POLITECNICO DI TORINO

MASTER's Degree in ICT FOR SMART SOCIETIES

MASTER's Degree Thesis

Compression and Cloud screening for Satellite Images

Supervisors

Prof. Enrico MAGLI

Prof. Diego VALSESIA

Candidate

Alessia SCARDI

JULY 2025

Compression and Cloud screening for Satellite Images

Alessia Scardi

Abstract

The increasing volume of Earth Observation (EO) data generated by the high-resolution satellites' sensors poses critical challenges to onboard storage, transmission, and real-time processing. The images collected during the space missions need to clearly show the surface of the Earth, to be used in multiple contexts and fields: land cover classification, vegetation, ice and water analysis, atmospheric correction and mineral mapping. Unfortunately, the sky is not always clear of clouds, which degrade the information quality when appearing in the satellite images, often making major portions of the data unfit for further analysis and losing the very purpose the images were meant to serve. Accurate cloud detection algorithms are required to discriminate cloudy pixels from cloud-free ones directly onboard the satellite. The algorithms generate binary segmentation masks that are subsequently exploited during the onboard compression stage. According to the CCSDS 123.0-B-2 standard for multispectral and hyperspectral image compression, it is possible to assign different compression parameters depending on pixel classes. Specifically, cloudy pixels, being less informative, can be more compressed than cloud-free ones. This particular compression approach reduces the volume of data to be transmitted, saving on onboard resources.

The thesis addresses the challenge of detecting clouds directly onboard through a lightweight U-Net, and create the segmentation masks used in the compression stage. Satellites are equipped with MultiSpectral Imagers (MSI) Instruments which are provided with scanners acquiring data line by line to create an image. For this reason, standard image-level processing is impractical in onboard scenarios, as it would require loading entire high-resolution images into memory. To address this constraint, the proposed lightweight U-Net architecture is trained using two memory-efficient strategies: chunk-based and sliding-window-based learning. These approaches allow the network to operate on smaller portions of the image, significantly reducing the memory footprint during inference. This design choice ensures that segmentation masks can be generated progressively during onboard execution, enabling real-time cloud screening. Particular emphasis is placed on the trade-off between segmentation accuracy and computational efficiency.

To further reduce computational complexity and adapt the network for efficient deployment on hardware platforms, the lightweight U-Net used to

create binary cloud masks is optimized through quantization, specifically weight ternarization. To this end, several quantization strategies have been investigated and empirically compared using a consistent set of performance metrics. The presence of ternary weights simplifies hardware implementation by lowering complexity in arithmetic operations, resulting in a reduction in both memory usage and inference latency while, for the majority of methods, maintaining high segmentation accuracy.

Experimental results confirm the effectiveness of the proposed methods in both compression efficiency and cloud screening accuracy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

From thousands of kilometers above, satellites like the European Space Agency's Sentinel-2 are humans' eyes in the sky. Their constant, moving scan provides a stream of high-resolution images that has given us an incredible new window into our planet's health. We can now watch the ice sheets of Greenland shrink year by year, trace the path of a raging wildfire across a remote landscape in near-real-time and see the shifts in vegetation signaling a coming drought. This data is vital for monitoring the health of a single farmer's field, for tracking deforestation in the Amazon basin, for managing our water resources, for responding to natural disasters with never-seen-before speed and insight and many other aspects. However, this incredible capability has created a monumental problem: we are capturing data far faster than we can send it back to Earth. The data pipelines from space are narrow and expensive, creating a digital traffic difficulty that forces us to be incredibly selective about what we bring home. Adding to this challenge is a problem as old as the sky itself: clouds. Like a thumb over a camera lens, clouds and their shadows often block the most interesting parts of an image, rendering it partially or completely useless. Sending these cloudy, worthless images all the way back to Earth is a total waste of our limited satellite bandwidth. This reality calls for a smarter approach, that almost allows satellites to make decisions for themselves, right there in orbit.

This thesis proposes a smart system designed to work directly on a satellite, integrating cloud detection with data compression. First, an efficient deep learning model, built on a Lightweight U-Net architecture, acts as a "cloud-spotter", identifying which parts of an image are clear and which are obscured by clouds. Second, the cloud map it creates guides an adaptive compression system based on the CCSDS 123.0-B-2 standard from ESA. This system is able to compress the data from cloudy pixels much more aggressively, while carefully preserving the precious detail in the clear, scientifically valuable areas. Nevertheless, making this work on a satellite, with its tight power and memory

budgets, requires practical strategies to handle massive hyperspectral images. This work explores two of them: a simple chunk-based approach and a more robust sliding-window technique. Furthermore, to make the neural network itself as lightweight as possible, this research focuses on a technique called ternary weight quantization. A wide range of methods to reduce the model down to its bare essentials are tested, making it fast and efficient enough to run on low-power hardware without sacrificing its accuracy.

This thesis is structured as follows. Chapter 2 provides an overview of remote sensing fundamentals and image compression paradigms, with a focus on the Sentinel-2 mission and the CCSDS 123.0-B-2 standard. Chapter 3 introduces neural networks and quantization techniques, with particular attention to U-Net and ternary models. Chapter 4 describes the methodology for cloud screening and onboard compression, including preprocessing steps and model deployment strategies, used to develop the thesis' models and results. Chapter 5 discusses the experimental results performed on the models measuring segmentation accuracy, compression performance, and model efficiency.

Chapter 2

Background on Remote Sensing and Compression Techniques

2.1 Remote Sensing

Remote sensing is the science of acquiring information about the Earth's surface without being in direct physical contact with it. This is primarily accomplished by measuring and analyzing the electromagnetic radiation, such as visible light, infrared, and microwave energy, that is either reflected or emitted from objects on the ground. The sensors used for this purpose are typically mounted on platforms like satellites and can be broadly classified into two categories: passive sensors detect the natural radiation that is reflected or emitted from the Earth's surface, with the primary source of energy being the Sun. In contrast, active sensors, such as radar (SAR) and lidar, provide their own source of energy and measure the backscattered signal. By processing the data collected from these sensors, remote sensing enables the creation of detailed imagery and data products that are essential for monitoring environmental changes, managing natural resources, and understanding complex planetary systems.

2.2 Satellite Imaging Systems and Hyperspectral Data

Satellite imaging systems are the backbone of modern Earth observation (EO) missions, enabling the systematic acquisition of Earth data from orbit. These systems can be classified into panchromatic, multispectral, and hyperspectral, based on their spectral resolution and number of bands. Although panchromatic sensors provide high-resolution grayscale images that capture broad spectral content, multispectral sensors typically record data across 4 to 20 discrete spectral bands. Hyperspectral imaging (HSI) systems, on the other hand,

collect data in hundreds of narrow, contiguous spectral bands across visible, near-infrared, and shortwave infrared regions of the electromagnetic spectrum (approximately 400–2500 nm). The dense spectral sampling of the hyperspectral images helps in the distinction of fine-grained material, particularly used in applications such as land cover classification, vegetation analysis, atmospheric correction, and mineral mapping [1].

A hyperspectral image can be mathematically modeled as a three-dimensional data cube $\mathbf{X} \in \mathbb{R}^{H \times W \times B}$, where H and W represent the spatial dimensions (height and width), and B is the number of spectral bands. Each pixel $\mathbf{x}_{i,j} \in \mathbb{R}^B$ encodes a spectral signature characterizing the surface material at position (i, j) [2]. The spectral signature, formed by recording the reflectance across a large number of wavelengths, allows the use of techniques such as sub-pixel classification and spectral unmixing, even in heterogeneous land-cover scenes [3].

The high data volume of hyperspectral images, key to their scientific value, creates a significant bottleneck for onboard storage, transmission, and processing systems. For example, NASA's HyspIRI sensor is estimated to generate up to 5 TB of data per day [1], far exceeding the downlink capabilities of typical satellite platforms. Therefore, both onboard data compression and intelligent data prioritization have become crucial technologies in the design of modern imaging satellites.

Figure 2.1: Copernicus Sentinel-2C takes the skies in January 2025. Credit: ESA.

2.2.1 Sentinel-2 mission

An illustrative example of a modern EO mission is the *Sentinel-2* constellation, operated by the European Space Agency (ESA) as part of the *Copernicus Programme*. The mission aims at providing global, high-resolution, multispectral coverage in support of Copernicus land, emergency-response and security services [4].

The original Sentinel-2 mission comprises two twin, sun-synchronous, satellites, Sentinel-2A and Sentinel-2B, launched respectively in 2015 and 2017. Sentinel-2A has already been replaced by Sentinel-2C in January 2025 (shown in Figure 2.1) and Sentilel-2B will be succeeded by Sentinel-2D, creating the second generation of Sentinel-2 satellites, active until 2035, improving global measurements. The two satellites fly in the same 786 km orbit, phased 180° apart so that any point between 83°N and 56°S is revisited every ~5 days (10 days with a single spacecraft) [4], [5].

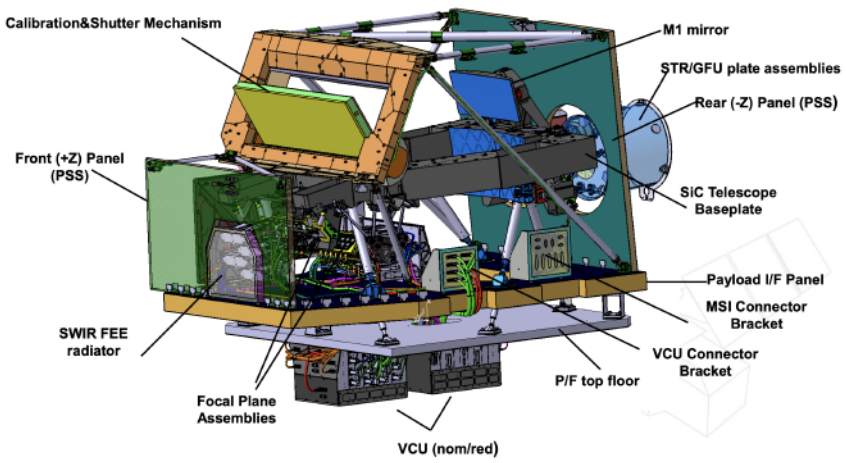


Figure 2.2: Onboard satellites MSI. Credit:ESA.

Each satellite is equipped with a MultiSpectral Instrument (MSI), shown in Figure 2.2, that enables systematic observation of terrestrial surfaces capturing data across 13 spectral bands, described in Figure 2.3.

MSI's core is a silicon-carbide, three-mirror anastigmatic telescope which directs light onto two separate focal-plane assemblies, one for the visible and near-infrared (VNIR) spectrum and another for the shortwave infrared (SWIR). Together, these assemblies form a continuous 25,000-pixel line of detectors, giving the instrument a swath of 290 km. Sentinel-2 operates in *push-broom mode*: every exposure records a complete cross-track line in all 13 spectral bands simultaneously, while the satellite's orbital motion provides the second dimension of the image. The line-by-line "carpet-mapping" is highly efficient, generating around 1.6 TB of raw data per orbit at three distinct resolutions:

four bands at 10 m, six at 20 m (including three red-edge channels), and three atmospheric bands at 60 m [4].

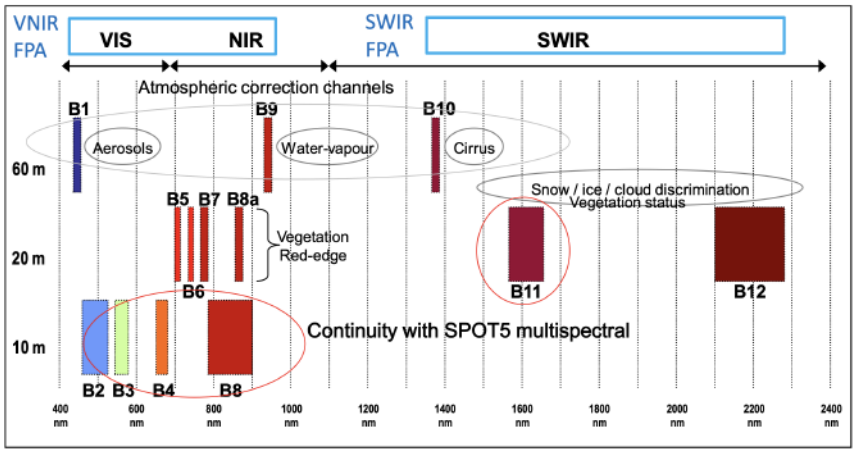


Figure 2.3: MSI spectral bands. Credit:ESA.

