TÉCNICO LISBOA

Embedded System for Onboard Object Detection in Hyperspectral Images

João Filipe Santos Lopes

2^{nd} Cycle Integrated Project Report in

Electrical and Computer Engineering

Supervisors: Prof. Horácio Neto
Prof. Mário Véstias

January 2025

Contents

1

Introduction

The goal of this work is to study object detection algorithms for Hyperspectral Image (HSI), to design an onboard object detection application, which will serve as the foundation for a future Master's thesis. The proposed system aims to be implemented on a device leveraging hardware-software co-design techniques to ensure real-time performance, efficient resource utilization, and low energy consumption.

1.1 Motivation

Hyperspectral images are becoming increasingly important in today's world due to the vast amount of information they contain. These images capture detailed spectral data across multiple wavelengths, making them useful for a wide range of applications, including agriculture, military surveillance, mining, environmental monitoring, and urban planning. Extracting useful information from HSIs is essential for these applications, and one of the key areas of research is Hyperspectral image target detection (HSITD).

Compared to traditional Red-Green-Blue (RGB) images, HSIs offer much richer information by capturing more detailed spectral signatures. However, this also makes them significantly more data-heavy, posing challenges for processing and storage. As a result, traditional object detection methods often fall short in handling the unique properties of HSIs.

Deep learning methods have been successfully applied to hyperspectral object detection tasks and are crucial for automating the analysis of HSIs. However, due to their high computational requirements, these methods are often run in the cloud, separate from the image acquisition hardware. This introduces latency, potential security risks, and delays caused by transmitting large amounts of data.

One solution to these challenges is to run object detection models directly on embedded systems, close to the acquisition system. This approach eliminates latency issues and improves data security by avoiding the need to send large images to external servers. However, implementing such models on embedded systems comes with its own set of challenges, including power consumption, cost, and size constraints, especially for onboard applications in aerospace or field operations.

To achieve real-time onboard detection, it is necessary to study the available object detection techniques for HSIs and design a customized hardware accelerator architecture. This involves carefully analyzing and optimizing a deep learning model to achieve the best possible implementation on a System-on-Chip Field Programmable Gate Array (SoC-FPGA) platform.

1.2 Objectives

The main objective of this work is to study existing hyperspectral image target detection algorithms, particularly deep learning implementations, to review and optimize one for onboard SoC-FPGA implementation, that is:

- Study hyperspectral target detection algorithms;
- Investigate hardware implementations of key features in hyperspectral image target detection;
- Implement and run some algorithms to experiment with parameters in preparation for later hardware implementation.

1.3 Document Outline

This work is divided into four major sections. The *Introduction* presents the motivation and objectives of this work. The *State of the Art* section describes HSIs, traditional hyperspectral target detection methods, deep learning architectures for the same applications, and current Field Programmable Gate Array (FPGA) implementations. The *Preliminary Work* section outlines the research conducted to better understand the studied architectures and prepare for later hardware implementation. Finally, the *Work Plan* section defines the future objectives and timeline for the Master's thesis work, including time management and task planning.

2

State of the Art

This chapter covers the foundational concepts of hyperspectral imaging and HSITD, with a focus on deep learning methods, particularly convolutional neural networks (CNNs) and architectures that use them. It explores the principles of hyperspectral imaging and discusses the implementation of these models on field-programmable gate arrays (FPGAs).

2.1 Hyperspectral Imaging

Hyperspectral images capture a broad range of spectral data. Unlike traditional Red, Green, and Blue (RGB) channels, hyperspectral images contain hundreds to thousands of channels, each corresponding to a specific wavelength (See Figure 2.1).

For example, in a 30x30x130 hyperspectral image, we have a 30x30 spatial image, where each pixel is associated with 130 different wavelength bands. This enables the acquisition of detailed spectral information. Since different materials reflect light uniquely at various wavelengths, hyperspectral data for each material serves as a unique ”fingerprint.” This capability makes hyperspectral imaging highly valuable in fields such as military surveillance, medical diagnostics, and material analysis, offering a vast range of applications.

These images are captured using specialized devices known as hyperspectral cameras. Each camera provides key information, including spatial resolution (images of dimension W (width) $\times$ and L (length)) and spectral resolution (the number of wavelength bands - N), resulting in a $W \times L \times N$ data structure. As expected, hyperspectral images are significantly larger compared to traditional RGB imagesThis increased data volume presents challenges for neural networks, as the larger input sizes demand more computational resources and efficient data processing techniques.

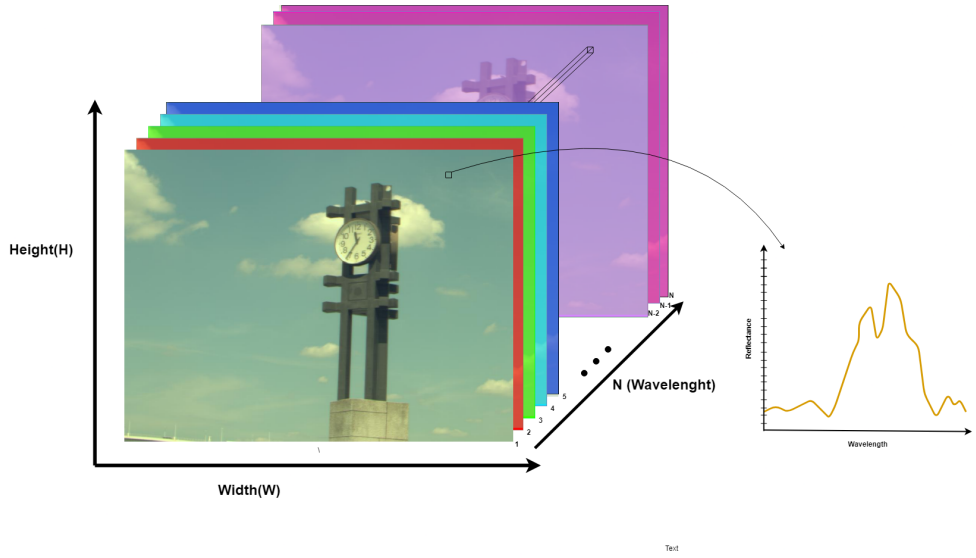

Figure 2.1: Hyperspectral Image, Clock_image from HSOD-Dataset [1]

2.1.1 Dimensionality Reduction of Hyperspectral Images

As mentioned above, hyperspectral images have three components: $W \times L \times N$, where $W \times L$ represents the spatial resolution, and N is the spectral resolution. For example, an image with dimensions 1024×720 and spectral data collected between 400–720 nm at 10 nm intervals has a spatial resolution of 1024×720 and a spectral resolution of 80 bands. This results in a total of 58,982,400 values to be stored for a single image.

Due to the high dimensionality of HSIs, dimensionality reduction is often necessary to facilitate computation. HSIs can be reduced in both spatial and spectral resolution. For spatial resolution, traditional techniques like downsampling or cropping can be applied, similar to reducing RGB images. For HSIs, spatial resolution reduction involves scaling down the $W \times L$ dimensions while preserving key structural information. Spectral resolution reduction, on the other hand, involves techniques that reduce the N bands, ensuring minimal loss of spectral information.

2.1.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is one of the most widely used algorithms for spectral resolution reduction in HSIs. It assumes that neighboring bands often contain redundant information about the object. PCA reduces the spectral dimension by eliminating the correlation between bands, effectively compressing the data while retaining the most significant spectral features [2]. Take a hyperspectral image with 80 spectral bands, each representing a specific wavelength. Each band captures data for the same spatial scene but at a slightly different wavelength, resulting in significant redundancy. For example, the reflectance at bands 20, 21, and 22 should be very similar for the same material because

adjacent wavelengths often contain overlapping information. PCA analyzes these 80 bands and finds patterns of variation.The first principal component (PC1) captures the most significant variance (i.e., the most important spectral information). The second principal component (PC2) captures the next highest variance, and so on. In practice, rather than keeping all 100 bands, PCA allows you to select a smaller subset of components that still retain most of the essential information. This drastically reduces the dimensionality of the data while minimizing the loss of important spectral details.

2.1.3 Optimal Neighborhood Reconstruction (ONR)

Optimal Neighborhood Reconstruction (ONR) is a hyperspectral band selection method that reduces the dimensionality of hyperspectral images while preserving critical spectral information. It worcks has each band in the dataset is reconstructed using its neighboring selected bands. ONR minimizes reconstruction error. [3] The goal is to select a subset of bands that can reconstruct the entire dataset with minimal error.

