in output:
    assert isinstance(value, torch.FloatTensor), "Decompose
                                      Frequency (High and Low) Test
                                      Failed! -> FloatTensor"
  print("Decompose Frequency (High and Low) Test Passed! ->
                                    FloatTensor")

asc_psca_decompose_frequency_testcase()

#Output:
Decompose Frequency (High and Low) Test Passed! -> List
Decompose Frequency (High and Low) Test Passed! -> Length (1) and
                                  Shape (1,80,200,200)
Decompose Frequency (High and Low) Test Passed! -> FloatTensor
```

8.7.4 Metric Index

Evaluation Metric: Configuring parameters for metrics such as PSNR, SSIM, and SAM. Utilizing original and denoised data for quantitative assessment. Testing various data types, lengths, shapes, and return types.

```
def test_sam():
  x_true = np.random.random((5, 5, 3))
  x_pred = np.random.random((5, 5, 3))
  sam_deg = sam(x_true, x_pred)
  assert isinstance(sam_deg, float), "SAM Matrix Test Failed! ->
                                      Float"
  print("SAM Matrix Test Passed! -> Float")
test_sam()

data_clean = np.random.random((200, 200, 192))
test_out = np.random.random((200, 200, 192))
results = quantitative_assess(data_clean, test_out)
assert isinstance(results, list), "PSNR and SSIM Matrix Test Failed
                                   ! -> List"
print("PSNR and SSIM Matrix Test Passed! -> List")
assert len(results) == 3, "PSNR and SSIM Matrix Test Failed! ->
                                  Length (3)"
print("PSNR and SSIM Matrix Test Passed! -> Length (3)")
for value in results:
  assert isinstance(value, float), "PSNR and SSIM Matrix Test
                                    Failed! -> Float" + exit()
print("PSNR and SSIM Matrix Test Passed! -> Float")

#Output:
SAM Matrix Test Passed! -> Float
PSNR and SSIM Matrix Test Passed! -> List
PSNR and SSIM Matrix Test Passed! -> Length (3)
PSNR and SSIM Matrix Test Passed! -> Float
```

8.7.5 Proposed Model

Configuring AAFEHDN network parameters involves transmitting spatial and spectral features of HSI images for noise decomposition and denoising prediction. Testing encompasses various data types, output shapes, and return types.

```
def aafehdn_block_network_testcase():
  spatial = torch.randn(1,1,200,200)
  spectral = torch.randn(1,1,24,200,200)

  output_aaffehdn = AAFEHDN(input_Channel=1, block_num=3, K=24,
                                    feature_Channel=20,
                                    layer_output=80)
  output = output_aaffehdn(spatial, spectral)

  assert isinstance(output, torch.Tensor), "Denoising (AAFEHDN)
                                    Test Failed! -> List"
  print("Denoising (AAFEHDN) Test Passed! -> List")

  assert (len(output) == 1) and ((output.shape == torch.Size([1, 1,
                                     200, 200])) and (output.shape
                                    == spatial.shape)), "Denoising
                                    (AAFEHDN) Test Failed! ->
                                    Length (1) and Shape (1,1,200,
                                    200)"
  print("Denoising (AAFEHDN) Test Passed! -> Length (1) and Shape (
                                    1,1,200,200)")

  for value in output:
    assert isinstance(value, torch.FloatTensor), "Denoising (
                                      AAFEHDN) Test Failed! ->
                                      FloatTensor"
  print("Denoising (AAFEHDN) Test Passed! -> FloatTensor")

aafehdn_block_network_testcase()

#Output:
```

```
Denoising (AAFEHDN) Test Passed! -> List
Denoising (AAFEHDN) Test Passed! -> Length (1) and Shape (1,1,200,
                                    200)
Denoising (AAFEHDN) Test Passed! -> FloatTensor
```

Chapter 9

Iteration 4

In the realm of Final Year Project 2, we compile an all-encompassing report encapsulating our discoveries. We undertake simulated experiments, subjecting our model to varied noise, including Gaussian and Random noise. Concurrently, we benchmark it against established state-of-the-art models. Our primary focus revolves around implementing and scrutinizing the AAFES-HDN network tailored for Hyperspectral Image denoising.

Within this phase, we intricately dissect the AAFES-HDN network's code structure, providing a detailed perspective on its components. These codes, extensively elucidated in the FYP-2 Midterm report, stand as evidence of our commitment and profound comprehension of the network's complexities.

9.1 Install the Packages

The configuration of necessary libraries to run the project code.

```
!pip install libtiff -qU

import os, glob, datetime, time, re
import shutil, random, math
from skimage.metrics import peak_signal_noise_ratio,
                                structural_similarity
from prettytable import PrettyTable
```

```
from matplotlib import pyplot as plt

import scipy.io
import numpy as np
import seaborn as sns
import plotly.express as px
import pandas as pd
import tifffile
import scipy.ndimage

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import MultiStepLR
import torch.nn.init as init
import torch.nn.functional as F

%matplotlib inline
```

9.2 Seed Randomness

To ensure uniformity and replicability in our Python-based neural network models, we implement a robust control system to manage randomness. This mechanism oversees random seed values in crucial components like the operating system, the native Python 'Random' module, the widely-used 'Numpy' library, and the 'Pytorch' framework API. By meticulously handling these seeds, we guarantee consistent outcomes in our neural network experiments, facilitating the monitoring and validation of diverse parameters and configurations in our models.

```
def SEED():
  seed = 123789 # Internation fixed number of seed...
  torch.manual_seed(seed)
  torch.cuda.manual_seed(seed)
  torch.cuda.manual_seed_all(seed)  # if you are using multi-GPU.
```

```
  np.random.seed(seed)  # Numpy module.
  random.seed(seed)  # Python random module.
  torch.manual_seed(seed)
  torch.backends.cudnn.deterministic = True
  torch.backends.cudnn.benchmark = False
  os.environ['PYTHONHASHSEED'] = str(seed)
```

9.3 Croped Specific Shape (Data): Train and Test

```
# WD Mall Dataset Google Drive Link. https://drive.google.com/file/
                                    d/
                                    1aIB5hYF1r_PA5DgJu5IPu2rRpxIO36TV
                                    /view?usp=sharing

# Download the dataset in your local/drive storage. The mount the
                                    Drive or setup the path of
                                    dataset folder.

dict_dc = r'/content/drive/MyDrive/Final-Year-Project (FYP)/IP-PU-
                                    DCMall (Datasets)/dc.tif'
def trim_image_in_linear_stretch(image, q_sequence_of_percentiles=2
                                    , maxout=1, min_out=0):

  def image_ojbect_process(image_ojbect, maxout=maxout, minout=
                                      min_out):
    # Truncate the HSI Image. # 2% of pixels to be truncated from
                                      both ends
    trim_down = np.percentile(a=image_ojbect, q=
                                      q_sequence_of_percentiles)
    trim_up = np.percentile(a=image_ojbect, q=100 -
                                      q_sequence_of_percentiles)
    # q = between 0 and 100 inclusive. If q is a single percentile
                                      and axis=None,
    # then the result is a scalar. The return output is specified
                                      in array.
```

```
    # Rescale the pixel values. Just like Numpy Clip image function
                                        . In ranage of b/w 0 and 1.
    image_ojbect_new = (image_ojbect - trim_down) / ((trim_up -
                                        trim_down) / (maxout - minout
                                        ))
    # Formula: ((Old Image - Low Percentile) / ((Max Percentile -
                                        Low Percentile) / (Max Edge -
                                         Min Edge))
    image_ojbect_new[image_ojbect_new < minout] = minout # Replace
                                        0.
    image_ojbect_new[image_ojbect_new > maxout] = maxout # Replace
                                        1a.

    return np.float32(image_ojbect_new)

  image = np.float32(image)
  height, width, band = image.shape
  new_image = np.zeros((height, width, band))

  for b in range(band):
    new_image[:, :, b] = image_ojbect_process(image[:, :, b])
  return new_image


#====> Read and Trim the dataset.
data_all = tifffile.imread(dict_dc)
data_all = data_all.transpose(1, 2, 0)
data_all = data_all.astype(np.float32)
data_all = trim_image_in_linear_stretch(data_all) # used to convert
                                    colorwise image... not to be
                                    black image.
data_all.shape

#====> Plot the figure of dataset.
fig = plt.figure(frameon=True, figsize = (8, 8))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
```

```
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((data_all[:,:,56],data_all[:,:,26],data_all[:,:
                                    ,16]),2))
plt.savefig("Washington DC Mall University_600-800_50-
                                    250_191_Clean_[56:26:16].png")
plt.show()
```

9.3.1 Testing Dataset

Height: 600 to 800 scale. Width: 50 to 250 sacle. Height and Width: (200, 200)

```
data_test = data_all[600:800, 50:250, :].astype(np.float32)
data_test.shape

#====> Plot the Test figure of dataset.
fig = plt.figure(frameon=False, figsize = (3, 3))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((data_test[:,:,56],data_test[:,:,26],data_test[
                                    :,:,16]),2))
plt.savefig("Washington DC Mall University_600-800_50-
                                    250_191_Clean_[56:26:16].png")
plt.show()
```

9.3.2 Training Dataset

Height and Width: 0 to 600 and 800 to 1280 scale, with total width. Concatenation: (0,600) with (800,1280) sacle. Height and Width: (1080, 307).