Copernicus User Level Data, shown in Table 2.1, need to go through a rigorous calibration and validation standard processing chain before it can be used for scientific analysis and quantitative Earth-observation applications [5]. The Mission Performance Cluster (MPC) is responsible for evaluating data quality, monitoring instruments' performance and ensuring compliance with product specifications. Furthermore, it oversees the development and maintenance of essential calibration resources, such as scientific algorithms and processing modules, to guarantee the delivery of consistent, high-quality data throughout the mission's lifecycle.

2.3 Image Compression for Remote Sensing

While scientifically valuable, the hyperspectral data contains significant redundancy due to high correlation between spectral bands. Even if this redundancy burdens storage systems, it can also be used for efficient data processing. Techniques like Principal Component Analysis (PCA), Independent Component Analysis (ICA), and deep autoencoders exploit this correlation to perform tasks such as compression, dimensionality reduction, and anomaly detection, all while preserving the data's most informative features. However, atmospheric conditions, particularly cloud cover, significantly affect the usability of optical hyperspectral data. Approximately 70% of the Earth's surface is obscured by clouds at any given time, rendering large portions of collected imagery uninformative for downstream applications. This has motivated the development of onboard cloud screening algorithms that operate in real-time to detect and discard cloud-covered regions before compression and transmission. Techniques range from threshold-based detectors, as Sen2Cor [6] and Fmask

Table 2.1: Overview of Sentinel-2 Level-1 and Level-2 products. Credit: [5]

Processing Level	**Key processing steps / outputs**
Level-1	*Radiometric corrections*: stray-light / crosstalk removal, defective-pixel exclusion, de-noising, de-convolution, relative & absolute calibration. *Geometric corrections*: inter-band & inter-detector co-registration, ortho-rectification.
Level-2	*a) Cloud screening.* *b) Atmospheric corrections*: thin-cirrus removal, terrain-slope and adjacency-effect compensation. *c) Retrieval of geophysical variables*: e.g. Fraction of Absorbed Photosynthetically Active Radiation (fAPAR), leaf chlorophyll content, Leaf Area Index (LAI), land-cover classification.
Level-3	Spatio-temporal synthesis products (e.g. generation of cloud-free mosaics by simulating/merging cloud-corrections applied to Level-2 imagery).

[7], to lightweight convolutional neural networks (CNNs) such as U-Net and KappaMask [8]. These approaches aim to preserve cloud-free observations while reducing data volume and avoiding unnecessary bandwidth waste.

2.3.1 Compression Paradigms

Image compression is essential for transmitting data on Earth quickly and efficiently. The compression is categorized into three paradigms: *lossless*, *lossy* and *near-lossless*, depending on the level of allowable information distortion.

Lossless Compression

Lossless compression paradigms allow to exactly reconstruct the original image, without losing information during the decompression stage [9]. While its primary advantage is complete data fidelity, the achievable compression ratios are only modest when compared to other techniques. The performance of lossless compression is fundamentally limited by the image's entropy, a statistical measure of its information content, which sets a theoretical boundary on how much the data can be compressed. The most common implementations are prediction-based algorithms such as Fast Lossless (FL) [10], DPCM [11], and the CCSDS-123 standard.

Lossy Compression

In lossy compression, the decompressed image results distorted with respect to the original one. The distortions are introduced during a quantization stage and cause the permanent loss of information when decompressing the image. The decompressed image results to be an approximation of the original, not its exact copy. However, lossy compression achieves higher compression ratios then the lossless technique. This compression paradigm is usually built on top of transform-based approaches like Principal Component Analysis (PCA), Discrete Cosine Transform (DCT), and Discrete Wavelet Transform (DWT). These transforms are effective because they concentrate the most significant visual information into a small number of coefficients, allowing to eliminate the less important data and significantly reduce the file size [9].

Near-Lossless Compression

Near-lossless compression is a compromise between the other two paradigms, offering better compression ratios than lossless methods while ensuring that data degradation is strictly controlled. This is achieved by limiting the pixel distortion to a previously defined absolute or relative error, with a more stringent definition constraining the error to the level of the instrument's intrinsic noise. This is implemented in three main ways [9]: quantizing the prediction error and then encoding it losslessly, as done in CALIC [12] and NL-CCSDS-123 [13] methods, pre-quantizing the entire original image before applying a lossless coder or using a two-stage process where a lossy version is created, and the residual difference is then quantized and losslessly encoded.

2.3.2 The CCSDS 123.0-B-2 Compression Standard

The *Consultative Committee for Space Data Systems (CCSDS)* developed the 123.0-B-2 standard to support efficient onboard compression of multispectral and hyperspectral image data. This standard extends its lossless-only predecessor (CCSDS 123.0-B-1) by introducing a near-lossless mode, which allows for lossy compression with user-defined absolute and/or relative error limits in the reconstructed image. Designed to be low-complexity and hardware-friendly, CCSDS 123.0-B-2 is now adopted by major space agencies like NASA and the European Space Agency (ESA). The new features, including a new hybrid entropy coder, enable significantly smaller data volumes while precisely controlling the quality of the decompressed images.

Compressor and Decompressor Structure

The compression chain of the CCSDS 123.0-B-2 standard consists of two main functional blocks: a Predictor and an Encoder, as shown in Figure 2.4. The Predictor stage generates a prediction for each sample, computes the prediction error, and quantizes it. The subsequent Encoder stage losslessly compresses the resulting stream of mapped quantizer indices.

The decompressor performs the inverse operations in a symmetric manner to ensure that predictions are identical at both ends, which is critical for a successful reconstruction [1].

Compressor Pipeline

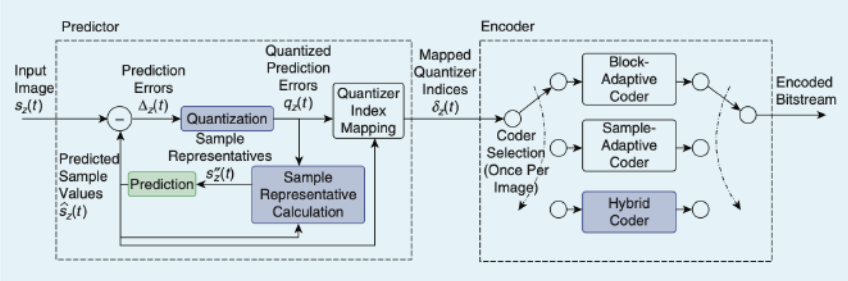


Figure 2.4: Compressor pipeline. Credit [1].

The compression process is a single-pass operation for each image sample.

1. **Prediction**: A causal predictor is used to estimate the current pixel value $\hat{s}_z(t)$.This prediction is not based on original pixel values but on *sample representatives*, values derived from previously coded data. This closed-loop design ensures the decompressor can perfectly replicate the prediction, which is calculated using a weighted sum of *local differences* computed from *local sums* of neighboring *sample representatives* in the current and previous spectral bands.

2. **Prediction Error Computation**: The prediction error (or residual) is the difference between the original sample and its predicted value:

$$\Delta_z(t) = s_z(t) - \hat{s}_z(t). \tag{2.1}$$

3. **Quantization (for Near-Lossless)**: The prediction error $\Delta_z(t)$ is uniformly quantized to a quantizer index $q_z(t)$. This stage is always active, but for lossless compression, the user sets the error limit to zero, which makes the quantization lossless ($q_z(t) = \Delta_z(t)$). For near-lossless compression, the quantizer's bin size is determined by user-specified

absolute and/or relative error limits, ensuring that the reconstruction error is bounded.

4. **Quantizer Index Mapping**: The potentially negative quantizer indices $q_z(t)$ are mapped to non-negative integers, $\delta_z(t)$, which are suitable for the entropy coders.

5. **Entropy Coding**: The sequence of mapped quantizer indices $\delta_z(t)$ is compressed using one of three available entropy coders:

   - **Sample-Adaptive**: Uses a family of Golomb-Power-of-2 (GPO2) codes, adapting the code for each sample.
   - **Block-Adaptive**: Partitions samples into blocks and encodes each block with the most efficient of five methods (including Rice coding and no compression).
   - **Hybrid** (new in Issue 2): Classifies samples as high- or low-entropy. It uses a variation of GPO2 codes for high-entropy samples and a family of 16 efficient variable-to-variable-length codes for low-entropy samples, which are prevalent in near-lossless mode.

Decompressor Pipeline

The decompression process symmetrically reverses the compression steps to reconstruct the image.

1. **Entropy Decoding**: The compressed bitstream is decoded using the corresponding entropy coder (sample-adaptive, block-adaptive, or hybrid) to perfectly recover the sequence of mapped quantizer indices $\delta_z(t)$.

2. **Inverse Quantizer Index Mapping**: The non-negative indices $\delta_z(t)$ are mapped back to the original quantizer indices $q_z(t)$.

3. **Prediction**: The decompressor calculates the exact same *sample representatives* from previously decoded data that the compressor used. Using the *sample representatives*, it applies the identical prediction algorithm to compute the predicted sample value $\hat{s}_z(t)$.

4. **Reconstruction:** The reconstructed sample value is obtained by adding the de-quantized error to the prediction. The de-quantized error is derived from the index $q_z(t)$ and the predicted value $\hat{s}_z(t)$, resulting in a final reconstructed sample not exceeding the user-specified error limits.

This closed-loop design, where both compressor and decompressor make predictions based on shared, reproducible data (the *sample representatives*),

is crucial to ensure controlled error reconstruction in near-lossless mode and bit-exact reproduction in lossless mode [1].

Chapter 3

Background on Deep Learning

3.1 Neural Networks and Deep Learning

The development of deep learning has been strongly influenced by biological systems like the human brain, which process sensory information through multiple hierarchical layers. The visual cortex, in particular, has served as a primary source of inspiration for the architecture of Artificial Neural Networks (ANNs).

In their 1962 work, Hubel and Wiesel discovered that neurons in the early visual cortex of cats are selectively responsive to primitive visual features, such as specific edge orientations [14]. These neurons respond to increasingly abstract visual stimuli as information moves through successive layers of the cortex, revealing that the brain performs hierarchical feature extraction: lower layers detect primitive features like edges and texture, and higher layers assemble these into more complex representations, identifying shapes, surfaces, and ultimately full objects.

This idea of hierarchical processing is central to deep learning. As the brain organizes perception into layers of increasing abstraction, deep neural networks are designed with multiple layers to learn a hierarchy of features. Early layers automatically extract low-level features from raw data, which are then combined in deeper layers to form more abstract, task-specific representations [15].

Traditional machine learning models often rely on shallow architectures, typically with one or two layers. While effective for some tasks, these models directly map inputs to outputs (in supervised environments) or to latent representations (in unsupervised settings) without intermediate stages of abstraction. However, those simple mappings can be insufficient for complex, high-dimensional data like satellite imagery, where meaningful features are often deeply embedded and require a hierarchical approach to be effectively captured.

Inspired by this biological paradigm, Deep learning (DL) is a subfield of machine learning that employs multi-layered neural networks to model complex patterns in the data. Unlike traditional algorithms, which often require manual feature extraction, deep learning models can automatically learn hierarchical representations from raw data since each layer contains non-linear processing units able to transform the output of the previous layer into a slightly more abstract representation. Although originally motivated by visual perception, this approach has proven remarkably effective across a broad range of domains, including image and speech recognition, natural language processing, and beyond.

3.1.1 Neural Networks (NNs)

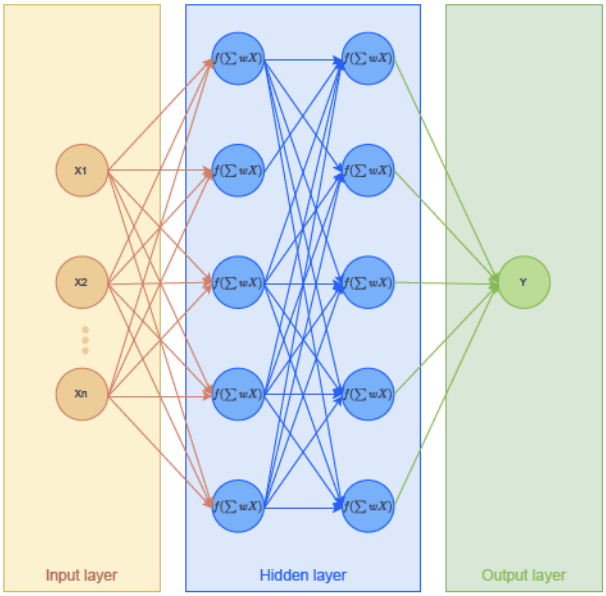


Figure 3.1: A diagram of a fully connected Artificial Neural Network (ANN), also known as a Multi-Layer Perceptron. It shows an input layer, a hidden layer where all neurons are interconnected, and an output layer.