2.2 Hyperspectral Imaging Target Detection

After applying dimensionality reduction methods, an hyperspectral image is obtained with relevant data available for further analysis. In this section, the available algorithms for target detection are examined and categorized based on the types of technologies they employ [4].

This work considers HSI algorithms for target detection, including hyperspectral imaging object detection, target detection, anomaly detection, and salient object detection. While the differences between these areas may be relevant, they all produce valid results for proposed work.

For the following algorithms, let x_i represent a ”pixel” from a HSI, an array of n dimensions, and H and W denote the height and width of a HSI, as defined previously.

In the field of HSITD, significant advancements have been made in pixel-by-pixel classification methods. More recently, however, there has been increased focus on architectures that are not based solely on pixel-by-pixel classification.

2.2.1 Spectral Angle Mapper

Spectral Angle based methods take advantage of the calculation of Spectral Angle Distance (SAD) between pixels on an image. The SAM(Spectral Angle Mapper) methodology compares each pixel with a target spectrum, doing this to all the pixels in the image and classifying each pixel based on a threshold [4]. The algorithm measures the spectral similarity between two pixels by calculating the

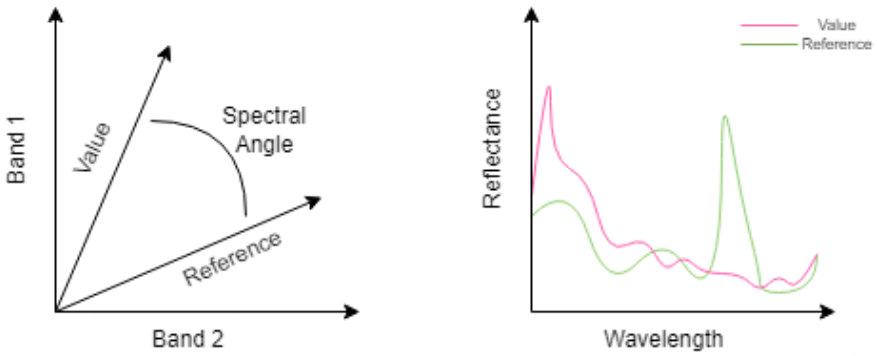

Figure 2.2: SAM

angle between them, treating each spectrum as a vector in a space where the number of dimensions corresponds to the number of bands (See Figure 2.2).

In hyperspectral imaging, SAD is a widely used similarity metric for comparing the spectral signatures of pixels in hyperspectral data. Each pixel in a HSI is represented as a vector, where each value corresponds to a specific wavelength. SAD calculates the angle between these spectral vectors in a high-dimensional space. The vectors are projected into a multidimensional space, and the angle between them is calculated to compute the spectral angle. This angle measures the difference between two spectral signatures. The Spectral Angle is defined as:

$$\theta = \cos^{-1}\left(\frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\|\|\mathbf{x}_2\|}\right), \tag{2.1}$$

where θ is the angle between the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Here, $\mathbf{x}_1$ and $\mathbf{x}_2$ represent the spectral vectors of two pixels in the hyperspectral image, and $\|\cdot\|$ denotes the vector norm.

Spectral Angle Distance is particularly useful in hyperspectral target detection and classification tasks, as it is robust to variations in illumination and focuses on the relative spectral shape (Wavelength present) rather than magnitude (Wavelength strength).

2.2.2 Constrained Energy Minimization (CEM)

The main objective of the Constrained Energy Minimization (CEM) technique for HSITD, is to increase the visibility of a particular target in an image while lowering background interference. In order to do this, CEM applies a filter that minimizes the contributions from other elements (background) and maximizes the gain for the target's distinct spectral signature.

Assume the observed hyperspectral data matrix is represented as:

$$\mathbf{I} = [\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N], \tag{2.2}$$

where $\mathbf{X}_i = [X_{i1}, X_{i2}, \ldots, X_{iB}]^T$ represents the spectral signature of the i-th pixel, N is the total number of pixels, and B is the number of spectral bands. Suppose the desired target signature is denoted by T. The objective of CEM is to design a filter $\mathbf{f} = [f_1, f_2, \ldots, f_B]$ that minimizes the background while maintaining the constraint $\mathbf{f} = 1$.

This technique emphasizes that particular signature while suppressing the background, requiring prior knowledge of the target's spectral properties to tune the filter appropriately for it to function as intended. CEM is a pixel-based spectral detector, producing a score for each pixel in the image, this score represents the probability that the pixel belongs to the target of interest, without taking spatial information into account.

One of the key advantages of CEM is its adaptability to different backgrounds. Since the objective is to emphasize a specific target signature, it is highly effective in scenarios where the background can vary significantly [4], [5].

2.2.3 Kernel-Based Methods

Most of the previously discussed algorithms have a kernel version since kernel-based methods extend the traditional methods to higher dimensions (kernel trick). This simplifies computations, as the inner product in the feature space replaces mapping the points to this higher dimension field. This extension enables classical algorithms to leverage high-order information, improving their effectiveness in target detection tasks. Common algorithms such as KMF, KASP, and KSAM have been successfully adapted using kernel methods, illustrating their flexibility and enhanced performance in hyperspectral imaging applications.

For example, the CEM algorithm referred above can be extended to a kernel version of it, K-CEM. Kernel Constrained Energy Minimization (K-CEM) extends the traditional CEM approach leveraging the kernel trick to handle non-linear data relationships, making it particularly effective in cases where the linear assumptions of standard CEM are insufficient [4], [5]..

By leveraging the kernel trick, K-CEM can be represented in terms of kernel functions, avoiding the explicit computation of the higher dimensional map. This extension of CEM is particularly useful in hyperspectral applications involving complex data distributions, where non-linear relationships between target and background are prominent.

2.3 Deep Learning Methods

Deep learning techniques are also used for hyperspectral image classification. This methods usualy are more refined and can beetter extrat usefull information from hypersctral images. In this sections we go deeper in some archtecture that have been developed for HSITD.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are fundamental algorithms in machine learning and deep learning applications, they process data matrices like images. CNNs are composed of various layers each with specific functions: convolutional, downsampling, pooling, strided convolution, upsampling, and activation function.

The convolution layer is an essential part of a CNN, designed to extract information from data. It applies the convolution operation between the data and a kernel (small matrix of weights). The convolutional operation can be expressed as

$$S_{(i,j)} = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \tag{2.3}$$

where $S_{(i,j)}$ is the result of the operation at a given (i,j) pixel from the input image $I_{m,n}$ and Kernel K.

The downsampling layer reduces the spatial resolution while retaining the most important information. It may also increase the channel size (C), in some cases. Downsampling helps reduce computational complexity and capture high-level, abstract features in CNNs. There are various methods for downsampling, such as Max Pooling, Average Pooling, and Strided Convolution.

Max pooling calculates the maximum value of the data within a specified window (W). The resulting maximum value is stored, and the operation is applied to all the pixels in the image, effectively reducing the size of the image by a factor of W in height and width. The formula for max pooling is:

$$\text{Output}(i,j) = \max_{m,n} \text{Input}(s \cdot i + m, s \cdot j + n), \quad 0 \leq m, n < k, \tag{2.4}$$

where s is the stride (step size for the window), k is the size of the pooling window/kernel,input(i,j) is the input value at position (i,j) and Output(i,j) is the output value at position (i,j).

Average pooling operates similarly to max pooling but instead of taking the maximum value, it calculates the average of the values inside the window. The formula for average pooling is:

$$\text{Output}(i,j) = \frac{1}{k \times k} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \text{Input}(s \cdot i + m, s \cdot j + n), \tag{2.5}$$

where the parameters s and k are the same as in max pooling.

Strided convolution applies a convolution operation with a stride greater than 1, skipping pixels to reduce the size of the output. It combines feature extraction and downsampling into a single step. The formula for strided convolution is:

$$\text{Output}(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \text{Input}(s \cdot i + m, s \cdot j + n) \cdot \text{Kernel}(m,n), \tag{2.6}$$

These downsampling methods are essential in convolutional neural networks to reduce the spatial dimensions of data while preserving relevant features, thereby improving computational efficiency and enabling the extraction of higher-level representations.