```
data_train1 = data_all[0:600, :, :]
data_train2 = data_all[800:1280, :, :]
data_train_DC = np.concatenate((data_train1, data_train2), axis=0)
data_train_DC.shape

#====> Plot the Train figure of dataset.
fig = plt.figure(frameon=False, figsize = (8, 8))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((data_train_DC[:,:,56],data_train_DC[:,:,26],
                                    data_train_DC[:,:,16]),2))
plt.savefig("Washington DC Mall University_Random50_0-600_800-
                                    1280_191_Train_Clean_[56:26:16].
                                    png")
plt.show()
```

9.3.3 Simulated Experiments

```
# add Sigma (gaussian) noise
noise_sigma = 0  # Sigma Noise (5, 25, 50, 75, 100)
# That Random Noise, the sigma/gaussian noise is zero...

# add Random noise
random_sigma = 0 # Random Noise (25)
# That Sigma Noise, the random/gaussian noise is zero...

# add Random noise
gaussian_sigma = 30 # Gaussian Noise (30)
# That Sigma/Random Noise, the random/sigma noise is zero...


def calculation_gaussian(value_i_b,B=200,N=30):
```

```
  Upper_left = np.negative(np.power(np.divide(np.subtract(value_i_b
                                    ,B),2),2))
  Upper_right = np.multiply(2, np.power(N, 2))
  value = np.exp(np.divide(Upper_left,Upper_right))
  return value

def Gaussian_Noise(Channel = None):
  B, N = 200, 30
  Upper = calculation_gaussian(Channel, B, N)
  Lower = np.sum(calculation_gaussian(np.arange(1,B), B, N))
  Final = np.multiply(B,np.sqrt(np.divide(Upper,Lower)))
  return Final

def noise_add_by_bands(data):
    h, w, c = data.shape
    noise = []
    for channel in range(c):
        # add alpha noise
        if ((noise_sigma == 5) or (noise_sigma == 25) or (
                                        noise_sigma == 50) or (
                                        noise_sigma == 75) or (
                                        noise_sigma == 100)):
          sigma = np.random.randint(1, noise_sigma)
          noise_sigma_band = np.random.normal(scale=(sigma / 255),
                                        size=[h, w])
          noise.append(noise_sigma_band)
        # add random noise
        if ((random_sigma == 25)):
          random_value = np.random.randint(1, random_sigma)
          random_band = ((random_value + np.random.randn(h, w))/255
                                        )
          noise.append(random_band)
        # add gaussian noise
        if ((gaussian_sigma == 30)):
          value = Gaussian_Noise(channel)
          gaussian_band = np.random.normal(scale=(value / 255),
                                        size=[h, w])
```

```
            noise.append(gaussian_band)

    noise = np.array(noise)
    noise = noise.transpose(1, 2, 0)
    noise_image = np.clip(data + noise, 0, 1).astype(np.float32)
    return noise_image
```

9.3.4 Data Augmentation

These methods are employed in the data augmentation process: Data Scaling. Data Striding. Data Rotation. Data Patch Size. Data Simulated Noise: Various types of noise (Random, Gaussian, and Alpha) are applied for denoising features.

```
def data_aug(img, rot_time, filp_mode):
  if filp_mode == -1:
    return np.rot90(img, k=rot_time)
    # Rotate an array by 90 degrees in the plane specified by axes.
    # Output same as the given input, in case of k = -1
  else:
    return np.flip(np.rot90(img, k=rot_time), axis=filp_mode)
    # Reverse the order of elements in an array along the given
                                        axis.
    # filp_mode: vertically (axis=0), horizontally (axis=1).

clean = []
noise = []
aug_times = 1
scales = [1]
patch_size, stride = 20, 20

def augmentation(data_train, count = 0):
  for s in scales:
    print(f"Per number of total: {s}/{scales}")
    data_scaled = scipy.ndimage.zoom(data_train, (s, s, 1)).astype(
                                        np.float32)
    data_scaled[data_scaled < 0] = 0
```

```
    data_scaled[data_scaled > 1] = 1

    print("data scaled shape of per number (",s,") : ", data_scaled
                                        .shape)
    h_scaled, w_scaled, band_scaled = data_scaled.shape
    for i in range(0, h_scaled - patch_size + 1, stride):
        for j in range(0, w_scaled - patch_size + 1, stride):
            for k in range(0, aug_times):
                count += 1
                x = data_scaled[i:i + patch_size, j:j + patch_size,
                                                  :]

                rot_time = np.random.randint(0, 4) # Generate: 0, 1
                                                  , 2, 3
                filp_mode = np.random.randint(-1, 2) # Generate: -1
                                                  , 0, 1
                x_aug = data_aug(x, rot_time, filp_mode) # Data
                                                  Augmented

                y_aug = noise_add_by_bands(x_aug) # Add the
                                                  Simulated Noise
                x_np = np.array(x_aug, dtype='float32')
                y_np = np.array(y_aug, dtype='float32')
                clean.append(x_np)
                noise.append(y_np)
  print("Total number of augmention : ",count)
```

9.3.5 Utilized Specific Dataset

Test: Simulated Noise Experiments.

```
noise_test = noise_add_by_bands(data_test)
noise_test.shape



fig = plt.figure(frameon=False, figsize = (3, 3))
ax = plt.Axes(fig, [0., 0., 1., 1.])
```

```
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((noise_test[:,:,56],noise_test[:,:,26],
                                    noise_test[:,:,16]),2))
plt.savefig("Washington DC Mall University_Random50_600-800_50-
                                    250_191_Noise_[56:26:16].png")
plt.show()

# Save the Mat file.
# scipy.io.savemat(f"DC_Mall TestDataset Original_Noise_{str(
                                    noise_sigma)}_Data.mat",{'
                                    original': data_test, 'noise':
                                    noise_test})
# scipy.io.savemat(f"DC_Mall TestDataset Original_Noise_Random{str(
                                    random_sigma)}_Data.mat",{'
                                    original': data_test, 'noise':
                                    noise_test})
scipy.io.savemat(f"DC_Mall TestDataset Original_Noise_Gaussian{str(
                                    gaussian_sigma)}_Data.mat",{'
                                    original': data_test, 'noise':
                                    noise_test})
```

Train: Data Augmentation, Simulated Noise Experiments, and Specified Batch Size.