Artificial Neural Networks (ANNs) are computational models designed to approximate complex functions by learning from data. They are composed by interconnected processing units, known as neurons, organized in layers, as shown in Figure 3.1. Each neuron performs a simple transformation on its inputs, and their collective action enables the network to model complex relationships. Neural networks are compatible with different learning paradigms:

- **Supervised learning**, where a Neural Network is trained on a labeled dataset to learn a mapping from inputs to target outputs.

- **Unsupervised learning**, where the network identifies latent patterns, structures, or groupings within unlabeled data.
- **Reinforcement learning**, where an *agent* learns an optimal *policy* for decision-making by interacting with an environment and receiving reward feedback.

This thesis focuses specifically on supervised learning for image classification tasks using hyperspectral satellite data. In this context, a neural network is trained to map high-dimensional image data to discrete class labels to perform the automatic detection of clouds: the network learns to binary classify each pixel of an image as *cloud* or *non-cloud*. The Architecture of a simple neural

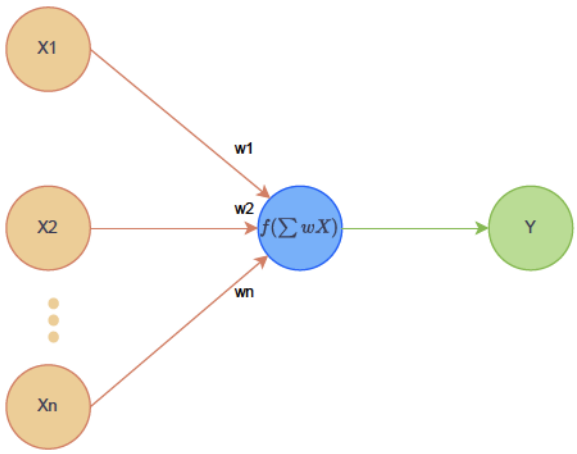


Figure 3.2: Simple Neural Network Architecture.

network is shown in Figure 3.2. Its fundamental computational unit is the neuron, able to compute a weighted sum of its inputs followed by the application of a non-linear activation function. Mathematically, the output y of a single neuron can be expressed as shown in eq. 3.1.

$$y = f(\mathbf{w}^\top \mathbf{x} + b) \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\mathbf{w} \in \mathbb{R}^n$ is a vector of learnable weights connecting the input layer and the neuron, $b \in \mathbb{R}$ is a bias term introducing an intercept, and f is a non-linear activation function.

Activation functions introduce non-linearity into the model, enabling the network to approximate complex, non-linear patterns that cannot be captured by linear transformations alone. The choice of the activation function significantly impacts a network's performance and training dynamics.

Among the earliest and most commonly used activation functions is the

Sigmoid function, defined in eq. 3.2.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.2}$$

The Sigmoid function maps its input to the range $[0, 1]$, making it suitable for binary classification outputs. However, it suffers from the vanishing gradient problem: for inputs with large absolute values, the function saturates, and its derivative approaches zero. This can cause learning to stall in deep networks, as gradient updates become negligible during backpropagation [16].

The **hyperbolic tangent (tanh)** function addresses some of these issues by mapping inputs to the range $[-1, 1]$, which is zero-centered. Its mathematical formulation is shown in eq. 3.3.

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{3.3}$$

While its zero-centered nature can lead to faster convergence in practice, the hyperbolic tangent also saturates and suffers from the vanishing gradient problem in deep architectures [17].

A significant advancement in deep learning came with the introduction of the **Rectified Linear Unit (ReLU)**, defined as in eq. 3.4.

$$f(z) = \max(0, z) \tag{3.4}$$

ReLU has become the default activation in many modern NN architectures due to its computational simplicity and its effectiveness in mitigating the vanishing gradient problem for positive inputs [18]. By setting negative inputs to zero, ReLU introduces sparse activations, which can be computationally efficient. However, this leads to a potential issue known as the *dying ReLU* problem, where neurons can become permanently stuck in a state of outputting zero if their weights are updated such that their input is always negative.

To address this, variants such as **Leaky ReLU** and **Exponential Linear Units (ELUs)** have been proposed. Equation 3.5 shows the mathematical formulation of the Leaky ReLU, which modifies the function to allow a small, non-zero gradient when the unit is not active.

$$f(z) = \begin{cases} z & \text{if } z > 0, \\ \alpha z & \text{otherwise,} \end{cases} \tag{3.5}$$

where $\alpha \in (0, 1)$ is a small constant [19]. ELUs further generalize this idea by

introducing exponential behavior in the negative region, as shown in eq. 3.6.

$$f(z) = \begin{cases} z & \text{if } z > 0, \\ \alpha(e^z - 1) & \text{if } z \leq 0, \end{cases} \tag{3.6}$$

which can help networks converge faster by pushing the mean activations closer to zero [20].

In classification tasks, especially those with multiple output classes, the **softmax function** is typically applied to the final layer. It converts a vector of real-valued scores into a probability distribution over K classes, as shown in eq. 3.7.

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, \ldots, K. \tag{3.7}$$

This formulation ensures that all output values lie in $[0, 1]$ and sum to 1, which is ideal for probabilistic interpretation in multi-class classification.

Once the network architecture is defined, the learning process involves minimizing a *loss function* (such as cross-entropy) that measures the discrepancy between the network's predictions and the true labels. This minimization is typically performed using gradient-based optimization algorithms, like Stochastic Gradient Descent (SGD). The gradients of the loss function with respect to the network's parameters are calculated efficiently using backpropagation algorithms [21], and these gradients are used to iteratively update the weights and biases. The interaction between layered representations, non-linear activations, and gradient-based optimization is what grants neural networks their remarkable expressive power and ability to generalize to unseen data.

3.1.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized class of neural networks that have become a cornerstone of modern deep learning. They are specifically designed to process data with a grid-like topology, such as images, videos and time series, and their design is heavily inspired by the organization of the animal visual cortex [22]. Unlike fully connected networks, CNNs leverage two key principles to efficiently learn from spatial data: local receptive fields and parameter sharing. This allows them to automatically and hierarchically learn spatial features, from simple edges in early layers to complex objects in deeper layers.

As shown in Figure 3.3, the typical CNN architecture consists of a sequence of layers, with the most common building blocks being convolutional layers, activation layers, and pooling layers, followed by one or more fully connected layers for classification or regression.

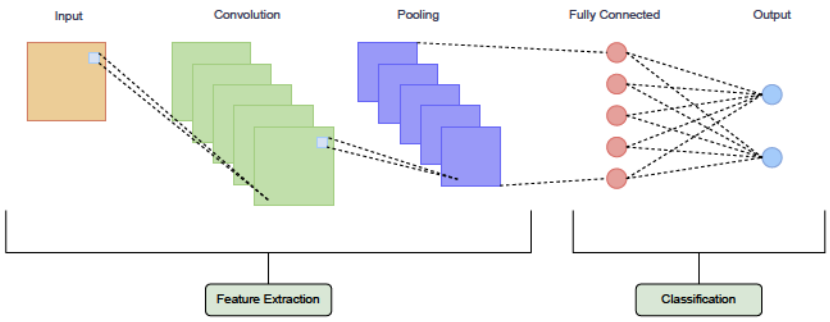


Figure 3.3: The architecture of a typical Convolutional Neural Network (CNN). It illustrates the flow from an input image through convolutional and pooling layers for feature extraction, followed by a fully connected and an output layer.

In detail:

1. **Convolutional Layer**: The convolutional layer is the core component of a CNN. Its purpose is to detect local features in the input data using a set of learnable filters, also known as kernels. Each kernel is a small matrix of weights that slides (or convolves) across the input volume, computing a dot product at each location. This operation produces a 2D feature map (or activation map), where each value indicates the presence and strength of that specific feature (such a vertical edge or a specific texture) at that spatial position.

   The 2D convolution operation for an input image X and a kernel K is defined as:

$$Y(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot K(m,n) + b \tag{3.8}$$

   where $Y(i,j)$ is the output at position (i,j) in the feature map, the summations are over the dimensions of the kernel, and b is a learnable bias term.

   Two critical hyperparameters control the behavior of the convolutional layer:

   - **Stride**: The number of pixels the kernel moves at each step. A stride of 1 results in overlapping receptive fields, while a larger stride produces a smaller output volume.
   - **Padding**: Adding a border of zeros around the input volume. This allows for control over the spatial dimensions of the output and helps to keep the kernel's receptive field centered on border pixels.

   Parameter sharing allows the CNN to use the same kernel (and its set of

weights) across all spatial locations of the input. This drastically reduces the number of learnable parameters compared to a fully connected network, making CNNs more memory-efficient and less prone to overfitting.

2. **Activation Layer**: Immediately following each convolutional layer, an activation function such as ReLU is applied element-wise to the feature map. This introduces non-linearity, allowing the network to learn more complex features.

3. **Pooling Layers**: used to perform spatial downsampling, which is used for two main purposes: reducing the dimensionality of the feature maps (decreasing computational cost) and creating a degree of invariance to small translations in the input data. Common pooling operations include:

   - **Max pooling**: For each local window in the feature map, it outputs the maximum value. This is effective at capturing the most prominent features.

   $$y_{i,j} = \max_{(m,n)\in P} x_{i+m,j+n} \tag{3.9}$$

   - **Average pooling**: It computes the average value within each window.

   $$y_{i,j} = \frac{1}{|P|} \sum_{(m,n)\in P} x_{i+m,j+n} \tag{3.10}$$

4. **Fully Connected Layers**: After several convolutional and pooling layers have extracted hierarchical features, one or more fully connected layers are typically used at the end of the network. The high-level feature maps are first flattened into a 1D vector. This vector then serves as the input to the fully connected layers, which perform the final mapping from features to output classes (often followed by a softmax function).

An efficient alternative to standard convolutions is **Depthwise Separable Convolutions**, which reduce model size and computational complexity by splitting the convolution into two separate steps [23]. First, a *depthwise* convolution applies a single 2D filter to each input channel independently creating an intermediate set of feature maps. This captures spatial patterns within each channel but does not combine information across channels. Second, a pointwise convolution handles the channel combination through a 1x1 convolution, combining the outputs of the depthwise convolution to produce the final feature maps. This 1x1 filter processes all channels at each pixel location to compute new features, effectively mixing the channel information. This factorization

significantly reduces the number of parameters and is a key component in modern, efficient architectures like MobileNet and Xception.

The convolutional operator is highly versatile and can be adapted to different data dimensionalities:

- 1D Convolutions are primarily used for sequential data like time series and audio signals.
- 2D Convolutions are the standard for image data, capturing spatial patterns.
- 3D and Multidimensional Convolutions are employed for volumetric data or video, where they can capture features in both spatial and temporal dimensions.

3.1.3 U-Net

The U-Net architecture [24] is a specialized Convolutional Neural Network originally designed for biomedical image segmentation. It is particularly powerful because it can be accurately trained on very few annotated images, making it very useful when the available data is limited, while achieving precise segmentation. Its architecture is characterized by a symmetric encoder-decoder structure, visually resembling the letter *U*, as shown in Figure 3.4.

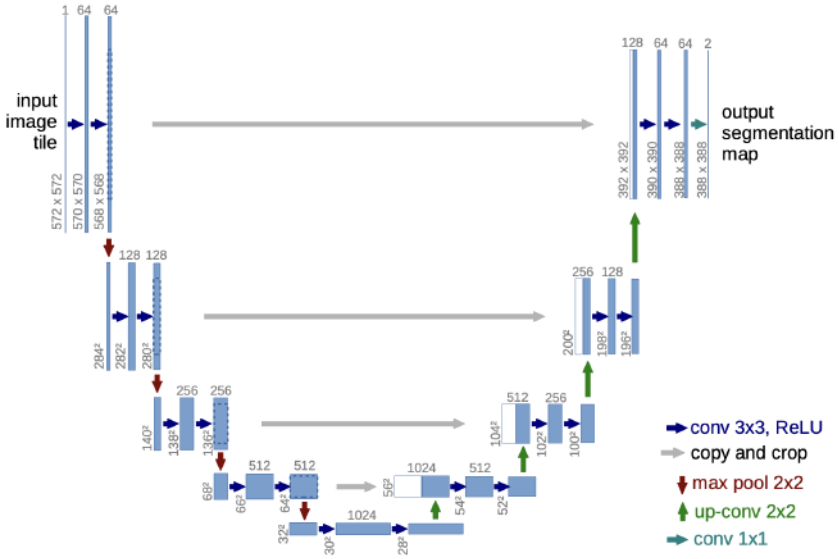


Figure 3.4: The U-Net architecture, showing the symmetric contracting (encoder) and expansive (decoder) paths with skip connections between them. Credit [24].