Another common type of layer found in CNNs is the upsampling layer. This layer allows the network to go from low-resolution data to high-resolution data and is crucial for many complex CNN architectures. Upsampling layers are often used to produce workable outputs by upscaling data that has been downsampled in previous layers. There are several methods to implement an upsampling layer, including:

1. Nearest Neighbor Interpolation - Nearest neighbor interpolation takes the value of the nearest pixel and copies it to fill in the gaps in the upsampled data. This is a simple and computationally efficient method for increasing the resolution of data;

2. Bilinear and Bicubic Interpolation - Bilinear interpolation calculates the value of new pixels by linearly interpolating between the values of nearby pixels along both the x and y axes. For bicubic interpolation, additional surrounding pixels are used to calculate the value of new pixels, leading to smoother results compared to bilinear interpolation. The general formula for interpolation is:

$$\text{Output}(i,j) = \sum_{p=0}^{1}\sum_{q=0}^{1} w_{p,q} \cdot \text{Input}(x_p, y_q), \tag{2.7}$$

   where $w_{p,q}$ is the weight coefficients determined by the interpolation method, Input(x_p, y_q) is the input pixel values at the nearby coordinates (x_p, y_q) and Output(i,j) is the interpolated pixel value at position (i,j).

Upsampling layers are essential for tasks like image segmentation, where reconstructing high-resolution outputs from low-resolution inputs is necessary. The choice of interpolation method depends on the application, with nearest neighbor being faster but less smooth, while bilinear and bicubic interpolation offer higher-quality results at the cost of increased computational complexity.

Activation functions are used to activate neurons and are often applied at the end of layers. They usually constrain the output values to a specific range, often between 0 and 1, allowing the network to model complex, non-linear relationships.

The *Softmax* activation function transforms the values of a vector into a probability distribution, making it ideal for multi-class classification and target detection:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The *ReLU (Rectified Linear Unit)* activation function activates each value by returning the maximum between the input value and zero, introducing non-linearity to the network

$$\text{ReLU}(x) = \max(0, x)$$

Batch normalization is used to normalize the inputs of each layer, improving the stability and performance of CNNs. It achieves this by ensuring that the inputs to each layer have a mean of zero and a standard deviation of one. Batch Norm is calculated as follows:

$$\hat{x}_i = \frac{x_i - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}$$

where x_i is the input value of the i-th neuron in the mini-batch, μ_{batch} is the mean of the inputs in the current mini-batch, σ^2_{batch} is the variance of the inputs in the mini-batch and ϵ is a small constant added to avoid division by zero.

After normalization, the output is scaled and shifted using learnable parameters γ (scale) and β (shift):

2.3.2 Attention Module

The attention mechanism was designed to enhance the performance of neural networks by selectively focusing on the most relevant parts of the input data. In the context of hyperspectral image target detection, attention mechanisms help to emphasize critical spectral or spatial features while suppressing irrelevant information. The attention module computes a weighted representation of the input features, allowing the network to "attend" to significant regions or bands based on their importance to the task [6].

To better understand Attention, let's consider an example with a small HSI, X_1 with dimensions $3 \times 3 \times 4$, representing a 9-pixel image with 4 spectral channels (See Figure 2.3). The first step is to flatten the image into a 9×4 matrix, where each row corresponds to a pixel, and each column represents a channel. For simplicity, in this example, we will apply attention in 2D. However, in practice, higher dimensions are typically used.

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \begin{bmatrix} 0.93 & 0.423 & 0.33 \\ 0.12 & 0.03 & 0.56 \\ 0.27 & 0.7 & 0.09 \end{bmatrix} \end{bmatrix}$$

Figure 2.3: Example of a small hyperspectral image matrix [X_1].

The first step is to compute the query matrix Q, the key matrix K, and the value matrix V as follows:

$$Q = X_1 W^Q, \quad K = X_1 W^K, \quad V = X_1 W^V, \tag{2.8}$$

where W^Q, W^K, and W^V are learnable and adjustable parameters, and I represents the input im-

age. The dimensions of these matrices depend on the mapping being performed, based on the input dimensions and the dimensions being projected into. In this example, the input image has 4 channels (a 4-dimensional vector per pixel), and it is mapped into a 2-dimensional space. Therefore, the weight matrices W^Q, W^K, and W^V will each have a shape of 4×2. After computing Q, K, and V, we obtain three 9×2 matrices:

- Q (Query) - contains the features the model will focus on;
- K (Key) - encodes the context or global information of the input;
- V (Value) - contains the feature values corresponding to the keys.

The general formula for the attention mechanism is given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.9}$$

where Q is the query matrix, K is the key matrix, V is the value matrix, and d_k is the dimension of the key vectors.

So now we multiply Q by K^T (a 9×2 matrix by a 2×9 matrix), resulting in a 9×9 matrix. Each row in Q represents a pixel, and each column in K represents a pixel from the context. This multiplication can be interpreted as calculating the similarity score between each pair of pixels. For example:

$$Q = \begin{bmatrix} -0.25 & 0.90 \\ 0.46 & 0.20 \\ -0.69 & -0.69 \\ -0.88 & 0.73 \\ 0.20 & 0.42 \\ -0.96 & -0.94 \\ 0.66 & -0.58 \\ -0.64 & 0.05 \\ -0.21 & -0.66 \end{bmatrix} \quad K^T = \begin{bmatrix} -0.92 & 0.73 & -0.31 & -0.73 & -0.57 & -0.90 & -0.58 & 0.33 & -0.90 \\ 0.71 & -0.32 & 0.93 & -0.54 & -0.97 & -0.69 & -0.32 & 0.45 & -0.49 \end{bmatrix}$$

$$QK^T = \begin{bmatrix} 0.87 & -0.47 & 0.91 & -0.30 & -0.73 & -0.39 & -0.15 & 0.32 & -0.22 \\ -0.29 & 0.28 & 0.04 & -0.44 & -0.46 & -0.56 & -0.33 & 0.24 & -0.51 \\ 0.15 & -0.28 & -0.42 & 0.87 & 1.06 & 1.10 & 0.62 & -0.53 & 0.96 \\ 1.33 & -0.88 & 0.95 & 0.25 & -0.20 & 0.29 & 0.27 & 0.04 & 0.43 \\ 0.11 & 0.02 & 0.32 & -0.37 & -0.52 & -0.47 & -0.25 & 0.25 & -0.39 \\ 0.22 & -0.40 & -0.57 & 1.20 & 1.46 & 1.51 & 0.86 & -0.73 & 1.32 \\ -1.02 & 0.67 & -0.74 & -0.17 & 0.18 & -0.20 & -0.20 & -0.04 & -0.31 \\ 0.62 & -0.48 & 0.24 & 0.44 & 0.32 & 0.54 & 0.35 & -0.19 & 0.55 \\ -0.27 & 0.05 & -0.54 & 0.51 & 0.76 & 0.64 & 0.33 & -0.36 & 0.51 \end{bmatrix}$$

This matrix represents how similar each pixel vector is to every other pixel vector in the input image. The closer two vectors are, the higher their similarity score will be. Figure 2.4 illustrates the similarity between different pixels in this 2D space.

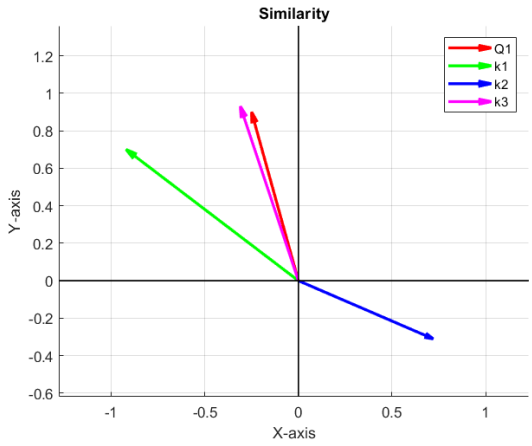

Figure 2.4: similarity between pixel 1 (Q1) and others pixels (K1,K2 and k3)

Next, all values of this matrix are divided by the square root of the dimension of the key vectors (in this case, $\sqrt{2}$). This scaling step is necessary to normalize the values and prevent them from growing too large, especially in high-dimensional spaces. At the end of this step, we are left with a matrix that contains the similarity scores between pixels. Each row shows how similar the corresponding query pixel is to all other pixels in the image.