```
augmentation(data_train_DC, count = 0)

original = np.array(clean)
noise = np.array(noise)
print("Original Dataset Shape : ", original.shape, noise.shape)

# Used specific batch_size = 24 = K
batch_size = 24  # K = 24
original_ = original[:batch_size]
noise_ = noise[:batch_size]
print("Batch-Size Dataset Shape : ", original_.shape, noise_.shape)
```

```
# Show the original features of HSI
fig = plt.figure(frameon=False, figsize = (3, 3))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((original_[20,:,:,56],original_[20,:,:,26],
                                    original_[20,:,:,16]),2))
plt.savefig("Washington DC Mall University_Random50_0-600_800-
                                    1280_191_AugTrain_Original_20-
                                    Image_[56:26:16].png")
plt.show()

# Show the noise features of HSI
fig = plt.figure(frameon=False, figsize = (3, 3))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
plt.imshow(np.stack((noise_[10,:,:,56],noise_[10,:,:,26],noise_[10,
                                    :,:,16]),2))
plt.savefig("Washington DC Mall University_Random50_0-600_800-
                                    1280_191_AugTrain_Noise_10-Image_
                                    [56:26:16].png")
plt.show()

# Save the Mat file.
# scipy.io.savemat(f"DC_Mall TrainDataset Original_Noise_{str(
                                    noise_sigma)}_Augmentation_Data.
                                    mat",{'original': original_, '
                                    noise': noise_})
# scipy.io.savemat(f"DC_Mall TrainDataset Original_Noise_Random{str
```

```
                                    (random_sigma)}_Augmentation_Data
                                    .mat",{'original': original_, '
                                    noise': noise_})
scipy.io.savemat(f"DC_Mall TrainDataset Original_Noise_Gaussian{str
                                    (gaussian_sigma)}
                                    _Augmentation_Data.mat",{'
                                    original': original_, 'noise':
                                    noise_})
```

9.4 Indian Pines (Real Dataset Evaluation)

```
def read_HSI():
  indian_pines = scipy.io.loadmat('/content/drive/MyDrive/
                                    Indian_pines.mat')['
                                    indian_pines']
  indian_pines = trim_image_in_linear_stretch(indian_pines)
  print(f"Indian Pines shape: {indian_pines.shape}")
  return indian_pines

X_indian_pines = read_HSI()

# Show the Indian Pines image of HSI.
fig = plt.figure(figsize = (6, 6))
ax = plt.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
fig.add_axes(ax)
plt.axis('off')
read_image = X_indian_pines[:,:,(6,8,203)]
plt.imshow(read_image)
plt.show()

# Save the Mat file.
scipy.io.savemat(f"Indian Pines TestDataset Original
                                    NoiseLess_20x20_PB.mat",{'
```

```
                                    original': X_indian_pines})
```

9.5 Matrix Index

In the specified context, the Matrix Index encompasses three pivotal measures: PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index), and SAM (Spectral Angle Mapper). These measures assess and quantify the quality and resemblance of hyperspectral imaging (HSI) data.

1. PSNR: Measures the ratio of the peak signal power to the noise power in a HSI samples, making it valuable for assessing HSI image quality.

2. SSIM: Evaluates the structural similarity between two HSI images.

3. SAM: Calculates the spectral angle between two spectral vectors, often used in remote sensing and image analysis to assess spectral similarity. - The SAM algorithm only used the vector direction, not to be vector length. - The SAM formula:

$$a = cos^{-1}\left(\frac{\sum_{i=1}^{nb} t_i r_i}{\sqrt{\sum_{i=1}^{nb} t_i^2}\sqrt{\sum_{i=1}^{nb} r_i^2}}\right) \tag{9.1}$$

   - a: spectral angle b/w two spectra

   - n: numbers of bands

   - t_i: reflectance value of band i in the test spectra

   - r_i: reflectance value of band i in the reference spectra

These metrics collectively offer a comprehensive means to assess and compare data quality and similarity.

```
SEED()
def sam(x_true, x_pred):
    assert x_true.ndim ==3 and x_true.shape == x_pred.shape
    h,w,c=x_pred.shape
    sam_rad = []
```

```
    for x in range(x_true.shape[0]):
        for y in range(x_true.shape[1]):
            tmp_pred = x_pred[x, y].ravel()
            tmp_true = x_true[x, y].ravel()
            s = np.sum(np.dot(tmp_pred, tmp_true))
            t = (np.sqrt(np.sum(tmp_pred ** 2))) * (np.sqrt(np.sum(
                                                tmp_true ** 2)))
            th = np.arccos(s/t)
            sam_rad.append(th)
    sam_deg = np.mean(sam_rad)
    return sam_deg

def quantitative_assess(data_clean,test_out):
    psnrs=[]
    ssims=[]
    height,width,band =data_clean.shape
    for b in range(band):
        psnr1 = peak_signal_noise_ratio(data_clean[:, :, b],
                                        test_out[:, :, b],
                                        data_range=1)
        ssim1 = structural_similarity(data_clean[:, :, b], test_out
                                        [:, :, b],win_size=11,
                                        data_range=1,
                                        gaussian_weights=1)
        psnrs.append(psnr1)
        ssims.append(ssim1)
    avg_psnr = np.mean(psnrs)
    avg_ssim = np.mean(ssims)
    Sam=sam(data_clean,test_out)
    return [avg_psnr,avg_ssim,Sam]
```

9.6 Spatial and Channel (Spectral) Attention Modules

The Spatial and Channel (Spectral) Attention Modules consist of two essential components.

- Initially, the Spatial Attention Module strives to enhance the spatial attributes of hyperspectral (HSI) images, with a primary emphasis on boosting spatial features while minimizing disturbances. Its fundamental goal is to elevate image quality by mitigating spatial noise.

- Next, the Module for Spectral (Channel) Attention is crafted to highlight the spectral or channel characteristics of hyperspectral (HSI) images. This specialized module focuses on improving the general quality and uniqueness of spectral bands, thereby enhancing their correlation. These attention modules play a pivotal role in refining and optimizing HSI images to extract valuable information for various applications.

```
SEED()
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()
        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1
        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=padding,
                                            bias=False)
        self.sigmoid = nn.Sigmoid() # real number is reduced to a
                                            value between 0 and 1.

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv1(x)
        return self.sigmoid(x)

class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ):
        super(ChannelAttention, self).__init__()
        # input signal composed of several input planes. The h and
                                            w dimensions of the
                                            output tensor are
```

```
                                        determined by the
                                        parameter output_size.
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

        self.fc1 = nn.Conv2d(in_planes, in_planes // 16, 1, bias=
                                        False)
        self.relu1 = nn.ReLU() # A rectified linear unit (ReLU).
                                        Manage non-linearity to a
                                         deep learning model and
                                        solves the vanishing
                                        gradients issue.
        self.fc2 = nn.Conv2d(in_planes // 16, in_planes, 1, bias=
                                        False)

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
        max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
        out = avg_out + max_out
        return self.sigmoid(out)
```

9.7 Geometrical Characteristics Features Extraction Modules

- Module for Extracting Spatial Features concentrates on acquiring information about the arrangement, patterns, and relationships among objects or elements in spatial data, essentially capturing details about spatial organization and interactions.
- The Multiscale Separable Spectral Features Extraction Module is crafted to grasp the nuances of spectral characteristics. Spectral attributes, intricately linked to the frequency domain, play a crucial role in comprehending diverse facets of images across distinct scales. The distinctive feature lies in their separability, allowing

independent analysis at varying scales for a more holistic view of the data.

```
SEED()
class Geometrical_Characteristics(nn.Module):
  def __init__(self, input_Channel=None, feature_Channel=None, K=
                                        None):
    super(Geometrical_Characteristics, self).__init__()

    self.Spatial_Feature_3 = nn.Sequential(
        nn.Conv2d(input_Channel, feature_Channel, 3, padding=1),
    )
    self.Spatial_Feature_5 = nn.Sequential(
        nn.Conv2d(input_Channel, feature_Channel, 5, padding=2),
    )
    self.Spatial_Feature_7 = nn.Sequential(
        nn.Conv2d(input_Channel, feature_Channel, 7, padding=3),
    )

    self.Spectral_Feature_3 = nn.Sequential(
        nn.Conv3d(1, feature_Channel, (K, 1, 1), 1, (0, 0, 0)),
        nn.Conv3d(feature_Channel, feature_Channel, (1, 3, 3), 1, (
                                            0, 1, 1))
    )
    self.Spectral_Feature_5 = nn.Sequential(
        nn.Conv3d(1, feature_Channel, (K, 1, 1), 1, (0, 0, 0)),
        nn.Conv3d(feature_Channel, feature_Channel, (1, 5, 5), 1, (
                                            0, 2, 2))
    )
    self.Spectral_Feature_7 = nn.Sequential(
        nn.Conv3d(1, feature_Channel, (K, 1, 1), 1, (0, 0, 0)),
        nn.Conv3d(feature_Channel, feature_Channel, (1, 7, 7), 1, (
                                            0, 3, 3))
    )

  def forward(self, Spatial, Spectral):
    Spatial_3 = self.Spatial_Feature_3(Spatial)
    Spatial_5 = self.Spatial_Feature_5(Spatial)
    Spatial_7 = self.Spatial_Feature_7(Spatial)
```