The network's main components are:

- The **Encoder** (Contracting Path): This part captures the context of the input image. It consists of successive blocks, each containing repeated convolutional layers followed by a pooling (or strided convolution) operation for downsampling. This process progressively reduces the spatial dimensions while increasing the number of feature channels, allowing the network to learn the content (*what*) of the image at an abstract level.

- The **Decoder** (Expanding Path): This part enables precise localization. It systematically upsamples the feature maps from the bottleneck while concatenating them with high-resolution feature maps from the corresponding level in the encoder via *skip connections*. These skip connections are the most critical innovation of U-Net, as they allow the decoder to recover fine-grained spatial information (*where*) that is lost during downsampling. Each upsampling step is followed by convolutions to refine the features.

The final layer of the U-Net is typically a 1x1 convolution that maps the feature channels to the desired number of output classes, followed by a Sigmoid (for binary segmentation) or softmax activation to produce the final probability map. The U-Net's design allows it to produce high-resolution segmentation masks and has led to its widespread adoption in medical imaging and other domains requiring precise delineation of structures.

3.1.4 Lightweight U-Net-based model

Building on the U-Net paradigm, a lightweight neural network for on-board satellite cloud detection is proposed in [25], demonstrating that a deep learning model can achieve performance comparable to traditional physical algorithms while significantly reducing computational complexity, for resource-constrained satellite hardware.

As shown in Figure 3.5, the network is a modified U-Net architecture designed for maximum efficiency:

- **Simplified Encoder**: The encoder is significantly shallower than a standard U-Net, consisting of only three blocks: an initial stem layer for feature extraction, followed by two downsampling blocks. This reduces model depth and complexity.

- **Efficient Convolutions**: To minimize parameters and inference time, each downsampling block employs a Depthwise Separable Convolution (SDC) with a stride of 2. This replaces both the standard convolution and the separate pooling layer of the classical U-Net, halving the spatial resolution efficiently.

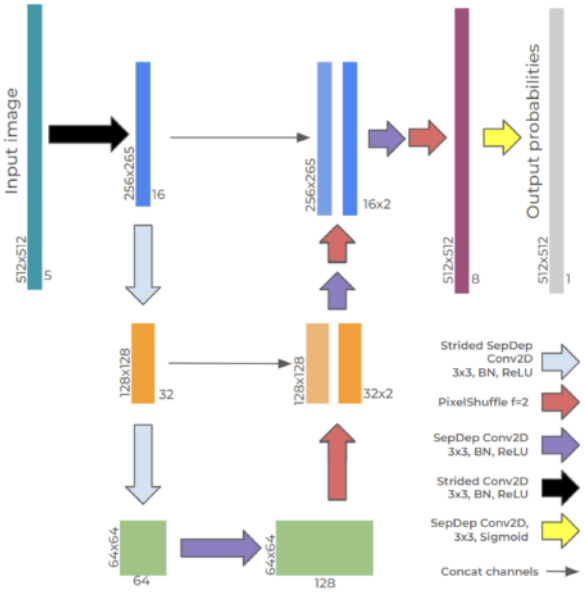


Figure 3.5: Architecture of the lightweight U-Net model designed for on-board cloud detection. It features a reduced number of blocks and uses efficient operations like Depthwise Separable Convolutions and PixelShuffle. Credit [25]

- **Efficient Upsampling**: The decoder mirrors the encoder's structure. Upsampling is achieved using PixelShuffle layers, which are computationally cheaper and avoid the checkerboard artifacts sometimes produced by transposed convolutions.
- **Skip Connections**: Like the original U-Net, skip connections link the encoder and decoder at each resolution level, ensuring that fine-grained spatial details are preserved for accurate segmentation. A small bottleneck block sits between the encoder and decoder to manage channel dimensions.
- **Output Layer**: The final block uses a standard convolution followed by a Sigmoid activation to generate the pixel-wise cloud probability map.

The network is trained on a specific subset of Sentinel-2 spectral bands (B2, B3, B4, B8, and B10) chosen for their effectiveness in distinguishing clouds. To stabilize training in such a shallow architecture, Batch Normalization layers are used before each activation. The model is optimized using the Binary Cross-Entropy (BCE) loss function, which is standard for binary segmentation tasks. This design enables the network to reduce the parameter count from millions to thousands (about 20k) while maintaining competitive accuracy.

Performance Evaluation and Results

The effectiveness of this lightweight design was validated through a set of experiments detailed in [25] and conducted both thick and thick cloud labeled

data. Its performance was compared to the highly optimized, traditional Sen2Cor physical algorithm. The evaluation was based on several key metrics crucial for on-board cloud screening, the same used for the purpose of this thesis and described in section 5.

The main comparison with physical algorithms shows that the Lightweight U-Net model significantly outperforms the best-performing version of Sen2Cor.

The lightweight model, with a Cloud Probability Threshold (CPT) of 0.80, achieves a Balanced Overall Accuracy (BOA) (eq. 5.4) of 0.86 and a precision (eq. 5.3) of 0.96. In contrast, the best Sen2Cor configuration reaches a BOA of only 0.79 and a precision of 0.88.

The Lightweight model also demonstrates a better False Positive Rate (5.2) with respect to Sen2Cor's: just 0.02 (2%), compared to 0.06 (6%). The reduction in false positives is a decisive advantage for on-board systems, as it minimizes the erroneous discarding of useful, clear-sky data while maintaining a higher cloud detection rate (TPR (eq. 5.1) of 0.73 compared to Sen2Cor's 0.63).

Another significant advantage of this neural network is its tunability via the Cloud Probability Threshold (CPT), which is the confidence threshold applied to the Sigmoid output that allows a direct trade-off between TPR and FPR to match specific mission goals:

- A low CPT, such as 0.5, maximizes cloud detection, achieving a very high TPR of 0.87 at the cost of a higher FPR (0.06). This setting is ideal for missions where it is critical to identify every possible cloud.
- A high CPT, such as 0.9, minimizes false alarms, achieving a near-perfect FPR of 0.00 (less than 0.5%), while still correctly identifying the majority of clouds (TPR of 0.58). This is suitable for missions where preserving clear-sky data is the top priority.

This flexibility is a marked improvement over physical algorithms, which have less adaptable, hard-coded thresholds.

3.2 Quantization of Neural Networks

The deployment of deep neural networks (DNNs) on resource-constrained platforms, such as edge devices or onboard satellite systems, produces significant challenges in terms of memory footprint, computational latency and power consumption. Model compression strategies are essential to mitigate these issues, and neural network quantization is one of the most effective approaches. While it enables efficiency gains, the process introduces quantization noise, which can lead to a degradation in the accuracy of the model [26].

Traditional DNNs are trained and often perform inference using 32-bit floating-point (FP32) arithmetic, which is a high-precision format, even if computationally expensive. The fundamental operation in dense and convolutional layers is the multiply-accumulate (MAC) operation, defined for a single neuron as:

$$A_n = b_n + \sum_m W_{n,m} \cdot x_m \tag{3.11}$$

The energy cost of this operation scales non-linearly with the number of bits required for the representation [27]. An 8-bit integer MAC, for instance, is significantly more energy- and area-efficient than its 32-bit floating-point counterpart. Furthermore, reducing the bit-width from 32 to 8 bits decreases the memory required to store weights and activations by a factor of 4, alleviating memory bandwidth bottlenecks, which are a major source of latency and power drain in embedded systems.

To leverage these hardware benefits, full-precision floating-point tensors are mapped to low-bit integer formats. The core idea is to approximate a floating-point vector x using a low-precision integer representation x^{int}. This mapping is controlled by two key parameters: a floating-point scale factor s_x, which defines the *step size* or *real-world value* of each integer increment, and an integer zero-point z_x, which ensures the value 0.0 is represented perfectly without error. The mathematical formulation between the real value and its quantized representation is shown in eq. 3.12.

$$\mathbf{x} \approx s_x(\mathbf{x}^{\text{int}} - z_x) \tag{3.12}$$

By quantizing both weights and activations, the computationally intensive summation performed by each neuron of the network ($\sum w$) can be reformulated so that the bulk of the work, the multiplications and additions, is performed entirely with simple integer arithmetic, as shown by eq. 3.13.

$$\hat{A}n \approx b_n + s_w s_x \sum_m (Wn, m^{\text{int}} - z_w) \cdot (x_m^{\text{int}} - z_x) \tag{3.13}$$

The expensive floating-point scale factors are only multiplied once after the entire summation is complete. To prevent numerical overflow, this sum is calculated in a high-precision (32-bit) accumulator. The final result is then requantized back to a low-bit format before being sent to the next layer, thus maintaining high computational efficiency throughout the entire network pipeline.

3.2.1 Post-Training Quantization vs Quantization-Aware Training

Quantization is applied to neural networks using two main strategies: **Post-Training Quantization (PTQ)** and **Quantization-Aware Training (QAT)** [26].

- **Post-Training Quantization (PTQ)** is a fast and data-light approach where a pre-trained FP32 model is converted to a low-precision format without any retraining. The process involves calibrating the optimal scale factor and zero-point for each tensor using a small, representative set of calibration data. Because PTQ does not require access to the original training pipeline or labeled data, it is simple and computationally cheap. However, this simplicity can come at the cost of accuracy, especially when quantizing to very low bit-widths (4-bit or less), as the model's weights have not been optimized to be robust to quantization noise.

- **Quantization-Aware Training (QAT)** simulates the effects of quantization during the training process. "Fake" quantization nodes are inserted into the network graph, which quantize weights and activations in the forward pass but allow full-precision gradients to flow through during the backward pass. This is typically achieved using a Straight-Through Estimator (STE), which approximates the gradient of the non-differentiable rounding function as 1. By exposing the model to quantization noise during training, QAT allows the optimizer to find weight distributions that are much more resilient to the effects of rounding and clipping. While computationally expensive, QAT almost always achieve higher accuracy than PTQ and is essential for obtaining good performance with aggressive, low-bit quantization.

3.2.2 Ternary Neural Networks (TNNs)

Ternary Neural Networks are an aggressive form of quantization where weights are constrained to a set of three values: $\{-\alpha, 0, +\alpha\}$, where α is a learnable or fixed scaling factor. This approach offers benefits with respect to the standard 8-bit quantization:

- Computational Efficiency: Multiplications can be replaced by simple additions and subtractions for the values $-\alpha$ and $+\alpha$, and conditional gating for the 0 values, which are much cheaper in hardware.

- High Sparsity: The presence of the zero value induces high levels of activation and weight sparsity, which can be exploited by hardware to

skip computations entirely, further reducing power consumption.

- Improved Expressiveness: Compared to binary networks (which only use two values), the zero in ternary networks provides a crucial extra degree of freedom, allowing TNNs to achieve accuracy much closer to their full-precision counterparts.

As with other low-bit schemes, TNNs are typically trained using Quantization Aware Training with a STE. Advanced techniques like Fine-Grained Quantization (FGQ) have been proposed to further improve accuracy [28]. Instead of using a single scaling factor α for an entire layer, FGQ partitions the weights into small groups and computes an independent scaling factor for each group. This allows the model to better adapt to the varying dynamic ranges of different features within a layer, resulting in minimal accuracy loss even with 2-bit (ternary) weights.

3.2.3 Ternary Weight Networks (TWN)

Ternary Weight Networks, introduced in [29], are a variant of TNNs, where the quantization is applied exclusively to the weights of a neural network, while activations are typically kept at full precision. This approach constrains the weights to the values $-\alpha$, 0, or $+\alpha$. Here, α is a positive, layer-specific scaling factor that compensates for the precision loss and it is optimized alongside the weights.

The key advantage of TWNs over binary networks lies in the inclusion of the zero value. This allows for true weight pruning, introducing significant sparsity that can be exploited by specialized hardware to skip MAC operations entirely, leading to substantial reductions in both computational latency and power consumption. The authors of [29] highlight that this approach achieves up to a 16x model compression rate compared to FP32 models and offers vastly superior expressive power, as a 3x3 ternary filter has over 19.000 possible patterns compared to just 512 for a binary filter.