Finally, the softmax function is applied to each row of this matrix. The softmax function converts the similarity scores into percentages, effectively turning them into a probability distribution. This allows interpreting the values as the relative importance of each pixel to the corresponding query pixel. We now have:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{bmatrix} 0.19 & 0.07 & 0.20 & 0.08 & 0.06 & 0.08 & 0.09 & 0.13 & 0.09 \\ 0.10 & 0.15 & 0.13 & 0.09 & 0.09 & 0.09 & 0.10 & 0.15 & 0.09 \\ 0.09 & 0.06 & 0.06 & 0.14 & 0.16 & 0.17 & 0.12 & 0.05 & 0.15 \\ 0.21 & 0.04 & 0.16 & 0.10 & 0.07 & 0.10 & 0.10 & 0.09 & 0.11 \\ 0.13 & 0.12 & 0.15 & 0.09 & 0.08 & 0.09 & 0.10 & 0.14 & 0.09 \\ 0.07 & 0.05 & 0.04 & 0.15 & 0.18 & 0.19 & 0.12 & 0.04 & 0.16 \\ 0.06 & 0.20 & 0.07 & 0.11 & 0.14 & 0.11 & 0.11 & 0.12 & 0.10 \\ 0.14 & 0.06 & 0.11 & 0.12 & 0.11 & 0.13 & 0.11 & 0.08 & 0.13 \\ 0.08 & 0.10 & 0.06 & 0.13 & 0.16 & 0.15 & 0.12 & 0.07 & 0.13 \end{bmatrix}$$

The attention weight between Q_1 (first row) and K_3 (third collum) is 20%, indicating that the first pixel assigns 20% of its focus to the third pixel, the attention weight from Q_3 (third line) to K_1(first column) is 9%. Next, we multiply the resulting 9×9 matrix by the 9×2 V matrix. This multiplication can be interpreted as a weighted average since the softmax ensures that the sum of each row in the 9×9 matrix equals 1, and each percentage represents how much weight to give to the corresponding value vector in V. The result is a 9×2 matrix that represents the final attention computation. If we want to return to the original 4-dimensional representation, we need to apply a final linear transformation using a 2×4 matrix. This projects the 9×2 output back into a 9×4 matrix, matching the dimensions of the input

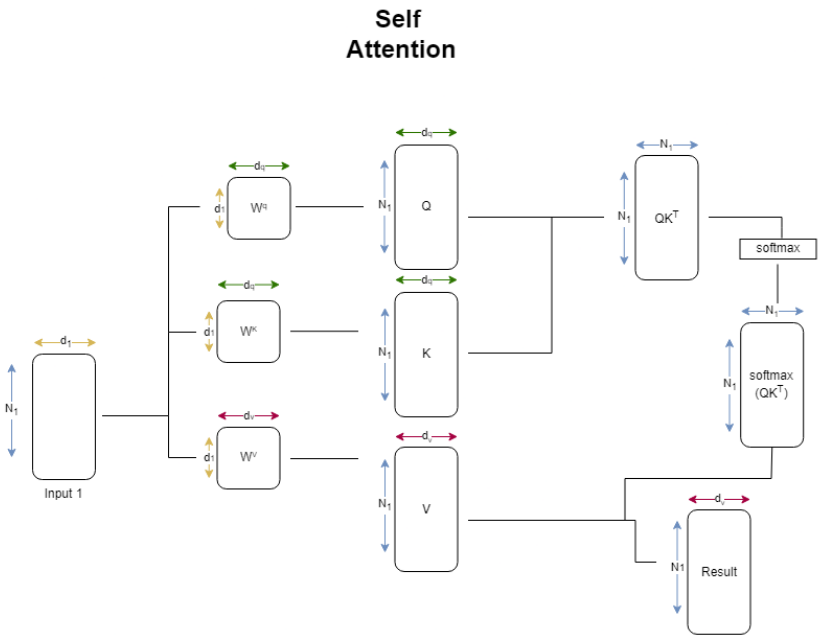

Figure 2.5: Self-attention mechanism.

image. The attention mechanism computes a compatibility score between the query and key vectors with a dot product, normalizes these scores using the softmax function, and then applies the scores to weight the value vectors. This results in an attention-weighted representation of the input data.

This process is specific to Self Attention (SA) (See Figure 2.5, where the Q, K, and V matrices are all derived from the same input (hence the term ”self”).

In addition to SA, there is also Cross Attention (CA), where the K and V matrices come from a different input than the Q matrix. For example, while Q may be derived from an image, K and V can be provided by a different source, such as textual data, another image, or any other context vector. This enables the model to attend to different types of data simultaneously, enhancing its ability to incorporate external information into its predictions.

Another important concept to mention is multi-head attention. The process explained so far describes the computation for a single head, but in practice, most attention-based models use multiple heads in parallel. This allows the model to focus on different aspects of the input data simultaneously. Each head performs the same attention computation independently, but with different weight matrices (W^Q, W^K, and W^V), enabling the model to capture a variety of relationships within the data.

The computations involved in SA are highly dependent on the dimensions of the input. Self-attention has quadratic time complexity with respect to the input size. In this case, the number of pixels is 9, resulting in a complexity of 9^2. For large data applications, such as hyperspectral images, this quadratic complexity becomes a problem because the vectors from the pixels also grow. To address this issue,

several variations of attention mechanisms have been developed to improve efficiency and reduce computational costs.

One of the most effective is the Neighborhood Attention (NA) mechanism [7], which uses a sliding window self-attention mechanism to localize each pixel's attention span to its nearest neighborhood. This means that, instead of attending to every pixel in the image, each pixel only attends to the pixels within a defined neighborhood window (See Figure 2.6). When the neighborhood size is maximized to cover the entire input, NA is equivalent to the regular self-attention.

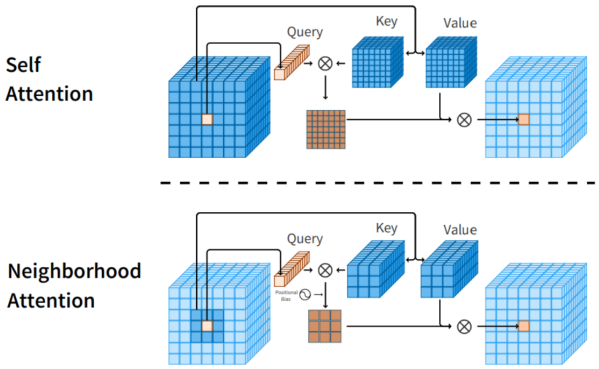

Figure 2.6: Example of the Neighborhood Attention mechanism.

The main advantage of Neighborhood Attention, compared to self-attention, is that it can be seamlessly integrated into existing self-attention architectures. By substituting SA with NA and setting the neighborhood size equal to the input size, we can achieve the same results as traditional self-attention. Then, by gradually reducing the kernel size of the neighborhood, the attention mechanism runs with more localized information.

2.4 Deep-Learning for Hyperspectral Image Target Detection

Deep learning is increasingly being used for target detection in Hyperspectral images. This section explores some of these deep learning methods and goes into detail on three algorithms that were considered for the proposed work.

2.4.1 Generic Hyperspectral Detection with CNNs

As discussed earlier, hyperspectral image target location has evolved significantly, particularly in terms of pixel-by-pixel classification. Similar to RGB images, where we classify targets as "target" or "no target" for every pixel (resulting in a segmentation task), this approach can also be applied to hyperspectral

images. For hyperspectral images, extensive research has been conducted in this field using 1D, 2D, and 3D CNNs [8], [9]. Illustrative examples of these operations can be seen in Figure 2.7.

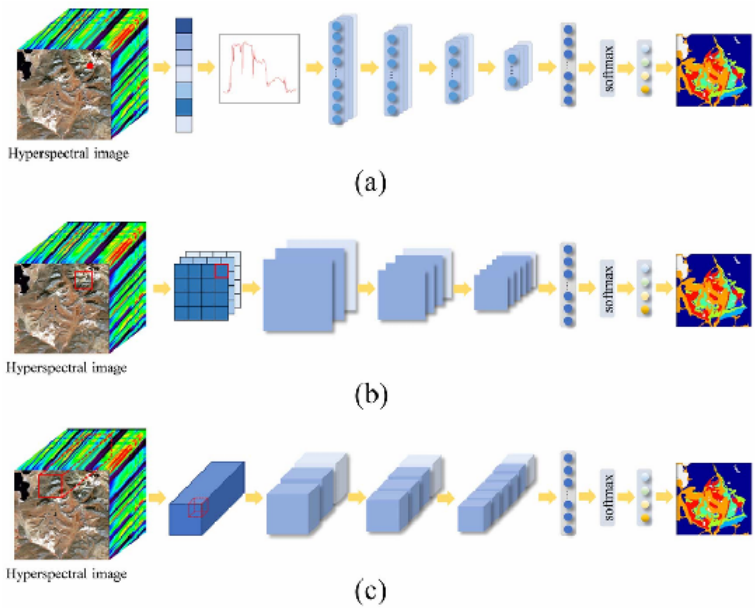

Figure 2.7: (a)1D,(b)2D and (c)3D CNN image taken from Evaluation [9]

A 1D CNN for hyperspectral pixel classification receives a single vector $\mathbf{x}_i$ as input, corresponding to the spectral signature of a single pixel. The convolution is computed along the spectral dimension of this pixel alone, ignoring the spatial component of the hyperspectral image. By repeating this operation for every pixel in the image, the CNN performs pixel-wise classification. The result is an output image with the same height (H) and width (W) as the original image but with a single channel representing the classification labels for each pixel.