```
        Spectral_3 = self.Spectral_Feature_3(Spectral)
        Spectral_5 = self.Spectral_Feature_5(Spectral)
        Spectral_7 = self.Spectral_Feature_7(Spectral)

        spatial = F.leaky_relu(torch.cat((Spatial_3, Spatial_5,
                                          Spatial_7), dim=1))
        spectral = F.leaky_relu(torch.cat((Spectral_3, Spectral_5,
                                           Spectral_7), dim=1)).squeeze(
                                           2)
        spatial_spectral = torch.cat((spatial, spectral), dim=1)

        return spatial_spectral
```

9.8 Attention and Adjacent Features Extraction Hybrid Dense Network (AAFEHDN)

Decompose Frequency (High and Low): The process of decomposing frequency into high and low components is integral to this model's operation.

- Through the utilization of Attentive Skip Connection for both high and low-frequency features, the model adeptly captures intricate details and broader patterns in the data. This twin-feature strategy augments the model's capacity to render contextually aware predictions and decisions.
- Moreover, the model employs Progressive Spectral Channel Attention (PSCA), dynamically adjusting its emphasis on particular spectral channels to suit varied frequency patterns in the input data based on the task's demands.

Together, these two phases highlight the model's ability to manage and manipulate data, focusing both on frequency characteristics and responsive spectral channel attention.

```
SEED()
```

```
# Attentive Skip Connection (High and Low Frequency Features)
class ASC(nn.Module):
  def __init__(self, channel):
    super().__init__()
    self.weight = nn.Sequential(
        nn.Conv2d(channel * 2, channel, 1),
        nn.LeakyReLU(), # Similar to Relu. Small slope for negative
                                            values instead of a flat
                                            slope. The slope
                                           coefficient is determined
                                            before training.
        nn.Conv2d(channel, channel, 3, 1, 1),
        nn.Sigmoid()
    )

  def forward(self, x, y):
    w = self.weight(torch.cat([x, y], dim=1))
    out = (1 - w) * x + w * y
    return out

# Progressive Spectral Channel Attention (PSCA)
class PSCA(nn.Module):
  def __init__(self, channel, channel_half):
    super().__init__()
    self.w_3 = nn.Conv2d(channel, channel, 1, bias=False)
    self.w_1 = nn.Conv2d(channel, channel_half, 1, bias=False)
    self.w_2 = nn.Conv2d(channel_half, channel, 1, bias=False)
    nn.init.zeros_(self.w_3.weight)

  def forward(self, x):
    x = self.w_3(x) * x + x
    x = self.w_1(x)
    x = F.gelu(x) # Gaussian cumulative distribution function. The
                                        GELU nonlinearity weights
                                        inputs by their percentile,
                                        rather than gates inputs by
                                        their sign as in ReLUs.
```

```
    x = self.w_2(x)
    return x
```

```
# Test the ASC model use this script.
data = torch.randn(1, 80, 200, 200)
model = ASC(80)
output = model(data, data)
output.shape

# Test the PSCA model use this script.
data = torch.randn(1, 80, 200, 200)
model = PSCA(80, 80//2)
output = model(data)
output.shape
```

9.9 Denoising Block (AAFEHDN) Network

The Core Denoising Module (AAFEHDN) stands as the central element in the AAFEHDN structure, embodying a sophisticated and detailed strategy for data processing. This architectural marvel transcends traditional networks, presenting an enchanting sequence of data manipulations. It seamlessly integrates spectral and spatial attention, adaptive correlations, and data transformations into a cohesive composition. Each phase in this elaborate process contributes to refining the input data, shaping it with precision at every step until it emerges as a flawless hyperspectral image, devoid of noise and imperfections. This journey resembles a ballet, where each movement is meticulously orchestrated to enhance the precision and clarity of the final result. With every layer uncovered, the intricacies of hyperspectral data unveil its concealed intricacies, narrating a captivating tale of exploration and ingenuity.

```
SEED()
class AAFEHDN_block(nn.Module):
    def __init__(self,block_num=None, channel=None):
        super(AAFEHDN_block,self).__init__()
        self.group_list_asc_model = []
```

```
        self.group_list_psca_model = []
        channel_half = int(channel//2)
        self.block_number = block_num

        for i in range(0, block_num-1):
          asc_model = ASC(channel)
          self.add_module(name='asc_model_%d' % i, module=asc_model
                                                  )
          self.group_list_asc_model.append(asc_model)

          psca_model = PSCA(channel, channel_half)
          self.add_module(name='psca_model_%d' % i, module=
                                                    psca_model)
          self.group_list_psca_model.append(psca_model)

        self.channel_attention = ChannelAttention(channel)
        self.spatial_attention = SpatialAttention(7)

    def forward(self, input):
      output_feature = [input]
      updated_feature = input
      for index in range(0, self.block_number-1):
        asc_model_group1 = self.group_list_asc_model[index](
                                          updated_feature,
                                          updated_feature)
        channel_attention_asc = self.channel_attention(
                                          asc_model_group1) *
                                          asc_model_group1
        spatial_attention_asc = self.spatial_attention(
                                          channel_attention_asc) *
                                          channel_attention_asc +
                                          input

        psca_model_group2 = self.group_list_psca_model[index](
                                          spatial_attention_asc)
        channel_attention_psca = self.channel_attention(
                                          psca_model_group2) *
```

```
                                        psca_model_group2
        spatial_attention_psca = self.spatial_attention(
                                        channel_attention_psca) *
                                         channel_attention_psca +
                                         input

        output_feature.append(spatial_attention_psca)
        updated_feature = spatial_attention_psca + input

      concat = torch.cat(output_feature, dim=1)
      return concat

# Test the Block by use the this script.
data = torch.randn(1, 80, 200, 200)
model = AAFEHDN_block(block_num=6, channel=80)
output = model(data)
output.shape
```

```
SEED()
class AAFEHDN(nn.Module):
    def __init__(self, input_Channel=1, block_num=3, K=24,
                                        feature_Channel=20,
                                        layer_output=80):
        super(AAFEHDN, self).__init__()
        self.geometricalCharacter = Geometrical_Characteristics(
                                        input_Channel,
                                        feature_Channel, K)
        self.ca_in = ChannelAttention(feature_Channel * 6)
        self.sa_in = SpatialAttention(7)
        self.concat_in = nn.Conv2d(feature_Channel * 6,
                                        layer_output, 3, 1,
                                        padding=1)
        self.AAFEHDN_block = AAFEHDN_block(block_num=block_num,
                                        channel=layer_output)
        self.FR = nn.Conv2d(layer_output*block_num, input_Channel,
                                        3, 1, padding=1)
```

```
    def forward(self, Spatial, Spectral):
        spatial_spectral = self.geometricalCharacter(Spatial,
                                                     Spectral)
        ca_in = self.ca_in(spatial_spectral) * spatial_spectral
        sa_in = self.sa_in(ca_in) * ca_in
        output_Geometrical =  self.concat_in(sa_in)
        AAFEHDN_output = self.AAFEHDN_block(output_Geometrical)
        Residual = self.FR(AAFEHDN_output)
        out = Spatial - Residual
        return out

    def _initialize_weights(self):
      for m in self.modules():
        if isinstance(m, nn.Conv2d):
          init.orthogonal_(m.weight)
          if m.bias is not None:
              init.constant_(m.bias, 0)
        elif isinstance(m, nn.Conv3d):
          init.orthogonal_(m.weight)
          print('init weight')
          if m.bias is not None:
              init.constant_(m.bias, 0)

# Test the Network by use this script.

### Indian Pines
# spatial = torch.randn(1,1,145,145)
# spectral = torch.randn(1,1,24,145,145)

### WDC Mall (Test Shape)
spatial = torch.randn(1,1,200,200)
spectral = torch.randn(1,1,24,200,200)

model = AAFEHDN()

output = model(spatial, spectral)
output.shape
```

9.10 Simulated and Real Dataset

In our research, we engage with both simulated and authentic datasets. Initially, we bring in the prepared Hyperspectral Imaging (HSI) data. Following this, we initiate the procedure of isolating the Noise and Clean attributes within the samples of both our training and testing datasets. This pivotal stage allows us to differentiate and examine the unique qualities of the data, enabling a thorough evaluation of our model's effectiveness.