The core of the TWN methodology is to find the optimal ternary weights $\mathbf{W}_t$, in the range $\{-1, 0, +1\}$ and scaling factor α that minimize the L2 distance to the original full-precision weights $\mathbf{W}$, as shown in equation 3.14. Then the effective weight used during inference is not $\mathbf{W}_t$ but its scaled version $\alpha \cdot \mathbf{W}_t$. The optimal ternary weights $\widetilde{W}$ and scaling factor α solve:

$$\alpha^{,}\mathbf{W}_t^{=} \arg \min_{\alpha \geq 0,, W_{t,i} \in -1,0,+1} |\mathbf{W} - \alpha \mathbf{W}_t|_2^2 \tag{3.14}$$

Solving this optimization problem directly is computationally expensive. Therefore, the authors propose an efficient, threshold-based approximation. For

a given threshold $\Delta > 0$, the ternary weights are determined as:

$$\widetilde{W}_i = \begin{cases} +1 & \text{if } W_i > \Delta \\ 0 & \text{if } |W_i| \leq \Delta \\ -1 & \text{if } W_i < -\Delta \end{cases} \tag{3.15}$$

With the ternary weights fixed, the optimal scaling factor α can be calculated directly as the average magnitude of the non-zeroed weights:

$$\alpha = \frac{1}{|\mathcal{I}\Delta|} \sum i \in \mathcal{I}\Delta |W_i| \quad \text{where } \mathcal{I}\Delta = \{i \mid |W_i| > \Delta\} \tag{3.16}$$

The authors of [29] suggest a practical rule of thumb for setting the threshold Δ based on the assumption of a normal weight distribution, approximating it as a fraction of the mean absolute weight for the layer:

$$\Delta = 0.7 \cdot \mathbb{E}[|W|] \tag{3.17}$$

A fundamental challenge in training all the Ternary Networks, is that the ternarization function is discrete and non-differentiable, which prevents gradients from flowing during backpropagation. To overcome this, Straight-Through Estimator (STE) is used. During training, the model maintains a full-precision "latent" copy of the weights. In the forward and backward passes, the ternarized version of the weights ($\alpha \cdot \mathbf{W}_t$) is used for all computations. However, during the weight update step, the calculated gradients are applied directly to the latent full-precision weights. This allows the model to accumulate small gradient updates effectively, ensuring stable learning while still optimizing a network that will be purely ternary at inference time. Once training is complete, the latent weights are discarded, leaving only the compact ternary model.

3.2.4 Absmean Ternary Quantization

Building on the principles of ternary networks, the BitNet b1.58 model [30] introduces a Quantization-Aware Training approach designed for Large Language Models. It employs a method called *absmean* quantization to constrain the network's weights to the set {-1,0,+1}. Unlike the explicit thresholding method used in TWNs, this technique first scales the entire weight matrix $\mathbf{W}$ by its mean absolute value $\gamma = E[|\ \mathbf{W}\ |]$. Each scaled weight is then simply rounded to the nearest integer. This implicitly sets the ternarization threshold at $0.5 \cdot \gamma$, dynamically adapting it to the weight distribution of each layer rather than using a fixed heuristic. The full-precision weights $\mathbf{W}$ are scaled by γ rounded to the nearest integer, and then clipped to the range $[-1, +1]$ according to the

following equation:

$$\tilde{\mathbf{W}} = \text{RoundClip}\left(\frac{\mathbf{W}}{\gamma + \epsilon}, -1, 1\right) \tag{3.18}$$

where ϵ is a small constant for numerical stability, and the RoundClip function performs the rounding and clamping operations:

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))) \tag{3.19}$$

A crucial distinction is that this scaling factor γ is only used during the quantization step in training and is not applied during inference. The resulting weight matrix $\tilde{\mathbf{W}}$ consists purely of $\{-1, 0, +1\}$ values, allowing matrix multiplication to be performed almost entirely with integer addition and subtraction, without a final floating-point scaling operation. Furthermore, to simplify the pipeline, activations are quantized to 8-bits using a symmetric scheme that eliminates the integer zero-point z_x. The model is trained from scratch using this quantization scheme, enabling it to match the performance of full-precision models while drastically reducing computational and memory costs.

3.2.5 Trained Ternary Quantization (TTQ)

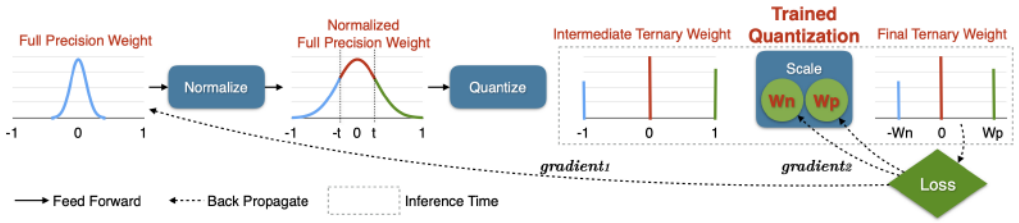


Figure 3.6: Trained Ternary Quantization pipeline. Credit [31].

Trained Ternary Quantization (TTQ)[31], is an advanced ternary Quantization Aware technique where network weights are constrained to an asymmetric set of three values: $\{-W_n, 0, +W_p\}$. The innovation of this method is that the scaling coefficients for negative weights W_n and positive weights W_p are independent, trainable parameters, allowing the network to learn optimal quantization levels for each layer.

During the forward pass, a latent full-precision weight tensor $\widetilde{\mathbf{w}}_l$ for layer l is quantized based on a layer-wise Δ_l threshold. A weight $\widetilde{w}_i$ is assigned to one

of the three values according to the rule of eq. 3.20.

$$w_i^t = \begin{cases} +W_p^l & \text{if } \tilde{w}_i > \Delta_l \\ 0 & \text{if } \mid \tilde{w}_i \mid \leq \Delta_l \\ -W_n^l & \text{if } \tilde{w}_i < -\Delta_l \end{cases} \tag{3.20}$$

While the threshold can be learned, the authors propose a practical heuristic for their experiments, setting it proportionally to the maximum absolute value of the latent weights in the layer:

$$\Delta_l = t \cdot max(\mid \widetilde{\mathbf{w}}_l \mid) \tag{3.21}$$

where t is a global hyperparameter.

As shown in Figure 3.6, the training process uses a Straight-Through Estimator (STE) with two distinct gradient paths to update the model parameters:

- **Updating Latent Weights** ($\widetilde{\mathbf{w}}_l$): The gradient with respect to a latent weight $\tilde{w}_i$ is a scaled version of the gradient from the corresponding quantized weight w_i^t. This scaling acts as a *learning rate multiplier*, allowing the latent weights to evolve and potentially change their ternary assignment across the threshold. The update rule for the gradient is:

$$\frac{\partial L}{\partial \tilde{w}_i} = \begin{cases} W_p^l \cdot \frac{\partial L}{\partial w_i^t} & \text{if } \tilde{w}_i > \Delta_l \\ 1 \cdot \frac{\partial L}{\partial w_i^t} & \text{if } |\tilde{w}_i| \leq \Delta_l \\ W_n^l \cdot \frac{\partial L}{\partial w_i^t} & \text{if } \tilde{w}_i < -\Delta_l \end{cases} \tag{3.22}$$

- **Updating Scaling Coefficients** (W_p^l, W_n^l): The scaling factors are updated by aggregating the gradients from all weights that were assigned to them. Let $I_p^l = \{i \mid \tilde{w}_i > \Delta_l\}$ be the set of indices for positive weights. The gradient for W_p^l is then the sum:

$$\frac{\partial L}{\partial W_p^l} = \sum_{i \in I_p^l} \frac{\partial L}{\partial w_i^t} \tag{3.23}$$

  A similar summation is performed for W_n^l over the set of negative-assigned weights.

The dual-path learning allows the model to simultaneously optimize the ternary assignments via $\widetilde{\mathbf{w}}_l$ and the ternary values themselves via W_p^l and W_n^l. By allowing the model to learn distinct magnitudes for positive and negative values, this asymmetric approach can more faithfully represent the original

weight distribution compared to symmetric methods, resulting in a significant improvement in model performance.

Chapter 4

Methodology

4.1 Onboard Cloud-Aware Compression

This thesis implements and evaluates an onboard data processing pipeline that integrates cloud screening with an adaptive compression scheme. The entire framework is designed to be compliant with the CCSDS 123.0-B-2 standard, leveraging its low-complexity, predictive coding architecture, which was detailed in Chapter 2. The primary goal is to significantly reduce data volume by applying higher compression to uninformative cloudy regions while preserving the scientific quality of clear-sky pixels through near-lossless or lossless encoding.

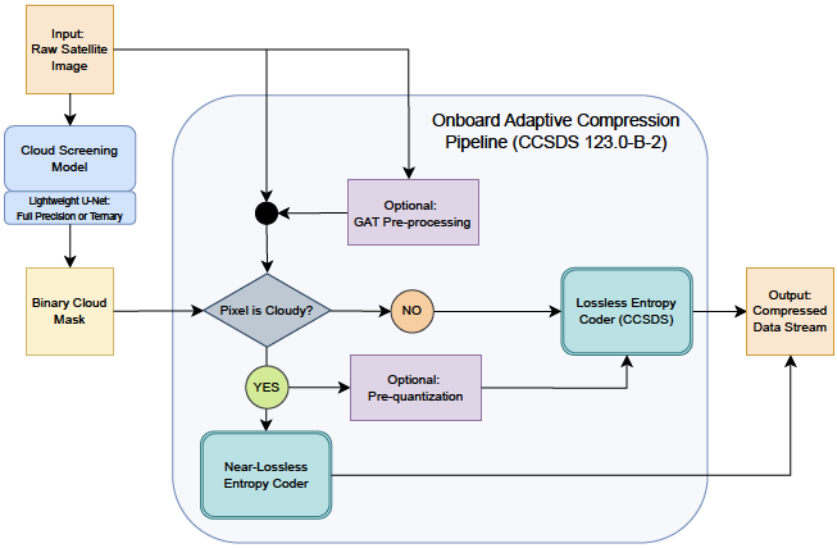


Figure 4.1: Cloud screening and compression pipeline.

To achieve this, the methodology combines two key stages: a deep learning-based cloud screening model to generate pixel-level masks, and conditional pre-processing techniques to be used before feeding the input images and the

cloud masks to the CCSDS compressor. The developed pipeline is showed in Figure 4.1.

4.1.1 Pre-processing techniques

Various pre-processing techniques exist to improve compression efficiency. This study focuses on two main approaches applied prior to the final entropy coding stage.

As an optional pre-processing step, the use of the **Generalized Anscombe Transformation (GAT)** [32] to stabilize the signal-dependent noise characteristic of optical satellite sensors is investigated. The goal of a variance-stabilizing transform like GAT is to convert the input signal into a domain where the noise is approximately constant and additive. This allows a simple, uniform quantizer to operate more efficiently across the entire dynamic range of the image. The decompressor mirrors the compressor's pipeline: an inversion of the GAT is mandatory if using it in the compression stage.

Another optional pre-processing technique to be used before the lossless compression stage (and consequently after the decompression) is a pre-quantization algorithm. It enables a near-lossless compression mode operating band-by-band by applying a quantization step size $Q_z = 2 * \delta_z + 1$, where δ_z is a user-defined parameter for spectral band z. This step is applied conditionally at the pixel level, guided by the binary cloud mask generated by the Lightweight U-Net model, described in Section 3.1.4, and further modified through this work: for pixels identified as non-cloudy, a fine quantization step is used, for pixels flagged as cloudy, a different, more aggressive quantization step, can be used. This is justified as cloudy regions contain less useful information for most downstream tasks, allowing them to be compressed more aggressively. Through this method it is possible to use a lossless compressor even if actually performing a near-lossless compression.

4.2 Cloud screening

To generate the cloud masks required by the adaptive compression pipeline, this thesis employs different cloud screening techniques, all based on the Lightweight U-Net described in section 3.1.4. The model was chosen for its high computational efficiency and small memory footprint, making it suitable for resource-constrained onboard environments. Its key features enable it to perform accurate cloud segmentation with a parameter count of only 20k. This model is trained to perform binary segmentation, classifying each input pixel as either *cloud* or *non-cloud*. The proposed techniques focus on training and testing the network,

both with full precision and quantized weights.

4.3 Training and inference techniques

As detailed in Chapter 2, the Sentinel-2 Multi-Spectral Instruments (MSI) collect images using pushbroom sensors which acquire data sequentially, as the spacecrafts move along their orbit. This acquisition mode, along with the stringent memory and computational constraints of satellite hardware, renders full-frame image processing impractical for real-time, onboard applications.