The classification results can be improved using information from neighbor pixels. In this case, a 2D CNN is used that incorporates spatial information from the image by using a small kernel around the pixel of interest. This kernel processes the neighboring pixels in a 2D window to capture spatial patterns and relationships. For hyperspectral images, 2D CNNs work by treating each spectral band as a separate channel, similar to RGB channels in traditional images. The convolution is then applied over these spatial dimensions, effectively combining spectral and spatial features for target detection. This makes 2D CNN more effective in detecting targets with spatially correlated features compared to 1D CNN, which only focus on spectral information.

A 3D CNN utilizes the full information available in a hyperspectral image, capturing both spectral and spatial features simultaneously at the cost of more computations. The convolutional kernel in a 3D CNN operates not only across the spatial dimensions but also along the spectral dimension. This allows the

network to learn complex spatial-spectral relationships, making 3D CNNs particularly powerful for hyperspectral target detection. For example, a 3D CNN can distinguish subtle spectral variations in different materials while simultaneously recognizing spatial patterns, leading to highly accurate classification and detection.

2.4.2 Spectrum-driven Mixed-frequency Network - SMN

The Spectrum-driven Mixed-frequency Network (SMN) model [10] includes attention modules(2.3.2) for target detection and image pre-processing and post-processing techniques ([1]) (See Figure 2.8)

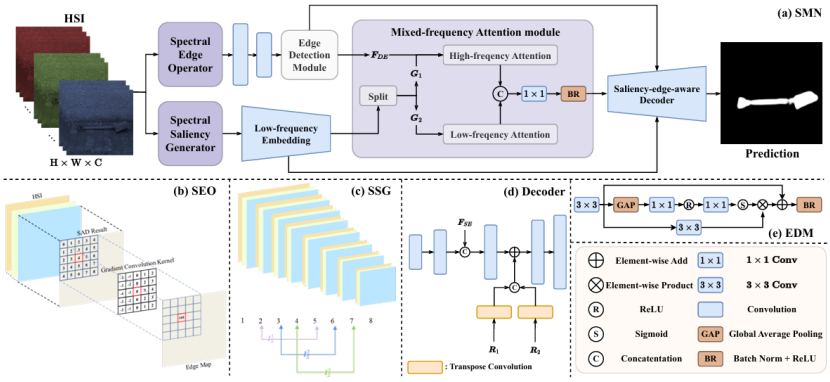

Figure 2.8: SMN_architecture [11]

SMN introduces a new pre-processing task to ensure the maximum of information from the available bands in the HSI. This pre-processing step includes two methods: the Spectral Saliency Generator (SSG) and the Spectral Edge Operator (SEO) The SSG is a pre-processing unit responsible for extracting saliency information from the original HSI. To better understand the details of the SSG, the pseudocode of the process was developed (See Algorithm 2.1).

SSG operates by creating an 8 layer Gaussian pyramid and then computing the comparison between pairs of Gaussian layers (2 with 5, 3 with 6, and 5 with 7) The comparison is executed via the calculation of the SAD between spectral vectors that are derived from the c-th (center) and s-th (surround) layers (2 and 5, for example). This is done three times (one for each pair of layers).The result is a Saliency_Map array, with three $H \times W \times 1$ images each with a different saliency information.

[1] Code available in https://github.com/laprf/SMN/tree/master

Algorithm 2.1: Pseudocode of the Spectral Saliency Generator

Result: `saliency_maps`: A list of saliency maps $\{I_S^k\}_{k=1}^3$, each of shape $H \times W \times 1$.

Pyramid[8];

Pyramid[0] ← HSI;

for i=1;i¡8;i++ **do**

  Apply Gaussian downsampling to the previous layer in pyramid[i-1];

  Save new layer in pyramid[i];

end

for `center_layer` c in [2,3,4] **do**

  Set `surround_layer_index` $s = c + 3$;

  Retrieve the `center_layer` v_c from `pyramid`[c];

  Retrieve the `surround_layer` v_s from `pyramid`[s];

  Upsample `surround_layer` v_s to match the spatial resolution of `center_layer` v_c;

  Initialize an empty saliency map I_S of the same size as the `center_layer`;

  foreach `pixel location` (i, j) in the `center_layer` **do**

    Extract the spectral vector $X1_i(i, j)$ from the `center_layer`;

    Extract the spectral vector $X2_i(i, j)$ from the upsampled `surround_layer`;

    Compute the Spectral Angular Distance (SAD)

    Store the SAD value $I_S(i, j)$ in the saliency map I_S;

  end

  Upsample the saliency map I_S to the original resolution $H \times W$;

  Append the upsampled saliency map I_S to `saliency_maps`;

end

SEO is a pre-processing unit like the SSG, but instead of Saliency information, it aims to retrieve critical edge information from the image. This information will be useful in later stages of the SMN model. Similar to what was done for the SSG, a pseudo code of SEO was developed to better understand the inner workings of this module (See Algorithm 2.2).

SEO extracts Spectral information by computing the gradient of the SAD in the vicinity of each pixel. Specifically, for a pixel (i,j) on the HSI, and any other pixel (p,q) within its neighborhood, SAD can be computed, resulting in a local spectral similarity map, to compute the value of the final edge image IE at (i,j), gradient convolution is applied to neighborhood map resulting in a single value.

The pre-processing unit generates 3 maps containing different edge information, one map for each neighborhood size (5,15,25).

Algorithm 2.2: Pseudocode of the Spectral Edge Operator

Result: `Edge maps`: A list of edge maps $\{I_S^k\}_{k=1}^3$, each of shape H × W × 1

Edge_maps;

Set `kernels` = [5, 15, 25];

for n_size in `kernels` **do**

  Set `half_size` = $n\_size//2$;

  Initialize a 4D array `sad_map` of shape $H \times W \times n\_size \times n\_size$;

  Calculate Local Spectral Similarity Map (SAD);

  foreach `pixel location` (i, j) in the HSI **do**

    Set `center_vector` to the spectral vector at (i, j);

    foreach `neighboring pixel` (p, q) within the neighborhood of size $n\_size \times n\_size$ **do**

      Set `neighbor_vector` to the spectral vector at (p, q);

      Compute the Spectral Angular Distance (SAD) as: Store $M(p, q)$ in `sad_map`[i, j, p, q];

    end

  end

  Step 2: Compute Edge Map using Gradients of the SAD Map;

  Initialize a 2D matrix `edge_map` of shape $H \times W$;

  foreach `pixel location` (i, j) in the HSI **do**

    Extract the local SAD map `local_sad_map` for the neighborhood around (i, j);

    Compute the x-gradient $G_x * M$ by convolving `local_sad_map` with G_x;

    Compute the y-gradient $G_y * M$ by convolving `local_sad_map` with G_y;

    Calculate the edge strength at (i, j) as:

$$I_E(i, j) = |G_x * M| + |G_y * M|$$

    Store $I_E(i, j)$ in `edge_map`[i, j];

  end

  Append `edge_map` to `Edge_maps`;

end

The maps computed by the SEO will go through two downsampling (see section 2.3.1) layers implemented with strided convolutions. Edge Detection Module (EDM) is responsible for extracting relevant information from the edge maps. The downsampling block before EDM takes as input the concatenated *Spectral Edge* images:

$$F_{SE} = f_D\left([I_E^1, I_E^2, I_E^3]\right),$$

where $f_D(\cdot)$ represents the downsampling operation. It is implemented using two Conv3×3 layers with a stride of 2, followed by a batch normalization layer, and the ReLU activation function.

The output of this downsampling block is a shallow edge feature F_{SE}. This is then passed through

the EDM to produce the deep edge feature F_{DE}:

$$F_{DE} = f_E(F_{SE}),$$

where $f_E(\cdot)$is the operation performed by the EDM, which consists of a shallow CNN, (see section2.3.1) with 3x3 convolution layers and global average polling (GAP) and Batch Norm (BR). The deep edge

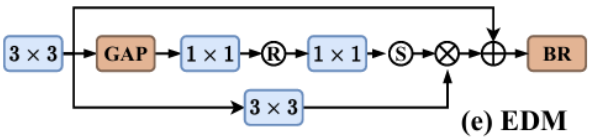

Figure 2.9: Edge detection module architecture

feature F_{DE} has dimensions $\frac{H}{16} \times \frac{W}{16} \times 256$, where H and W are the height and width of the original input image, and 256 is the number of channels of the edge feature. To improve the quality of the edge features, F_{DE} is transformed into an edge image M_E using the operation: $M_E = f_{\text{CU}}(F_{DE})$, where $f_{\text{CU}}(\cdot)$ is implemented using, a conv1 $\times$ 1 convolutional layer and an upsampling layer.