```
SEED()

TestData = "/content/drive/MyDrive/DC_Mall TestDataset
                                    Original_Noise_50_Data.mat"
TrainData = "/content/drive/MyDrive/DC_Mall TrainDataset
                                    Original_Noise_50_Augmentation_Data
                                    .mat"

# TEST DATA
test_data_DC = scipy.io.loadmat(TestData)
test_data_clean_DC = test_data_DC['original']
test_data_noise_DC = test_data_DC['noise']

# TRAIN DATA
data = scipy.io.loadmat(TrainData)
data_clean = data['original']
data_noise = data['noise']

number, heigth, width, band = data_clean.shape
data = np.zeros((2, number, heigth, width, band))
for n in range(number):
    data[0, n, :, :, :] = data_clean[n, :, :, :]
    data[1, n, :, :, :] = data_noise[n, :, :, :]
data = torch.from_numpy(data.transpose((1, 0, 4, 2, 3)))

test_data_clean_DC.shape, test_data_noise_DC.shape, data.shape
```

9.11 Model and Data Initialization

In the project's initial stage, we concentrated on initializing the model and data. Firstly, we meticulously configured hyperparameters, ensuring essential settings were appropriately set. Following that, we established dedicated folders to archive our model's historical records, encompassing the optimizer and scheduler. Leveraging PyTorch with CUDA support, we efficiently loaded our dataset and model, capitalizing on GPU acceleration. Additionally, we instituted foundational elements crucial for model training, encompassing loss function definition, optimizer configuration, and scheduler setup with specific learning rates. This thorough approach to model and data initialization established the foundation for our ensuing machine learning pursuits.

```
SEED()

batch_size = 24
argsK = 24
n_epoch = 100
k = int(argsK/2)

milestone=[180]
lr = 1e-5 # 0.00005

cuda = torch.cuda.is_available()
print("cuda is ",cuda)

argument_sigma_role = "sigma_50" # Sigma Noise (50)
# argument_sigma_role = "Random_25" # Random Noise (25)
# argument_sigma_role = "Gaussian_200_30" # Gaussian Noise (200,30)

if os.path.isdir(f"AAFEHDN_Denoising_{argument_sigma_role}"):
  shutil.rmtree(f"AAFEHDN_Denoising_{argument_sigma_role}")
  print(f"Delete AAFEHDN_Denoising_{argument_sigma_role} Folder")
if not os.path.exists(f'AAFEHDN_Denoising_{argument_sigma_role}'):
  os.makedirs(f'AAFEHDN_Denoising_{argument_sigma_role}')
  print(f"Create AAFEHDN_Denoising_{argument_sigma_role} Folder")
else:
```

```
  pass

save_dir = os.path.join('model')
if os.path.exists(save_dir):
    shutil.rmtree(save_dir)
    print("Delete Model Folder")
if not os.path.exists(save_dir):
  os.mkdir(save_dir)
  print("Create Model Folder")
else:
  pass

DLoader = DataLoader(dataset=data, num_workers=0, drop_last=False,
                                  batch_size=batch_size, shuffle=
                                  True)

model = AAFEHDN()

if cuda:
  model = model.cuda()

criterion = nn.L1Loss().cuda()
optimizer = optim.Adam(model.parameters(), lr=lr)
scheduler = MultiStepLR(optimizer, milestones=milestone, gamma=0.2)
```

9.12 Classification and Target Detection AAFEHDN

In the realm of classification and target detection, a method known as AAFEHDN is employed. This approach involves a two-step process: training and testing.

- Throughout the training process, a model is developed adhering to defined procedures, like configuring 100 epochs and employing a dataloader batch size of 24 for dataset iteration. The model acquires insights into the correlations among distinct bands in Hyperspectral Imaging (HSI) data, a vital aspect for precise classification

and target detection. Subsequent to training, a quantitative assessment is executed, and key files are stored for future utilization.

- Furthermore, the approach utilizes K-Adjacent Characteristics, establishing correlations among the initial, final, and central bands of the HSI dataset. This arrangement molds the dataset into a three-dimensional structure, contributing to improved precision in Spatial-Spectral samples for enhanced classification and target detection.

```
SEED()

psnrs_matric, ssims_matric, sams_matric, time_train, time_test =
                                    list(), list(), list(), list(),
                                    list()
for epoch in range(0, n_epoch):
  model.train()
  lr = optimizer.state_dict()['param_groups'][0]['lr']
  epoch_loss = 0
  start_time = time.time()

  for _, batch_yx in enumerate(DLoader):
    optimizer.zero_grad()
    if cuda:
      batch_x, batch_y = batch_yx[:,0,:,:].cuda(), batch_yx[:,1,:,:
                                          ].cuda()

    iter_band = np.arange(band)
    np.random.shuffle(iter_band)

    for b in iter_band:
      x = batch_y[:,b, :, :]
      noise_free = batch_x[:,b, :, :]

      x = torch.unsqueeze(x, dim=1).type(torch.FloatTensor)
      noise_free = torch.unsqueeze(noise_free, dim=1).type(torch.
                                          FloatTensor)
```

```
    if b < k: # first
        y = batch_y[:,0:argsK, :, :]
    elif b < band - k: # last
        y = torch.cat((batch_y[:,b - k:b, :, :],batch_y[:,b + 1:b
                                                + k + 1, :, :]),1)
    else:
        y = batch_y[:,band - argsK:band, :, :]

    y = torch.unsqueeze(y, dim=1).type(torch.FloatTensor)

    if cuda:
      x = x.cuda()
      y = y.cuda()
      noise_free = noise_free.cuda()

    learned_image=model(x, y)
    loss = criterion(learned_image, noise_free)
    epoch_loss += loss.item()
    loss.backward()
    optimizer.step()

scheduler.step()
batch_number = data.size(0) // batch_size
final_time = time.time() - start_time
time_train.append(final_time)

if(epoch<n_epoch):
  model.eval()
  data_noise = torch.from_numpy(test_data_noise_DC)
  hight,width,cs=data_noise.shape
  data_noise = data_noise.permute(2, 1, 0)
  test_out = torch.zeros(data_noise.shape).type(torch.FloatTensor
                                        )

  start_time_test = time.time()
  for channel_i in range(cs):
    x_data = data_noise[channel_i, :, :]
```

```
    x_data = torch.unsqueeze(x_data, dim=0).type(torch.
                                        FloatTensor)
    x_data = torch.unsqueeze(x_data, dim=0).type(torch.
                                        FloatTensor)

    if channel_i < k:
        y_data = data_noise[0:argsK, :, :]
    elif channel_i < cs - k:
        y_data = torch.cat((data_noise[channel_i - k:channel_i, :
                                            , :],data_noise[
                                            channel_i + 1:channel_i
                                             + k + 1, :, :]))
    else:
        y_data = data_noise[cs - argsK:cs, :, :]

    y_data = torch.unsqueeze(y_data, dim=0).type(torch.
                                        FloatTensor)
    y_data = torch.unsqueeze(y_data, dim=0).type(torch.
                                        FloatTensor)

    if cuda:
      x_data,y_data = x_data.cuda(), y_data.cuda()

    with torch.no_grad():
      out = model(x_data, y_data)

    out = out.squeeze()
    test_out[channel_i,:,:] = out

end_time_test = time.time() - start_time_test
time_test.append(end_time_test)

test_out = test_out.permute(2,1,0)
denoise_image_out = test_out.cpu().numpy()

PSNR,SSIM,SAM = quantitative_assess(test_data_clean_DC,
                                    denoise_image_out)
```

```
    psnrs_matric.append(PSNR)
    ssims_matric.append(SSIM)
    sams_matric.append(SAM)


  total_epoch_loss = epoch_loss / batch_number
  print(f"Epoch : {epoch+1}/{n_epoch} and Loss : ({total_epoch_loss
                                    :.5f}) and (PSNR : {PSNR} &
                                    SSIM : {SSIM} & SAM : {SAM})
                                    and Train Time : ({final_time:.
                                    1f}) Sec and Test Time : ({
                                    end_time_test:.1f}) Sec")

  torch.save(model, os.path.join(save_dir, 'model_%03d.pth' % (
                                    epoch + 1)))
  checkpoint = {
                'optimizer': optimizer.state_dict(),
                "epoch": epoch,
                'lr_schedule': scheduler.state_dict()
              }
  torch.save(checkpoint, os.path.join(save_dir, 'checkpoint_%03d.
                                    pth' % (epoch + 1)))
```

9.13 Quantitative Evaluation (PSNR-SSIM-SAM)

To evaluate our outcomes, we compute the peak values and standard deviations for each of these metric indicators. Furthermore, we ascertain the mean time necessary for both model training and testing. This thorough examination furnishes us with valuable perspectives on the efficiency and efficacy of our model, aiding us in making informed decisions and optimizations.