To address these challenges, this thesis implements and evaluates two memory-efficient processing strategies build for onboard cloud screening. These methods align directly with the natural data flow of the sensor and enable the use of deep learning models in resource-constrained environments. Their primary advantages include reduced memory footprint, simplified integration with existing data pipelines, and the potential for parallelized computation.

The two strategies adopted are *slice-based* and *sliding window* approaches, both allowing partial, incremental inference that fits within the hardware constraints. They are described in the following paragraphs while the obtained results are presented in Chapter 5.

4.3.1 Slice-based

In the slice-based approach, the image is divided into non-overlapping patches (slices or chunks), each of which is processed independently by the cloud screening network. The number of slices is customable and it is selected by choosing a constant offset.

By processing smaller portions of the image, slice-based training and inference help prevent memory overflow that could result in data loss or failed predictions. To ensure consistency between training and inference, the network is trained on the same type of slices used during the testing phase, without changing the offset. Figure 4.2 displays an example of the slice-based approach to generate cloud masks. The image is divided in 16 slices using an offset of 32 lines and the Cloud Probability Threshold (CPT) is fixed to 0.7. The generated cloud mask is compared to the ground truth and to the prediction of the full precision model trained on the entire image and tested following a slice-like method.

A limitation of this approach is the lack of contextual overlap between slices, which degrades accuracy near stripe boundaries where objects or clouds are partially visible, especially when the images provided to the network during training is made by few lines (in case of smaller offsets). The network, made

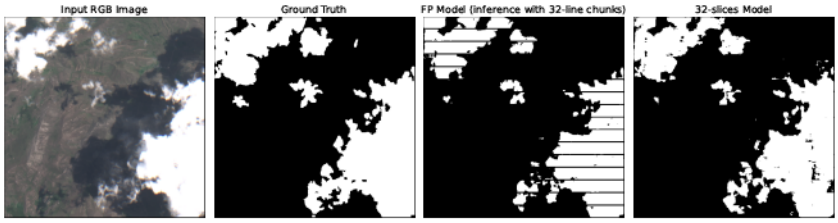


Figure 4.2: From left to right: the RGB composite, the ground truth cloud map, the FP model trained without using slices and tested using slice-like approach, the FP model trained and tested using slice-like approach. The Cloud Probability Threshold (CPT) used for cloud detection precision is 0.7 and the number of slices used is 32.

by relatively few parameters, is not able to capture the connections between the features when the stripes are small.

To improve the accuracy of the network when the slices are too small it is possible to use the sliding window approach, described in the following section.

4.3.2 Sliding window

The sliding window approach helps the network to bypass the problem of small spatial context in the image by processing overlapping patches that shift across the image with a fixed stride, smaller than the window size. Each pixel is thus seen in multiple contexts, and the final prediction is obtained by averaging the results from all overlapping windows. This method reduces boundary artifacts and improves prediction quality as each pixel is evaluated multiple times with slightly shifted context. With respect to the slice-based approach, the sliding window method increases inference time and memory consumption due to redundant processing of overlapping areas. Testing is also performed using sliding windows to allow prediction consistency and avoid performance degradation caused by differences in the training and inference conditions.

Figure 4.3 shows an example of cloud masks generated using the sliding window approach. The window size is fixed to 32 and the Cloud Probability Threshold (CPT) to 0.5. The ground truth cloud mask is compared to the sliding window approach's cloud mask generated with different shifts (4 and 16) and with the full precision model's prediction when trained using the standard approach (no windows).

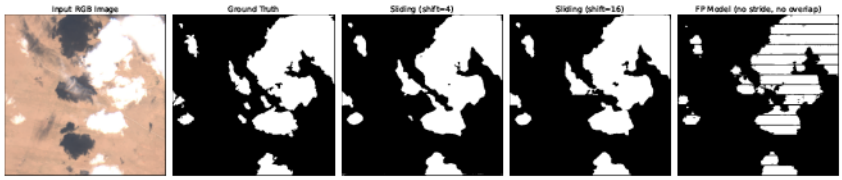


Figure 4.3: From left to right: the RGB composite, the ground truth cloud map, the FP model trained using sliding windows of 32 lines shifted by 4 and 16 lines and finally the sliding window approach used only for testing the FP model. The Cloud Probability Threshold (CPT) used for cloud detection precision is 0.5 and the number of slices used is 32.

4.4 Ternarization methods

To adapt the Lightweight U-Net model for deployment on satellite hardware, this thesis investigates several ternarization methods based on Quantization-Aware Training (QAT). Those methods are used to reduce the computational complexity of neural networks by constraining weight values to a limited set. This approach frees the hardware from the need of multipliers, as multiplications can be replaced by simple additions and subtractions and skipped entirely in the case of zero weights.

To ensure a fair comparison, each quantization method is applied to the same baseline Lightweight U-Net architecture (Section 3.1.4), which utilizes separable convolutional layers, batch normalization, and ReLU activations. All quantized models are trained using consistent optimizers, loss functions, and training schedules, allowing for a direct performance comparison against each other and the full-precision baseline.

The ternarization methods implemented for the purpose of the thesis are listed below.

- **Modified TTQ**: This thesis implements a modified version of the Trained Ternary Quantization (TTQ) method from [31]. TTQ is implemented by extending PyTorch's *nn.Conv2d* module to support ternary quantization using asymmetric scaling factors. Each custom convolutional layer uses two learnable parameters, W_p and W_n, initialized to 1.0 and updated during training via standard gradient descent. During the forward pass, the ternarization threshold for each layer is computed as $\Delta = t \cdot \max(|w|)$,where t is a global hyperparameter fixed at 0.05, as it is done in the original TTQ method. A Straight-Through Estimator (STE) is used to approximate the gradients during backpropagation, allowing the updates to be applied to a latent full-precision copy of the weights, which are also used to initialize the model. Using the STE, the quantized

weights are detached from the computation graph and combined with the original weights such that gradients flow only through the latent variables. The final quantized weights used in inference thus take values from $\{-W_n, 0, W_p\}$, and the full-precision weights are discarded post-training.

While this implementation preserves the main concepts of the TTQ method, such as asymmetric scaling, layer-wise thresholding, STE-based training, and latent full-precision weight updates, the two methods are not exactly alike. First, in the original paper the weights are normalized to the range $[-1, +1]$ prior to thresholding, ensuring consistent scale across layers. This step is omitted in this thesis' implementation, where the threshold is instead directly computed from the unnormalized full-precision weights. Second, in the original TTQ, the gradients for the scaling coefficients W_p and W_n are computed only over the subsets of weights mapped to +1 and -1, respectively, while this implementation relies on PyTorch's automatic differentiation to update these parameters globally, based on the total loss. Third, the manual scaling of gradients for the latent weights, which depends on their ternary assignment as specified in the original TTQ paper, is not applied in this implementation. Finally, unlike the optional constant sparsity enforcement in the original work, this implementation allows the layer-wise sparsity to be learned naturally during training.

For the experimental evaluation, this method is referred to as *TTQ + FP + STE* but other combinations are tested to identify the impact of different components: *TTQ + noFP + STE* uses the modified TTQ with STE described above but with random weight initialization instead of pre-trained full precision weights. *TTQ + FP + noSTE* uses FP weights initialization without the STE during backpropagation and *TTQ + noFP + noSTE* uses weights randomly initialized and does not use the STE approach.

- **Symmetric modified TTQ**: A symmetric variant of TTQ is also implemented, keeping the modifications with respect to the original TTQ paper described above. The quantization process maps full-precision weights to ternary values, but the scaling factor is shared for both positive and negative weights such that $W_p = W_n = W$. This simplification reduces the number of trainable parameters per layer while preserving the benefits of ternarization. The ternarization threshold is computed in the same way, using a fixed scaling factor t multiplied by the maximum absolute weight value, but the quantized weights take values from the symmetric set {-W, 0, +W}. During training, the unique scaling coefficient W is treated as a

learnable parameter and it is optimized through the full-precision weights using standard backpropagation. As in the asymmetric case, ternarized weights are used in the forward pass, while latent full-precision weights are updated via the Straight Through Estimator. The symmetric formulation simplifies the deployment pipeline but reduces model accuracy due to the constraint of equal magnitude for positive and negative ternary weight values.

This method is referred as *symTTQ + FP + STE* in the experimental results, and other variants are not tested.

- ***Absmean* ternarization**: This symmetric ternary quantization approach is inspired by the *absmean* quantization function introduced in the BitNet b1.58 paper [30]. The quantization rule is implicit and requires no additional learnable parameters. The process involves two steps:
  - Quantization: Full-precision weights are first scaled by an implicit, layer-wise factor $s = 1/mean(|\ w\ |)$, rounded to the nearest integer, and clipped to the ternary set {-1, 0, +1}.
  - Rescaling: The resulting ternary weights are then rescaled by multiplying by $1/s$ to return them to the original dynamic range.

  Unlike TTQ, which relies on separate trainable scaling factors for positive and negative weights, this symmetric approach uses a single scaling derived from the layer's own statistics, simplifying the optimization process. A crucial distinction must be made about the inference process. The original BitNet b1.58 performs inference using pure {-1, 0, +1} weights. In contrast, this thesis implements the absmean rule in the style of Ternary Weight Networks (TWN), where the scaling factor is re-applied. The final weights used for inference are therefore $\{-\gamma, 0, +\gamma\}$, where $\gamma = mean(|\ w\ |)$. Training is conducted using a STE to update the latent full-precision weights.

  This technique is referred as *absmean + STE* o simply *absmean*. Another version of the model is tested, *absmean + noSTE*, which does not use the Straight Through Estimator.

- **Per-channel asymmetric TTQ**: This method extends the modified Trained Ternary Quantization (TTQ) by performing per-output-channel ternarization. Unlike the original TTQ approach, which uses a single pair of scaling coefficients (W_p, W_n) per layer, this method assigns a distinct pair of learnable scaling factors to each output channel of a convolutional layer: each output channel i is associated with its own

$W_p^{(i)}$ and $W_n^{(i)}$, allowing the quantized weights to adapt more precisely to channel-wise statistics. The ternarization threshold Δ_i is computed individually for each output channel as $\Delta_i = t \cdot \max(|w_i|)$, where w_i represents the full-precision weights of the i-th output channel and t is a global hyperparameter fixed to 0.05, consistent with the original TTQ paper and its modified simpler version. The full-precision weights are mapped to $-W_n^{(i)}$, 0, or $+W_p^{(i)}$ depending on whether they fall below $-\Delta_i$, within the threshold, or above $+\Delta_i$, respectively. During training, the latent full-precision weights are updated using the Straight-Through Estimator (STE), while the per-channel scaling coefficients are optimized through standard backpropagation. The channel-wise adaptation introduces additional degrees of freedom into the quantization scheme and it is particularly helpful in deep convolutional networks where feature distributions vary considerably across channels.

This approach is referred as *perchannelTTQ* in the following paragraphs.

- ***Absmean* ternarization with distillation loss**: This method integrates multiple strategies including knowledge distillation and a symmetric ternarization function based on the *absmean* rule. The quantized model uses a simple quantization function that scales the full-precision weights by the inverse of their mean absolute value, rounds and clips the result to the ternary set {-1, 0, +1}, and then rescales the quantized weights back to their original range, as it is done in the *absmean* ternarization. During training, the full-precision latent weights are retained and updated via the Straight-Through Estimator (STE), enabling gradients to flow through the ternarized operations. A pre-trained full-precision lightweight U-Net model serves as a *teacher* in a knowledge distillation setup, and the quantized model (called *student*) is trained to match both the ground truth and the teacher's soft predictions. The total loss function is a weighted combination of binary cross-entropy (with the labels) and mean squared error (between softened logits of teacher and student), modulated by a temperature parameter. Higher temperatures make the output distribution softer (less confident), helping the quantized student learn from the relative probabilities of the teacher, not just the hard binary predictions. This method is introduced by Hinton et al. in [33] and, for the purpose of this work, it is referred as *absmean + distillation loss*.

- **TTQ model using a warmup strategy**: To improve training stability, a warmup strategy is integrated into the standard TTQ framework. For an initial number of epochs, the model is trained using its original full-precision weights, allowing it to converge to a reasonable state before

the potentially disruptive effects of quantization are introduced. This allows the network to adapt before ternarization is activated. At the end of the warmup period, the forward pass switches to a ternary mode, where weights are projected to $\{-W_n, 0, W_p\}$ based on a fixed threshold. The quantized weights are detached from the computational graph and combined with the original latent weights using the Straight Through Estimator (STE), ensuring proper gradient flow during backpropagation. This method is referred as *TTQ + warmup* in the experimental results.