The resulting edge map M_E is a single-channel grayscale image with the same spatial dimensions as the original image.

The Low-Frequency Embedding module receives the Spectral Saliency images (SSG) I_S^1, I_S^2, and I_S^3 and concatenates them into

$$F_S = \left[I_S^1, I_S^2, I_S^3\right],$$

Concatenating the Spectral Saliency images, rather than feeding them into the network one by one, reduces the number of computations, improving inference speed.

The concatenated features F_S are then transformed into a deep saliency feature F_{DS} through the low-frequency embedding process, $F_{DS} = f_L(F_S)$, where $f_L(\cdot)$ represents the low-frequency embedding operation. The low-frequency backbone can be one of three models (Resnet18, PVTv2-b1, swin_t) each differing in the number of parameters and complexity (number of FLOPS) according to Table 2.1.

Metrics	FLOPs (G)	#Params (M)
SMN-R	**14.58**	7.27
SMN-S	17.23	16.87
SMN-P	14.76	10.23

Table 2.1: Performance comparison of different models for the low-frequency backbone of SMN.

The Low-frequency Embedding process generates:

$$\mathbf{R} = \{\mathbf{R}_i \mid i = 1, 2, 3\},$$

Each stage captures progressively smaller input regions: $\frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ of the input feature, the deep saliency feature F_{DS} is obtained from the last stage of the low-frequency embedding, $\mathbf{R}_3$, with dimensions $F_{DS} = \mathbf{R}_3 \in \mathbb{R}^{\frac{H}{16} \times \frac{W}{16} \times 256}$. Additionally, the shallow saliency information from the first and second stages of embedding, $\mathbf{R_1}$, is retained for later use in the decoder stage.

The Mixed-frequency Attention (MA) module is the core of the model and combines the inputs from the Edge Detection Module and the Low-Frequency Embedding module. It includes two main components: High-Frequency Attention, which captures relationships between high-frequency edge features, and Low-Frequency Attention, which applies a self-attention mechanism to refine low-frequency saliency features.

The deep saliency feature F_{DS}, with channel dimension 256, is divided into two groups along the channel dimension:

$$\mathbf{G}_1 = F_{DS}(:, :, 0 : \lfloor C_S/2 \rfloor), \quad \mathbf{G}_2 = F_{DS}(:, :, \lfloor C_S/2 \rfloor : C_S),$$

where $\mathbf{G}_1$ is the first half of the deep saliency feature and $\mathbf{G}_2$ the other half. These groups are fed into the two heads of the Attention module. The high-frequency attention head employs a Neighborhood Attention Mechanism NA. The NAM operation is a variation of the Attention module as discussed in section 2.3.2. Instead of using equation 2.9 to compute the attention, it considers:

$$f_{\text{NAM}}(\mathbf{X}, \mathbf{Y}) = \sigma \left(\mathbf{Q}_{i,j} \mathbf{K}^T_{\rho(i,j)} + B_{i,j} \right) \mathbf{V}_{\rho(i,j)},$$

where $\mathbf{X}$ and $\mathbf{Y}$, $\rho(i, j)$ represents the local neighborhood around pixel, (i, j),$B_{i,j}$ denotes the relative positional bias, and $\sigma(\cdot)$ is the softmax function. For this approach, they adapted NA to compute CA.

The cross-attention between the deep edge feature F_{DE} and the first group of saliency features $\mathbf{G}_1$ is $F_H = f_{\text{NAM}}(F_{DE}, \mathbf{G}_1)$ and F_H is the refined saliency feature constrained by edge information. The low-frequency attention head processes $\mathbf{G}_2$ through a self-attention mechanism to refine the saliency features $F_L = f_{\text{NAM}}(\mathbf{G}_2, \mathbf{G}_2)$, F_L represents the refined low-frequency saliency feature. The outputs from the high-frequency head (F_H) and low-frequency head (F_L) are concatenated along the feature dimension to form the mixed-frequency output F_{out}: $F_{\text{out}} = \delta \left(f_{\text{c}} \left([F_H, F_L] \right) \right)$, where $\delta(\cdot)$ is the ReLU activation function and $f_c(\cdot)$ represents a 1×1 convolution.

The last module is the saliency-edge-aware decoder which employs a cascading structure of convolutional layers. This makes a gradual upscaling of the mixed-frequency feature while combining of shallow features from previous modules.

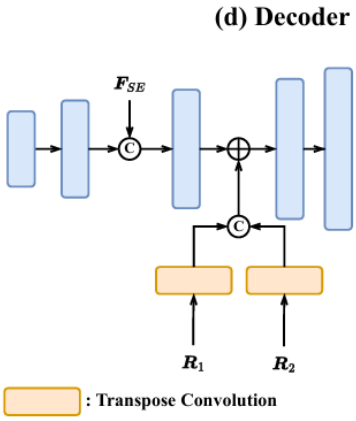

Figure 2.10: Decoder

The decoder consists of five convolutional layers (input feature maps $\mathbf{D}_{\text{in}} \in \{D^i_{\text{in}} \mid i = 1, 2, 3, 4, 5\}$ and output feature maps $\mathbf{D}_{\text{out}} \in \{D^i_{\text{out}} \mid i = 1, 2, 3, 4, 5\}$, each accompanied by Batch normalization, ReLU activation function, and an interpolation operation.

To preserve shallow information, the shallow edge feature $F_{SE}$is concatenated with the output of the second layer D^2_{out}, forming the input for the third layer$D^3_{\text{in}} = \left[D^2_{\text{out}}, F_{SE}\right]$. The saliency features $\mathbf{R}_1$ and $\mathbf{R}_2$ are separately upsampled to match the spatial dimensions. These outputs are concatenated along the channel dimension and added to the output of the third layer D^3_{out}, forming the input for the fourth layer $D^4_{\text{in}} = D^3_{\text{out}} + [f_{\text{tc1}}(\mathbf{R}_1), f_{\text{tc2}}(\mathbf{R}_2)]$, where $f_{\text{tc1}}(\cdot)$ and $f_{\text{tc2}}(\cdot)$ represent transposed convolutional layers.

2.4.3 Object Detection via Unified Spectral-Spatial Feature Aggregation

Similar to SMN, the work [12] considers also a pre-processing step of the HSI, followed by a deep-learning architecture. The main difference is in the division of the spectral and spatial information. While the SMN divides the information into Edge and Saliency, this approach focus on the division of spacial and spectral information.

This method is characterized for having 2 backbones (See Figure 2.11), the upper one processes the spectral information and the bottom one the spatial information, at given stages in this 2-backbone dataflow there are Spatial-Spectral Aggregation modules that combine the data. The output of these modules is then fed into a Detection Head.

Like in the previous approach, this model employs an HSI image pre-processing step. For Spectral information PCA,(see section 2.1.2) is applied and for Spatial information ONR,(see section 2.1.3). The network is composed of five stages, each stage processes these features through a feature extraction module denoted as Stage$_i$, this outputs maps for both spectral and spatial components. Next a Spectral-Spatial Aggregation (SSA) module is placed in the last three stages (between Stage 3 and 4, 4 and 5 and at the end of 5). The SSA module combines the two kinds of features and outputs the aggregated

maps for the spectral and spatial information. The outputs of aggregation are concatenated with the original feature maps. Then they are fed into the next stage.