```
SEED()

psnr_max, psnr_std = np.amax(psnrs_matric), np.std(psnrs_matric,
                                    ddof=1)
```

```
ssim_max, ssim_std = np.amax(ssims_matric), np.std(ssims_matric,
                                    ddof=1)
sam_max, sam_std = np.amin(sams_matric), np.std(sams_matric, ddof=1
                                    )
traintime_average, testtime_average = np.average(final_time), np.
                                    average(time_test)

pt = PrettyTable()
print(f"Note: Improvements shown are over original pairs of
                                    Hyperspectral Image using {
                                    argument_sigma_role}")
pt.field_names = ["Model", "MPSNR", "MSSIM", "MSAM", "TRAIN_TIME",
                                    "TESTTIME"]

pt.add_row(["HSI_CDCN Denoising Model ","{:.4f} {:.4f}".format(
                                    psnr_max, psnr_std),"{:.4f} {:.4f
                                    }".format(ssim_max,ssim_std),"{:.
                                    4f} {:.4f}".format(sam_max,
                                    sam_std), "Average {:.4f}".format
                                    (traintime_average), "Average {:.
                                    4f}".format(testtime_average)])
data = pt.get_string()
with open(f'./AAFEHDN_Denoising_{argument_sigma_role}/
                                    AAFEHDN_Denoising_{
                                    argument_sigma_role}.txt', 'w')
                                    as f: f.write(data)
print(pt)
```

```
# Save the data in CSV file for further experiments.
SEED()

data = {
    "PSNR": psnrs_matric,
    "SSIM": ssims_matric,
    "SAM": sams_matric,
    "TrainTime": final_time,
    "TestTime": time_test,
```

```
}
import pandas as pd
data = pd.DataFrame().from_dict(data)
data.to_csv(f'./AAFEHDN_Denoising_{argument_sigma_role}/CSV_{
                                    argument_sigma_role}.csv', index=
                                    False)
data
```

```
# PSNR Graph Plot
epochs = range(0, n_epoch)
plt.plot(epochs, psnrs_matric, 'b', label='MPSNR')
plt.title(f'MPSNR with {argument_sigma_role}')
plt.xlabel('Bands')
plt.ylabel('MPSNR')
plt.legend()
plt.xlim(0,n_epoch)
plt.savefig(f'./AAFEHDN_Denoising_{argument_sigma_role}/
                                    AAFEHDN_Denoising_Graph_MPSNR_{
                                    argument_sigma_role}.png')
plt.show()

# SSIM Graph Plot
epochs = range(0, n_epoch)
plt.plot(epochs, ssims_matric, 'b', label='MSSIM')
plt.title(f'MSSIM with {argument_sigma_role}')
plt.xlabel('Bands')
plt.ylabel('MSSIM')
plt.legend()
plt.xlim(0,n_epoch)
plt.savefig(f'./AAFEHDN_Denoising_{argument_sigma_role}/
                                    AAFEHDN_Denoising_Graph_MSSIM_{
                                    argument_sigma_role}.png')
plt.show()

# SAM Graph Plot
epochs = range(0, n_epoch)
plt.plot(epochs, sams_matric, 'b', label='MSAM')
```

```
plt.title(f'MSAM with {argument_sigma_role}')
plt.xlabel('Bands')
plt.ylabel('MSAM')
plt.legend()
plt.xlim(0,n_epoch)
plt.savefig(f'./AAFEHDN_Denoising_{argument_sigma_role}/
                                    AAFEHDN_Denoising_Graph_MSAM_{
                                    argument_sigma_role}.png')
plt.show()
```

9.14 Image Prediction

In the realm of image forecasting, we employ Matplotlib's image plotting functionality to illustrate the Clean, Noise, and Denoise Hyperspectral Imaging (HSI) dataset. These visual depictions, presented as figures, offer a lucid and intuitive view of the data. This facilitates the analysis and comparison of distinct states within the HSI dataset, encompassing clean images, noisy renditions, and denoised versions. Such a visual strategy serves as a valuable tool for researchers and analysts, fostering deeper insights into the dataset and the efficacy of denoising techniques, thereby enhancing comprehension of hyperspectral image processing.

```
SEED()

def show_image(data_noise,test_out,data_clean):
  def Noise_Image():
    fig = plt.figure(frameon=False, figsize = (5, 5))
    ax = plt.Axes(fig, [0., 0., 1., 1.])
    ax.set_axis_off()
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    fig.add_axes(ax)
    plt.imshow(data_noise[:, :, (56,25,16)])
    plt.savefig(f"./AAFEHDN_Denoising_{argument_sigma_role}/
                                    Washington DC Mall
                                    University_Noise_{
```

```
                                        argument_sigma_role}.png")
    plt.axis('off')
    plt.title('noise image')
def Denoise_Image():
    fig = plt.figure(frameon=False, figsize = (5, 5))
    ax = plt.Axes(fig, [0., 0., 1., 1.])
    ax.set_axis_off()
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    fig.add_axes(ax)
    plt.imshow(test_out[:, :, (56,25,16)])
    plt.savefig(f"./AAFEHDN_Denoising_{argument_sigma_role}/
                                        Washington DC Mall
                                        University_Denoise_{
                                        argument_sigma_role}.png")
    plt.axis('off')
    plt.title('denoise image')
def Clean_Image():
    fig = plt.figure(frameon=False, figsize = (5, 5))
    ax = plt.Axes(fig, [0., 0., 1., 1.])
    ax.set_axis_off()
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    fig.add_axes(ax)
    plt.imshow(data_clean[:, :, (56,25,16)])
    plt.savefig(f"./AAFEHDN_Denoising_{argument_sigma_role}/
                                        Washington DC Mall
                                        University_Clean_{
                                        argument_sigma_role}.png")
    plt.axis('off')
    plt.title('clean image')

Noise_Image()
Denoise_Image()
Clean_Image()
plt.show()
```

```
# Show the Pictures.
show_image(test_data_noise_DC,denoise_image_out,test_data_clean_DC)

# Save the Denoise Mat data file.
scipy.io.savemat(f"./AAFEHDN_Denoising_{argument_sigma_role}/
                                    Washington DC Mall
                                    University_Denoise_{
                                    argument_sigma_role}
                                    _Dataset_Image.mat",{'Image':
                                    denoise_image_out})
```

9.15 Comparison Model: MemNet

```
class MemNet(nn.Module):
    def __init__(self, in_channels, channels, num_memblock,
                                        num_resblock):
        super(MemNet, self).__init__()
        out_channnels = 1
        self.feature_extractor = BNReLUConv(in_channels, channels)
        self.reconstructor = BNReLUConv(channels, out_channnels)
        self.dense_memory = nn.ModuleList(
            [MemoryBlock(channels, num_resblock, i+1) for i in
                                                range(num_memblock)]
        )
        self.freeze_bn = True
        self.freeze_bn_affine = True

    def forward(self, x):
        residual = x
        out = self.feature_extractor(x)
        ys = [out]
        for memory_block in self.dense_memory:
            out = memory_block(out, ys)
        out = self.reconstructor(out)
        # out = out + residual
        return out
```