- **TTQ model with warmup and annealed threshold**: This approach enhances the warmup strategy by incorporating a time-varying (annealed) delta threshold. Rather than using a static threshold for all epochs, a threshold Δ is computed dynamically based on training progress. It starts from a relatively large initial value and gradually decays towards a smaller final threshold during the training process, implementing a linear interpolation schedule. At each forward pass, the threshold is recalculated as a function of the current epoch and the maximum absolute weight in the layer, modulated by a multiplicative factor. This dynamic adjustment reduces the model's complexity in early stages, while progressively refining its quantized representations. The ternarization itself is applied using a symmetric masking rule: weights above $+\Delta$ are assigned $+W_p$, those below $-\Delta$ to $-W_n$, and the rest are mapped to zero. This method is referred as *annealed TTQ + warmup*.

Chapter 5

Experimental Results

This Chapter focuses on the Experimental Results collected to validate the proposed methods discussed in Chapter 4 and their analysis, along with a description of the dataset, the experimental framework and the performance metrics used to perform the tests.

5.1 The Dataset

The development and evaluation of the algorithms in this thesis is founded on the CloudSEN12 dataset[34], which is used for both training and testing the models. CloudSEN12 is a large-scale, globally distributed dataset developed to improve the semantic understanding of clouds and cloud shadows in Sentinel-2 imagery. It is composed by 49,400 image patches (IPs), each covering a 5.090×5.090 meter area, distributed across diverse geographical regions to allow model generalization. Each patch includes a rich combination of satellite data: Sentinel-2 Level-1C (Top-Of-Atmosphere) and Level-2A (Surface Reflectance), Sentinel-1 SAR imagery, Digital Elevation Models (MERIT), land cover maps, and auxiliary data such as water occurrence and solar geometry.

CloudSEN12 is designed to aid the training of deep learning models under supervised, semi-supervised, and weakly-supervised settings and it provides three levels of annotation quality: high-quality pixel-level annotations (10.000 IPs), scribble annotations (10.000 IPs), and no annotations (29.250 IPs). This work focuses on the high-quality annotated images, which categorize pixels into four semantic classes: clear, thick cloud, thin cloud, and cloud shadow. The reliability of these labels is ensured through rigorous protocols involving active learning tools such as *IRIS*. For the binary classification task at of this thesis, the labels are grouped into two superclasses: cloud (thick cloud) and non-cloud (clear sky, thin cloud and cloud shadow). Each of the 10.000 used patches is downsampled to a 509 × 509 pixel image, padded to 512x512, and used for

both training and evaluating the models.

5.2 Experimental Framework

The Lightweight U-Net used with full-precision or quantized weight is trained using CloudSEN12 spectral bands B2, B3, B4, B8 and B10, with pixels normalized as Top-of-Atmosphere (TOA) (Sentinel-2 Level-1C) and padded to 512x512 pixels. The networks is trained on a GPU Quadro RTX 6000, using a batch size of 32 for 50 epochs. The chosen optimazer is Adam, with an initial learning rate of 0.001, while the used loss function is the Binary Cross Entropy. The training set and the test set are respectively made by approximately 9000 and 1000 images.

The Cloud Probability Threshold (CPT), which is applied to the sigmoid to determine the cutoff above which a pixel is classified as cloudy, is changed in the range $[0, 1]$ to perform the tests and compare the different models' performance.

5.3 Evaluation Metrics

The performance of the Deep Learning algorithms are evaluated using classical performance metrics.

- **True Positive Rate (TPR)**: the conditional probability that the algorithm declares positive given the sample is truly positive. Also addressed as Sensitivity or Recall.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = P(\,\text{predicted cloud} \mid \text{true cloud}\,) \tag{5.1}$$

- **False Positive Rate (FPR)**: the conditional probability that the algorithm produces a false alarm on a truly negative sample.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = P(\,\text{predicted cloud} \mid \text{true clear}\,) \tag{5.2}$$

- **Precision (Positive Predicted Value)**: it measures the reliability of a positive prediction.

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = P(\,\text{true cloud} \mid \text{predicted cloud}\,) \tag{5.3}$$

- **Balanced Overall Accuracy (BOA)**: it averages class-wise accuracies, making it insensitive to class imbalance (useful if cloud pixels are only a

few percentage of an image).

$$\text{BOA} = \frac{1}{2}\Big(\text{TPR} + \text{TNR}\Big) = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{TN}+\text{FP}}\right) \tag{5.4}$$

where the True Negative Rate is evaluated as $\text{TNR} = \frac{\text{TN}}{\text{TN}+\text{FP}}$.

- **Intersection-over-Union (IoU)**: also called Jaccard Index, measures the overlap between the predicted cloud mask and the ground truth, penalizing both misses and false alarms.

  This index ranges between 0 (no similarity) to 1 (identical sets).

$$\text{IoU} = \frac{\text{TP}}{\text{TP}+\text{FP}+\text{FN}} \tag{5.5}$$

5.4 Stripes and Sliding-window approach

To enable efficient training and inference on resource-constrained platforms such as onboard satellite systems, this work adopts two alternative strategies to process input images: the slice-based and sliding window approaches. These methods are useful since satellites typically acquire images line by line using pushbroom sensors, making it natural to process stripes, as deepened in section 4.3.

Moreover, using smaller input chunks mitigates memory overload risks that could otherwise result in loss of data during inference. In the following, models trained and evaluated using different slicing configurations are compared, as well as sliding window setups with various window sizes and shifts. Figure 5.1 shows the effect of different image slicing during model training on the resulting cloud mask. The top-left image shows the RGB composite of the input image and the Ground Truth Cloud Mask is below it, the top-center image presents the prediction obtained from the Full Precision (FP) model trained using a single 512-line chunk (traditional approach, without slices), with inference performed using 32-line chunks. The other three images show the predictions generated by FP models trained and tested on sliced input images of different chuck heights: 32-line slices (top-right), 16-line slices (bottom-center), and 8-line slices (bottom-right). All predictions are generated using a fixed CPT of 0.5. When performing inference on the full precision model trained traditionally, the output appears visually degraded. When the image is trained using slices, the cloud masks appear more loyal to the ground truth image.

Furthermore, by looking at Table 5.1 it is possible to compare different models trained using different numbers of slices, also changing the CPT: the 32-slices model results to be the best performing in terms of TPR and BOA

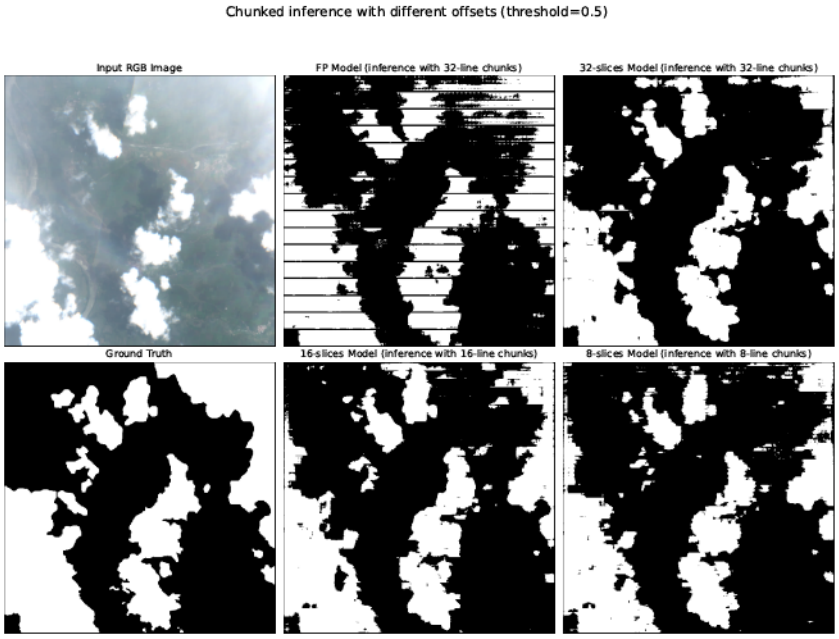


Figure 5.1: Slices approach: the first column shows the RGB composite and the corresponding Ground Truth, then the predicted cloud masks obtained from Full Precision models trained and evaluated using different slicing strategies. All results are generated with a fixed CPT of 0.5.

which are even higher than the one of the model trained using 512x512 pixel images. On the other hand, the 16-slices model is the worse-performing in terms of TPR but the best in terms of Precision.

Another application consists in training the Full Precision model using a sliding window approach. During both training and inference, the model is presented with a limited vertical portion of the full image (a window) that slides over the entire height of the image with a fixed step, also called *shift*. Through this method, the network sees similar spatial contexts across the dataset, learning more robust local features with respect to the previous approach: when slicing the images without sliding the window, the context provided to the network totally changes from slice to slice, as the network is trained simply using smaller images, which does not improve the learning process.

During inference, predictions from overlapping windows are aggregated by averaging to produce a coherent full-image prediction: for each window, the model predicts a partial cloud mask, and all overlapping predictions are accumulated. A weight mask tracks the number of times each pixel is covered, allowing for proper averaging. For this reason, the final prediction results smoother than in the case of the slices approach. After averaging the overlapping outputs the final binary mask is obtained properly setting the CPT to obtain the desired level of accuracy in cloud detection.

Figure 5.2 illustrates the results obtained using the sliding window approach

Table 5.1: Performance Metrics of Full-Precision Models Using Different Slicing Strategies and Cloud Probability Thresholds (CPT).

Slicing Strategy	CPT	TPR	FPR	Precision	BOA	IoU
Full Image (1 Slice, 512 Offset)						
	0.5	0.867	0.036	0.905	0.915	0.795
	0.7	0.777	0.015	0.955	0.881	0.750
	0.9	0.623	0.003	0.988	0.810	0.619
16 Slices (32 Offset)						
	0.5	0.847	0.028	0.924	0.909	0.792
	0.7	0.768	0.013	0.961	0.877	0.744
	0.9	0.628	0.003	0.988	0.813	0.623
32 Slices (16 Offset)						
	0.5	0.888	0.051	0.877	0.918	0.789
	0.7	0.817	0.025	0.931	0.896	0.770
	0.9	0.675	0.006	0.980	0.835	0.666
64 Slices (8 Offset)						
	0.5	0.866	0.046	0.885	0.910	0.778
	0.7	0.791	0.023	0.934	0.884	0.749
	0.9	0.650	0.007	0.976	0.822	0.640

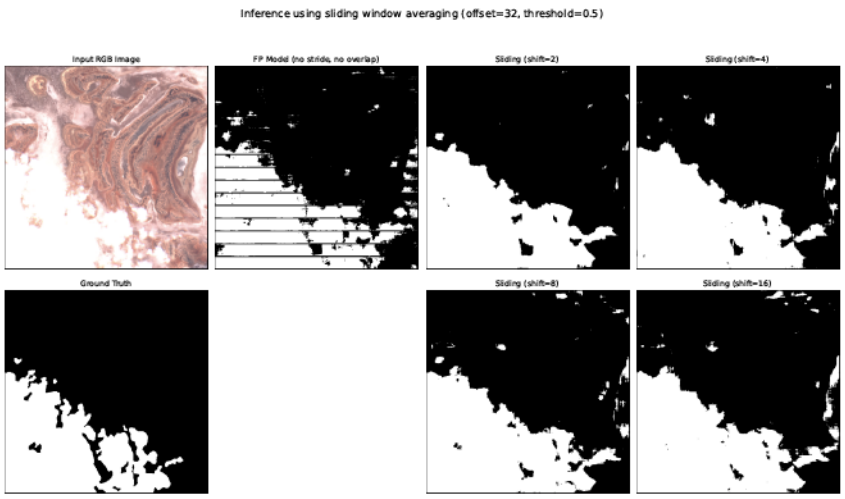


Figure 5.2: Sliding window approach with window size=32: cloud mask predictions from Full Precision model using different shift values (2, 4, 8, 16). Fixed CPT of 0.5.

with a Full Precision model and a window size of 32 lines. The top-left image shows the RGB composite of the input image and the Ground Truth mask is below it. The top-center image displays the 32-lines inference output generated using a model trained on the full image without overlapping windows. The remaining images show the results of inference performed with sliding window shifts of 2, 4, 8 and 16 lines and a fixed CPT of 0.5.