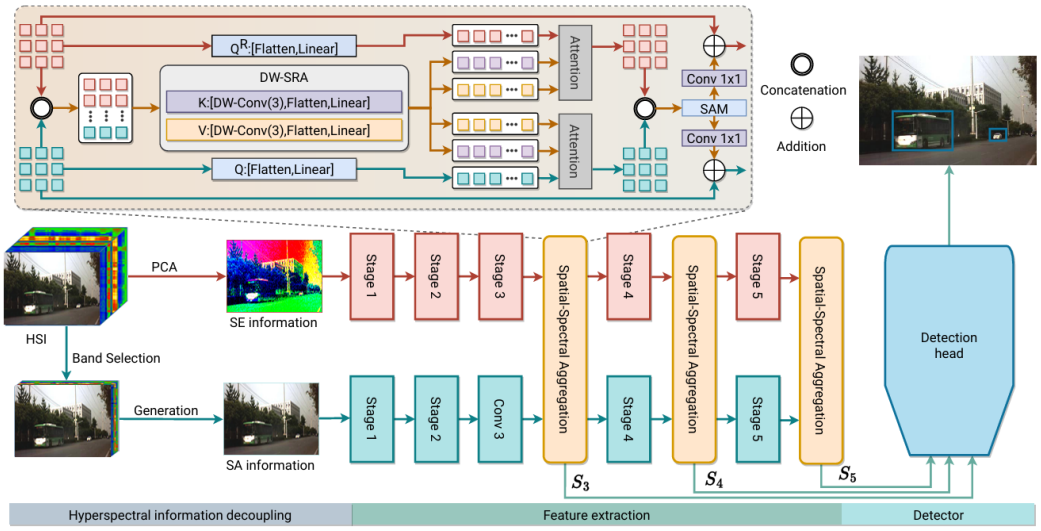

Figure 2.11: Enter Caption

2.5 Hardware Solutions for Hyperspectral Image Target Detection

Field Programmable Gate Arrays (FPGA) offer a highly adaptable platform for accelerating and building digital functions. FPGAs are often used to accelerate existing software applications, allowing for faster results than traditional software implementations. SoC-FPGA combine a traditional FPGA with an integrated processor. Compared to traditional FPGAs, SoC-FPGAss provide a seamless combination of hardware acceleration and software programmability, enabling Hardware-Software Co-design. This integration reduces latency, simplifies system architecture, and enhances the ability to handle complex workloads directly on the device, making them ideal to run complex deep learning models. In HSITD, executing algorithms with significant computational demands is required due to the dimension of the data and the computational complexity of the target detection algorithms described above. The work proposed in [13] implements in FPGA two key computations associated with hyperspectral image target detection: Spectral Angle Distance (SAD) and Constrained Energy Detector, a traditional target detection method in hyperspectral imaging. The goal is to achieve faster computations by applying these operations to hardware. In this architecture, the reference vector X_1 is a 55-band pixel, with each band stored as a 55 array of 16-bit signals, and the pixel vector X_2 is stored in a similar structure.

The work implements three versions of the architecture: pipelined, serial, and parallel, that explore different tradeoffs between performance and hardware resources (See Table 2.2). Essentially, this method applies the Spectral Angle Mapper (SAM) algorithm (see 2.2.1), where the SAD computation

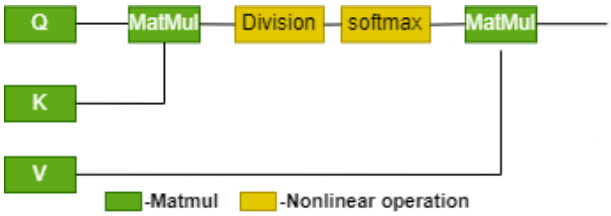

Figure 2.12: Attention

forms the core of the matching process.

Table 2.2: Performance and hardware resource usage for SA implementations. Adapted from [14]

	Performance Metrics			Hardware Resources Usage			
	Max F (MHz)	**Latency**	**time (ms)**	**Registers**	**LUTs**	**DSP48Es**	**Block RAMs**
SA serial	198.50	127	56	4221	2814	6	0
SA parallel	185	77	36.4	15184	7562	168	0
SA pipeline	185	1	0.5	17998	9380	168	23

The attention module (section 2.3.2) presented in SMN (see section 2.4.2) and Spectral-Spatial Feature Aggregation (section2.4.3) offers parallelism opportunities since attention itself is highly dependent on matrix multiplication. Additionally, multi-head attention provides further parallelism, as multiple heads can be computed in parallel. One key optimization is the quantization of the weight matrices. Since the Q, K, and V matrices are stored in memory, quantization is required to reduce memory usage. Quantization involves converting floating-point numbers to a lower-precision format, which reduces the size of the matrices at the cost of some accuracy loss. Most of the work related to attention acceleration has been done in the context of Visual Image Transformers (ViTs). These architectures take advantage of the transformer structure by incorporating multiple attention layers, typically using multi-head attention. However, the models we studied, such as SMN, have only two attention heads, one for cross-attention and one for self-attention. Therefore, we can focus on optimizing single-head. Focusing on batch matrix multiplication and the non-linear softmax function, as shown in Figure 2.6.

2.6 Conclusions

This chapter discussed hyperspectral images, and traditional and deep learning-based methods for hyperspectral image target detection. Traditional methods like Spectral Angle Mapper and Constrained Energy Minimization are computationally simple but fail to capture spatial information, limiting their performance in complex scenarios. Deep learning models such as Convolutional Neural Networks (CNNs), Spectral Mixed Network (SMN), and S2ADet overcome this limitation by leveraging both spectral and spatial features, achieving higher accuracy in most scenarios at the cost of computational complexity.

3

Preliminary Work

As preliminary work, 1d and 2d CNNs were tested and their performance was evaluated.

Figure 3.1 illustrates a segmentation operation of a hyperspectral image from the Pavia University dataset with nine classes. The left figure is the ground truth results of the image, the center image is the segmentation result obtained with 1D CNN, and the right figure is the result obtained with a 2D CNN.

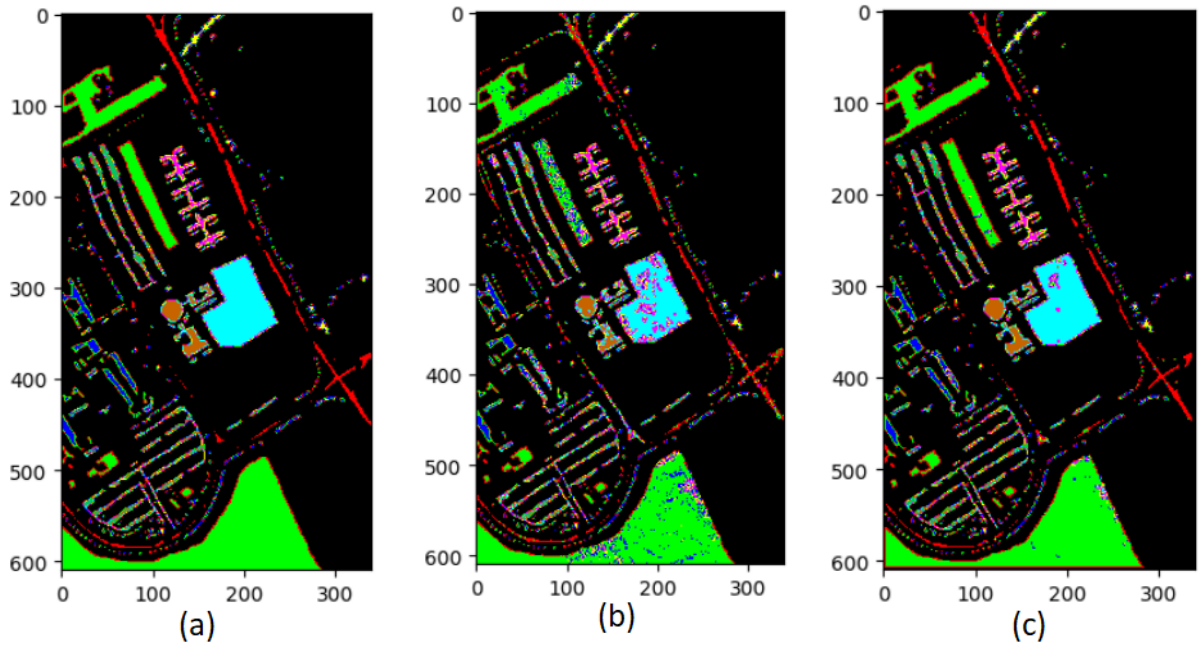

Figure 3.1: (a) Ground Truth from Pavia University dataset[1], (b) 1D CNN Result, (c) 2D CNN Result

As can be observed from the figures, the result obtained from the 2D CNN is more accurate than that obtained with the 1D CNN, as expected.

Phyton code for the pre processing modules SSG and SEO, in the SMN, was developed to fully understand its functionality (see sections 2.4.2). The SMN architecture was trained and tested with the HSOD dataset and the results obtained (See Table 3.1) are very close to those published in the original article.

Table 3.1: Performance of the SMN Method on the HS-SOD Dataset

Method	MAE	S-measure	Max F-measure	AUC	CC
SMN	0.072	0.778	0.680	0.906	0.693

The effect of changes in the SEO kernel size of the SMN model over the final accuracy was evaluated. Figure 3.2 gives a visualization of the output map for different kernel sizes. The first image is the color version of the HSI, the second is the ground Truth and the others are the result of the salient detection considering different sets of kernel sizes in the SEO module.