```
class MemoryBlock(nn.Module):
    """Note: num_memblock denotes the number of MemoryBlock
                                        currently"""
    def __init__(self, channels, num_resblock, num_memblock):
        super(MemoryBlock, self).__init__()
        self.recursive_unit = nn.ModuleList(
            [ResidualBlock(channels) for i in range(num_resblock)]
        )
        self.gate_unit = BNReLUConv((num_resblock+num_memblock) *
                                            channels,
        channels, 1, 1, 0)

    def forward(self, x, ys):
        """ys is a list which contains long-term memory coming
        from previous memory block xs denotes the short-term
        memory coming from recursive unit
        """
        xs = []
        residual = x
        for layer in self.recursive_unit:
            x = layer(x)
            xs.append(x)

        gate_out = self.gate_unit(torch.cat(xs+ys, 1))
        ys.append(gate_out)
        return gate_out


class ResidualBlock(torch.nn.Module):

    def __init__(self, channels, k=3, s=1, p=1):
        super(ResidualBlock, self).__init__()
        self.relu_conv1 = BNReLUConv(channels, channels, k, s, p)
        self.relu_conv2 = BNReLUConv(channels, channels, k, s, p)
```

```
    def forward(self, x):
        residual = x
        out = self.relu_conv1(x)
        out = self.relu_conv2(out)
        out = out + residual
        return out


class BNReLUConv(nn.Sequential):
    def __init__(self, in_channels, channels, k=3, s=1, p=1,
                                        inplace=True):
        super(BNReLUConv, self).__init__()
        self.add_module('bn', nn.BatchNorm2d(in_channels))
        self.add_module('relu', nn.ReLU(inplace=inplace))
        self.add_module('conv', nn.Conv2d(in_channels, channels, k,
                                            s, p, bias=False))

#===> Test the Code.
## Train and Dataset
spatial = torch.randn(24, 1, 20, 20).cuda()
model = memnet().cuda()
output = model(spatial)
output.shape
```

9.16 Comparison Model: DeNet

```
cfg = ([64, 1], [64, 1], [64, 1],
       [128, 1], [128, 1], [128, 1],
       [256, 2], [256, 2], [256, 2],
       [128, 1], [128, 1], [128, 1],
       [64, 1], [64, 1], [64, 1], [64, 1])

class DeNet(nn.Module):
    def __init__(self, kernel_size=3, init_weights=True):
        super(DeNet, self).__init__()
        layers = []
```

```
        layer_output = 1
        final_channels = 1
        out_channels = 64
        layers.append(nn.Conv2d(
            in_channels=layer_output,
            out_channels=out_channels,
            kernel_size=kernel_size,
            padding=1, bias=True))
        layers.append(nn.ReLU(inplace=True))
        in_channels = out_channels
        for out_channels, dilation in cfg:
            if dilation == 1:
                padding = 1
            elif dilation == 2:
                padding = 2
            layers.append(nn.Conv2d(in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=kernel_size,
            padding=padding,
            dilation=dilation,
            bias=False))
            layers.append(nn.BatchNorm2d(num_features=out_channels)
                                                    )
            layers.append(nn.ReLU(inplace=True))
            in_channels = out_channels

        layers.append(nn.Conv2d(in_channels=64,
        out_channels=final_channels,
        kernel_size=kernel_size,
        padding=1, bias=False))
        self.denet = nn.Sequential(*layers)

    def forward(self, Spatial):
        out = self.denet(Spatial)
        return Spatial - out

#===> Test the Code.
```

```
## Train and Dataset
spatial = torch.randn(24,1,20,20).cuda()
model = DeNet().cuda()
output = model(spatial)
output.shape
```

Chapter 10

Implementation Details

Here, we transition to Iterations 3 and 4, presenting users with a comprehensive exposition of the complete implementation process for the AAFEHDN Network, specifically designed for Hyperspectral Image denoising. In this phase, we provide in-depth insights into the implementation of every individual component, accompanied by detailed descriptions of their functionalities. Additionally, we delve into the experiments conducted with comparison models and offer an elaborate examination of the code used in these experiments.

Chapter 11

User Manual

In the context of Hyperspectral Denoising with the AAFEHDN framework, this Final Year Project (FYP) represents a substantial contribution to the Research and Development field. While the User Manual is not explicitly detailed in this report, we outline key points to facilitate learning and encourage further development in this FYP.

- Review of Literature: We present foundational concepts from the literature, providing guidance on initiating independent research.
- Project Vision: Detailed objectives, scope, business aspects, and problem statements form the core of our project vision.
- Software Requirements Specifications: Quality attributes, use cases, sequence diagrams, and the test and development plan are meticulously covered in our software requirements specifications.
- Iterations: Each concept and procedural step for commencing this FYP is highlighted, emphasizing the importance of following each iteration for learning and contributing further.
- Web Application: Due to GPU constraints taking 25 minutes for AAFEHDN model training, building a web application faced challenges. However, detailed steps for running and predicting results are provided in notebooks, overcoming the GPU limitations.

Chapter 12

Conclusions and Future Work

In summary, the domain of hyperspectral image (HSI) processing confronts substantial challenges tied to environmental intricacies, corruption, and degradation, resulting in diverse noise types. While traditional image denoising methods have found success, CNN-based HSI denoising methods grapple with insufficient noise suppression and feature extraction. To address these challenges, we introduced the revolutionary HSI denoising algorithm, the Attention and Adjacent Features - Hybrid Dense Network (AAFHDN). This algorithm excels in breaking down high-frequency features, safeguarding geometrical characteristics, and extracting band correlation of neighboring spatial and multi-scale separable spectral features. Experimental assessments on simulated and real-world noisy images underscored AAFHDN's superior performance over traditional methods, both quantitatively and visually. The enhanced denoising capabilities of our method hold promise for improving subsequent classification and target detection tasks in HSIs.

Concerning the HSI data at hand, careful preprocessing techniques, including cropping, introducing simulated noise, data augmentation, and defining specific batch sizes, contribute to a well-organized dataset comprising Training, Testing, and Matric components. Our approach underscores the critical analysis of spectral band correlation within K-Adjacent noisy bands during HSI data processing. Preserving intricate geometrical characteristics is a foundational principle achieved through feature extraction modules operating in both spatial and spectral dimensions. This guarantees the retention of essential geometrical details and structured prior information for further analysis. The decompose

frequency process, utilizing a hybrid dense network with attention modules, effectively separates higher and lower frequency noisy features. The denoising process's prediction of clean data plays a crucial role in classification and target detection, offering more accurate insights for improved outcomes.

Bibliography

[1] Liu, J., Zhang, L., Zhang, L., & Gao, W. (2020). Hyperspectral image denoising using 3-D graph Fourier transform. IEEE Transactions on Geoscience and Remote Sensing, 58(7), 4832-4845.

[2] Liu, W., Zhang, L., Wu, X., & Zhang, D. (2021). Hyperspectral image denoising via nonlocal low-rank tensor decomposition. IEEE Transactions on Geoscience and Remote Sensing, 59(9), 7892-7907.

[3] Wang, Z., Zhang, L., & Zhang, L. (2021). Hyperspectral image denoising via multi-attention weighted low-rank representation. IEEE Transactions on Geoscience and Remote Sensing, 59(12), 10055-10068.

[4] Zhang, L., Yang, J., Zhang, D., & Zhan, Y. (2017). Hyperspectral image denoising using low-rank matrix factorization with adaptive graph regularization. IEEE Transactions on Geoscience and Remote Sensing, 58(5), 3681-3694.

[5] Li, X., Wang, S., Zhang, L., & Xu, Z. (2019). Hyperspectral image denoising based on structured low-rank matrix factorization. IEEE Transactions on Geoscience and Remote Sensing, 57(6), 3922-3936.

[6] Chen, X., et al. (2021). Hyperspectral image denoising using spatial filtering and tensor sparsity. IEEE Transactions on Geoscience and Remote Sensing, 59(8), 6673-6687.

[7] Lu, J., et al. (2021). Hyperspectral image denoising with adaptive spectral filtering and total variation regularization. IEEE Transactions on Geoscience and Remote Sensing, 59(3), 2395-2408.

[8] Wang, Z., et al. (2021). Hyperspectral image denoising based on spatial-spectral decomposition and spectral filtering. IEEE Transactions on Geoscience and Remote Sens-

ing, 59(7), 5865-5877.

[9] Han, Y., et al. (2021). Hyperspectral image denoising via nonlocal low-rank tensor approximation with spectral filtering. IEEE Access, 9, 15666-15677.

[10] Wang, Z., et al. (2021). Hyperspectral image denoising based on spatial-spectral decomposition and spectral filtering. IEEE Transactions on Geoscience and Remote Sensing, 59(7), 5865-5877.

[11] Lu, Z., et al. (2021). Hyperspectral image denoising using wavelet transform and spatial-spectral nonlocal self-similarity. Remote Sensing, 13(15), 2964.

[12] Zhang, Z., et al. (2022). Hyperspectral image denoising using spatial-spectral wavelet transform and nonlocal self-similarity. IEEE Transactions on Geoscience and Remote Sensing, 60(1), 411-424.

[13] Chen, X., et al. (2022). Hyperspectral image denoising using wavelet transform and tensor sparsity. IEEE Transactions on Geoscience and Remote Sensing, 60(3), 2244-2258.

[14] Li, H., et al. (2022). Hyperspectral image denoising using wavelet-based tensor low-rank and sparse decomposition. Remote Sensing, 14(7), 1376.

[15] Liu, J., et al. (2021). Hyperspectral image denoising using wavelet-based principal component analysis. Remote Sensing, 13(13), 2547.

[16] Kong, W., et al. (2021). Hyperspectral image denoising based on low-rank and sparse decomposition with adaptive principal component analysis. Remote Sensing, 13(20), 4126.

[17] Gao, J., et al. (2022). Hyperspectral image denoising using low-rank and sparse decomposition with adaptive principal component analysis. Remote Sensing, 14(4), 782.

[18] Li, M., et al. (2022). Hyperspectral image denoising based on low-rank and sparse decomposition with adaptive principal component analysis. Journal of Applied Remote Sensing, 16(3), 036502.

[19] Jiang, H., Li, C., & Yi, C. (2019). Sparse and low-rank hyperspectral image restoration via total variation regularization. IEEE Transactions on Geoscience and Remote

Sensing, 57(10), 8041-8055.

[20] Zhang, Y., & Gao, X. (2020). Hyperspectral image denoising via adaptive collaborative sparse representation. IEEE Transactions on Geoscience and Remote Sensing, 58(4), 2836-2852.

[21] Chen, Y., et al. (2020). Hyperspectral image denoising using a combination of low-rank and sparse priors. IEEE Transactions on Geoscience and Remote Sensing, 58(6), 4016-4029.

[22] Li, H., et al. (2021). HSI-Net: Hyperspectral image denoising via self-attention convolutional neural network. IEEE Transactions on Geoscience and Remote Sensing, 59(9), 7529-7543.

[23] Fu, X., et al. (2022). Hyperspectral image denoising using hybrid residual attention network. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 15, 4817-4829.

[24] Hui, F., et al. (2020). Hyperspectral Image Denoising with U-Net. IEEE Geoscience and Remote Sensing Letters, 17(4), 732-736.

[25] Chen, Y., et al. (2020). Hyperspectral Image Denoising Using a 3D Convolutional U-Net. IEEE Transactions on Geoscience and Remote Sensing, 58(12), 8757-8771.

[26] Li, C., et al. (2022). Hyperspectral Image Denoising Using Global and Local Low-Rank Models. IEEE Transactions on Geoscience and Remote Sensing, 60(6), 4409-4422.

[27] Zhang, H., et al. (2020). Hyperspectral image denoising via local low-rank tensor approximation. IEEE Transactions on Geoscience and Remote Sensing, 58(8), 5553-5569.

[28] Optical Remote Sensing of Land Surface, Processing Hyperspectral Images. Feb. 2023

[29]: Q. Yuan, Q. Zhang, J. Li, H. Shen and L. Zhang, "Hyperspectral Image Denoising Employing a Spatial–Spectral Deep Residual Convolutional Neural Network," in IEEE Transactions on Geoscience and Remote Sensing, vol. 57, no. 2, pp. 1205-1218, Feb. 2019.

[30]: Lai, Z., and Fu, Y. (2023). Mixed Attention Network for Hyperspectral Image Denoising. arXiv preprint arXiv:2301.11525.

[31]: X. Cao, X. Fu, C. Xu and D. Meng, "Deep Spatial-Spectral Global Reasoning Network for Hyperspectral Image Denoising," in IEEE Transactions on Geoscience and Remote Sensing, vol. 60, pp. 1-14, 2022, Art no. 5504714.

[32]: Y. Chen, W. Cao, L. Pang and X. Cao, "Hyperspectral Image Denoising With Weighted Nonlocal Low-Rank Model and Adaptive Total Variation Regularization," in IEEE Transactions on Geoscience and Remote Sensing, vol. 60, pp. 1-15, 2022, Art no. 5544115.

[33]: Zhuang, L., and Bioucas-Dias, J. M. (2018). Fast hyperspectral image denoising and inpainting based on low-rank and sparse representations. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 11(3), 730-742.

[34]: H. Zhang, H. Chen, G. Yang and L. Zhang, "LR-Net: Low-Rank Spatial-Spectral Network for Hyperspectral Image Denoising," in IEEE Transactions on Image Processing, vol. 30, pp. 8743-8758, 2021

[35]: Q. Shi, X. Tang, T. Yang, R. Liu and L. Zhang, "Hyperspectral Image Denoising Using a 3-D Attention Denoising Network," in IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 12, pp. 10348-10363, Dec. 2021

[36]: H. Sun, M. Liu, K. Zheng, D. Yang, J. Li and L. Gao, "Hyperspectral Image Denoising via Low-Rank Representation and CNN Denoiser," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 15, pp. 716-728, 2022.

[37]: Huang, X., Du, B., Tao, D., and Zhang, L. (2020). Spatial-spectral weighted nuclear norm minimization for hyperspectral image denoising. Neurocomputing, 399, 271-284.

[38]: H. Ma, G. Liu and Y. Yuan, "Enhanced Non-Local Cascading Network with Attention Mechanism for Hyperspectral Image Denoising," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 2448-2452.

[39]: A. Maffei, J. M. Haut, M. E. Paoletti, J. Plaza, L. Bruzzone and A. Plaza, "A Single

Model CNN for Hyperspectral Image Denoising," in IEEE Transactions on Geoscience and Remote Sensing, vol. 58, no. 4, pp. 2516-2529, April 2020.

[40]: J. Lin, T. -Z. Huang, X. -L. Zhao, T. -X. Jiang and L. Zhuang, "A Tensor Subspace Representation-Based Method for Hyperspectral Image Denoising," in IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 9, pp. 7739-7757, Sept. 2021.

[41]: Q. Yuan, Q. Zhang, J. Li, H. Shen and L. Zhang, "Hyperspectral Image Denoising Employing a Spatial–Spectral Deep Residual Convolutional Neural Network," in IEEE Transactions on Geoscience and Remote Sensing, vol. 57, no. 2, pp. 1205-1218, Feb. 2019.

[42]: GlobeNewswire News Room, “The Worldwide Hyperspectral Imaging System Industry is Expected to Reach 35.8 Billion by 2026”. February, 2023

[43] R. O. Green et al., “Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS),” Remote Sens. Environ., vol. 65, no. 3, pp. 227–248, Sep. 1998

[44] Hyperspectral data set, lesun.weebly.com/hyperspectral-data-set, March 29, 2023

[45] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. Memnet: A persistent memory network for image restoration. In Proceedings of the IEEE international conference on computer vision, pages 4539–4547, 2017.

[46] Yi Chang, Luxin Yan, Houzhang Fang, Sheng Zhong, and Wenshan Liao. Hsi-denet: Hyperspectral image restoration via convolutional neural network. IEEE Transactions on Geoscience and Remote Sensing, 57(2):667–682, 2018.