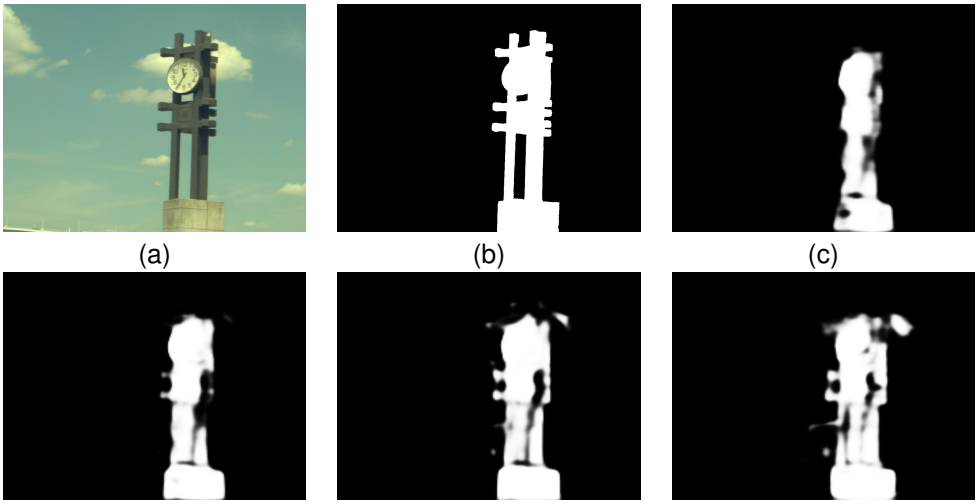

Figure 3.2: (a) Color image, (b) Ground Truth, (c) 3,5,7, (d) 5,9,13, (e) 5,15,25, (f) 25,35,45

The correctness of the saliency improves as the size of the kernels increases and starts degrading for too large kernels.

Table 3.2 shows the effect of the kernel size on the accuracy and complexity of the SMN model.

Table 3.2: SMN Performance and accuracy on HS-SOD Dataset for different kernel sizes of the Spectral Edge Operator

SEO Input	MAE	S-measure	Max-Fmeasure	AUC	CC	GFLOP Count
3,5,7	0.079	0.733	0.622	0.875	0.636	0.5
5,9,13	0.075	0.761	0.666	0.891	0.672	1.6
5,15,25	0.072	0.778	0.680	0.906	0.693	6.2
25,35,45	0.076	0.749	0.643	0.902	0.643	20.3

As shown, kernel size has a profound impact on the computational complexity of the model (FLOP count). Also, for high values of kernel size, the accuracy of the model remains the same, which indicates that, for the model implementation in hardware, we can reduce the size of the kernels to reduce the computations without affecting the overall accuracy. This preliminary study allowed us to verify the model functionality and identification for potential hardware acceleration.

4

Work Proposal

The thesis objective is to implement a hardware/software architecture to execute the SMN model in a SoC-FPGA device. The SMN model will be trained with the HSOD dataset, and will be implemented in a ZCU104 development board, with a Zynq UltraScale+ MPSoC device. The model blocks will first be implemented in C/C++, then the most computation-intensive blocks will be selected for hardware acceleration, a dedicated hardware IP will be developed and the final hardware/software system will be implemented and tested. The hardware components will be designed using the AMD Xilinx Vivado Suite (Vitis HLS, Vivado, and Vitis IDE).

Figure 4.1 illustrates the proposed timeline for the thesis work schedule.

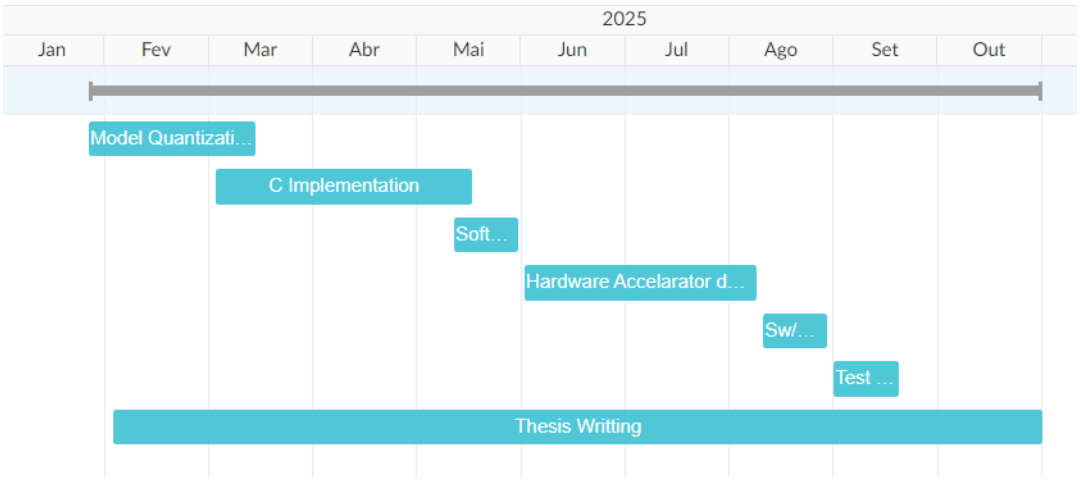

Figure 4.1: Thesis Planning

From the image 4.1 the planning is divided into 7 major tasks:

- Model exploration and quantization in phyton: using available libraries like Brevitas;
- Embedded C/C++ implementation of the model: Implementation of the full model architecture for inference, using the quantization and optimizations developed in the previous task;

- Hardware/Software division: Profiling of the C/C++ implementation with Attention module as the prominent target.
- Hardware Accelerator: Implement the hardware accelerators defined in the previous task using Vitis HLS;
- Hardware/Software System: Integrate the accelerator components with the Software running in the embedded system achieving Hardware/Software implementation;
- Test and Validation: Evaluation of the Hardware/Software implementation;
- Thesis writing: Writing of the thesis explaining in detail the work developed.

Bibliography

[1] N. Imamoglu, Y. Oishi, X. Zhang, G. Ding, Y. Fang, T. Kouyama, and R. Nakamura, "Hyperspectral image dataset for benchmarking on salient object detection," in *2018 Tenth international conference on quality of multimedia experience (qoMEX)*. IEEE, 2018, pp. 1–3.

[2] C. Rodarmel and J. Shan, "Principal component analysis for hyperspectral image classification," *Surveying and Land Information Science*, vol. 62, no. 2, pp. 115–122, 2002.

[3] Q. Wang, F. Zhang, and X. Li, "Hyperspectral band selection via optimal neighborhood reconstruction," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 12, pp. 8465–8476, 2020.

[4] B. Chen, L. Liu, Z. Zou, and Z. Shi, "Target detection in hyperspectral remote sensing image: Current status and challenges," *Remote Sensing*, vol. 15, no. 13, p. 3223, 2023.

[5] C.-I. Chang, "Constrained energy minimization (cem) for hyperspectral target detection: Theory and generalizations," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[6] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[7] A. Hassani, S. Walton, J. Li, S. Li, and H. Shi, "Neighborhood attention transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 6185–6194.

[8] S. Freitas, H. Silva, J. M. Almeida, and E. Silva, "Convolutional neural network target detection in hyperspectral imaging for maritime surveillance," *International Journal of Advanced Robotic Systems*, vol. 16, no. 3, p. 1729881419842991, 2019.

[9] Z. Wang and R. Zuo, "An evaluation of convolutional neural networks for lithological mapping based on hyperspectral images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.

[10] P. Liu, T. Xu, H. Chen, S. Zhou, H. Qin, and J. Li, "Spectrum-driven mixed-frequency network for hyperspectral salient object detection," *IEEE Transactions on Multimedia*, vol. 26, pp. 5296–5310, 2024.

[11] ——, “Spectrum-driven mixed-frequency network for hyperspectral salient object detection,” *IEEE Transactions on Multimedia*, 2023.

[12] X. He, C. Tang, X. Liu, W. Zhang, K. Sun, and J. Xu, “Object detection in hyperspectral image via unified spectral-spatial feature aggregation,” *arXiv preprint arXiv:2306.08370*, 2023.

[13] P. Horstrand, S. Lopez, G. M. Callico, J. F. Lopez, and R. Sarmiento, “Novel architectures for real-time matching in hyperspectral images,” in *2011 3rd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2011, pp. 1–4.

[14] ——, “Novel architectures for real-time matching in hyperspectral images,” in *2011 3rd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE, 2011, pp. 1–4.