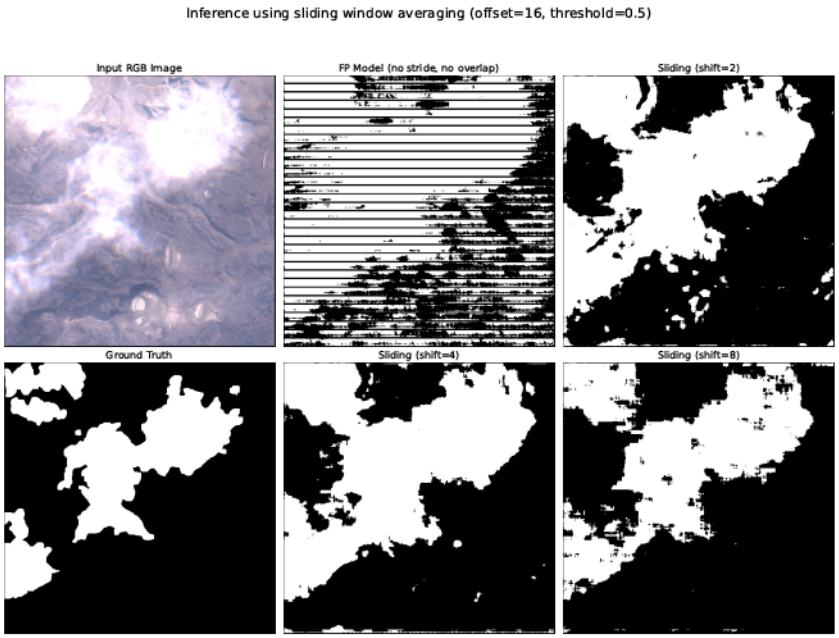


Figure 5.3: Sliding window approach with window size=16: cloud mask predictions from Full Precision model using different shift values (2, 4, 8). Fixed Cloud Probability Thresholds (CPT) of 0.5.

Similarly, Figure 5.3 and Figure 5.4 show the results obtained using the sliding window approach with a Full Precision model and a window size of 16 and 8 lines, respectively. In both cases the top-left image shows the RGB composite of the input image and the Ground Truth mask is shown below it. The top-center image displays the 16-lines and 8-lines, respectively, inference output generated using a model trained on the full image, without overlapping windows. The remaining images show the results of inference performed with sliding window shifts of 2, 4 and 8 lines in the first case and 2 and 4 lines in the second. The threshold used to establish how precisely a detected cloud has to be labels as such is fixed to 0.5. Compared to the corresponding images generated with a 32-lines window size, the results appear degraded in both cases.

The overall best sliding window model, in terms of performance is the one having window size equal to 32 and a shift of 2 lines, allowing improved prediction smoothness and the best spatial continuity in the final cloud mask

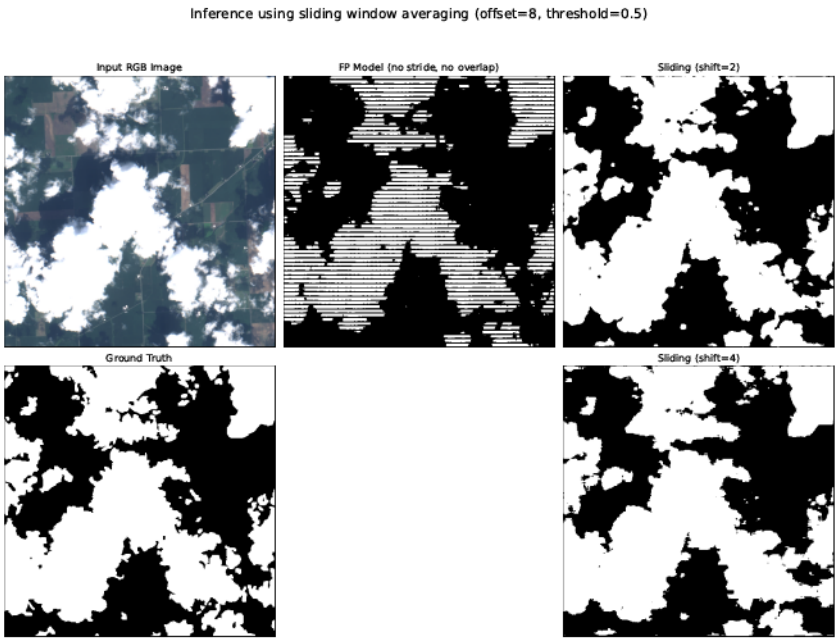


Figure 5.4: Sliding window approach with window size=8: cloud mask predictions from Full Precision model using different shift values (2, 4). Fixed Cloud Probability Thresholds (CPT) of 0.5.

with respect to the other approaches. However, it is necessary to account for the trade-off between the precision of the model in predicting the cloud mask and the computational effort needed to process overlapping windows. Smaller shift values lead to increased inference time and higher resource consumption.

Table 5.2 shows the results of the sliding window experiments, offering a comparison across three window sizes (32, 16, and 8 lines) and their corresponding shift values. The performance is evaluated at high-confidence Cloud Probability Thresholds (CPT) of 0.7 and 0.9. Several key trends can be identified from this analysis. First, for any given window size, a smaller shift, which results in a greater overlap, consistently improves segmentation accuracy. This is most evident in the 32-line window configuration, where reducing the shift from 8 lines to 2 lines at a CPT of 0.7 improves the IoU of 3%. This trend holds because smaller shifts allow each pixel to be evaluated more frequently and within a greater variety of contexts, which smooths out prediction errors and increases spatial consistency. Second, the impact of the window size itself shows a more complex relationship. While larger windows are generally expected to perform better due to a wider field of view, the results show that the 8-line window with a 4-line shift achieves the highest IoU (0.783) and Balanced Overall Accuracy (0.907) among all tested configurations at CPT=0.7. This counterintuitive result suggests that for the Lightweight U-Net architecture, a smaller, more focused window may be more effective at learning fine-grained

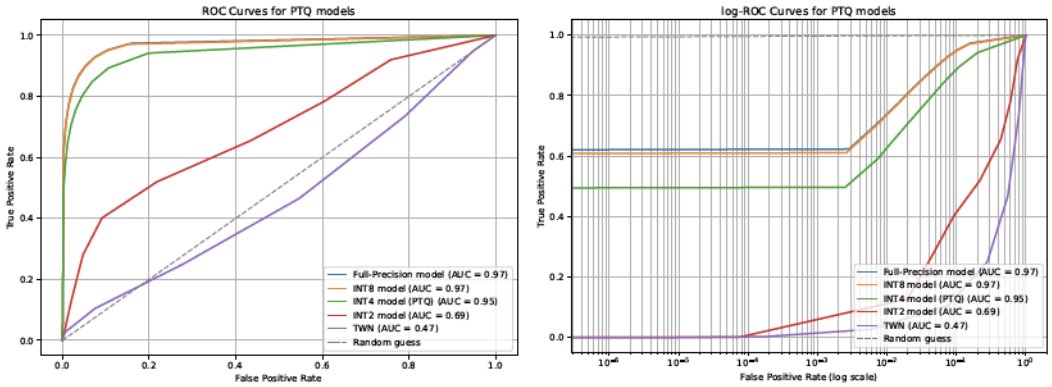
cloud features, while larger windows might introduce noise or irrelevant contextual information that slightly worsen performance. As expected, the effect of the CPT is consistent across all configurations. Increasing the CPT from 0.7 to 0.9 invariably leads to a lower False Positive Rate (FPR) at the cost of a lower True Positive Rate (TPR). This demonstrates the fundamental trade-off between minimizing false alarms, preserving clear-sky data, and maximizing cloud detection sensitivity.

Table 5.2: Performance Metrics for the Sliding Window Approach with Different Window Sizes, Shifts, and Cloud Probability Thresholds (CPT).

Window Size	Shift	CPT	TPR	FPR	Precision	BOA	IoU
32 lines	2 lines	0.7	0.784	0.014	0.957	0.885	0.757
		0.9	0.645	0.004	0.986	0.821	0.640
	4 lines	0.7	0.720	0.009	0.969	0.855	0.704
		0.9	0.586	0.002	0.990	0.792	0.582
	8 lines	0.7	0.738	0.009	0.970	0.865	0.722
		0.9	0.600	0.002	0.992	0.799	0.597
	16 lines	0.7	0.784	0.013	0.961	0.886	0.760
		0.9	0.637	0.003	0.989	0.817	0.633
16 lines	2 lines	0.7	0.765	0.012	0.963	0.876	0.743
		0.9	0.630	0.003	0.989	0.814	0.626
	4 lines	0.7	0.789	0.016	0.952	0.886	0.758
		0.9	0.657	0.004	0.985	0.826	0.651
	8 lines	0.7	0.713	0.009	0.970	0.852	0.698
		0.9	0.588	0.002	0.990	0.793	0.585
8 lines	2 lines	0.7	0.814	0.020	0.942	0.897	0.775
		0.9	0.663	0.004	0.986	0.829	0.657
	4 lines	0.7	0.845	0.032	0.914	0.907	0.783
		0.9	0.714	0.008	0.973	0.853	0.700

5.5 Lightweight U-Net: Post-training Quantization

Many quantization methods have been applied to the Lightweight U-Net described in section 3.1.4. The Post-Training Quantization techniques used are INT8, INT4, INT2 and TWN. Figure 5.5 displays the ROC curves for the different PTQ techniques, showing that the Area Under the Curve (AUC) increases while increasing the precision of the model, reaching the same results



(a) ROC curve.

(b) log-ROC curve.

Figure 5.5: ROC curves for Post-Training Quantization using different cloud detection thresholds and different precision models (INT8, INT4, INT2, TWN).

of the Full Precision model, whose weights are saved as 32-bit floating-point numbers, by mapping them to signed 8-bit integers in the range $[-128, 127]$. The accuracy of the INT8 model is very similar to the FP model's one, offering 4x memory savings. In the INT4 model weights are quantized to values in the range $[-8, 7]$, providing 8x memory savings over FP32 but providing a degradation in the performance. On the other hand, if using INT2 models, weights are highly compressed and they can take four possible values: $[-2, -1, +1, +2]$. The accuracy of the model is severely dropped.

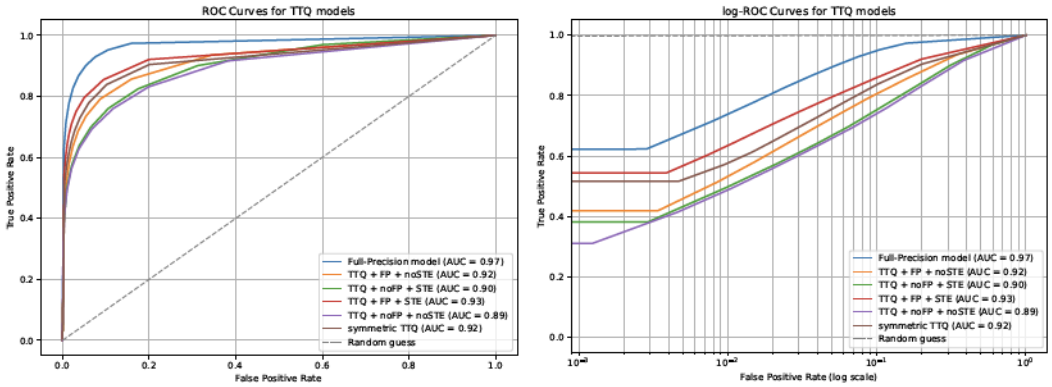
Figure 5.5b shows log-ROC curved which allow a more accurate analysis on the points where the FPR is low, where false detections can significantly impact the usability of the data.

While the INT8 and INT4 models work really well, the TWN, implemented as described in section 3.2.2, behaves as a random guess, showing the worst results for the PTQ models. This happens because the network is trained using FP32 weights and then tested using only three values for the weights: $-1, 0, +1$. The approximation is too tight and the outputs are close to random.

TWN is the only method, among the tested Post-Training Quantization's ones, to use ternary weights and it is outperformed by the Quantization-Aware methods for ternarization described in the next section.

5.6 Lightweight U-Net: Quantization-Aware Models for Ternarization

This section presents a comprehensive evaluation of the Quantization-Aware Training (QAT) methods described in Section 4.4. The analysis is structured into three parts: an overall performance comparison using ROC curves, a detailed quantitative analysis of key metrics, and a qualitative visual evaluation



(a) ROC curve. (b) log-ROC curve.

Figure 5.6: ROC curves for the baseline TTQ models, highlighting the importance of using both a Straight-Through Estimator (STE) and pre-trained Full-Precision (FP) weights for initialization.

of the generated cloud masks.

5.6.1 Overall Performance via ROC Analysis

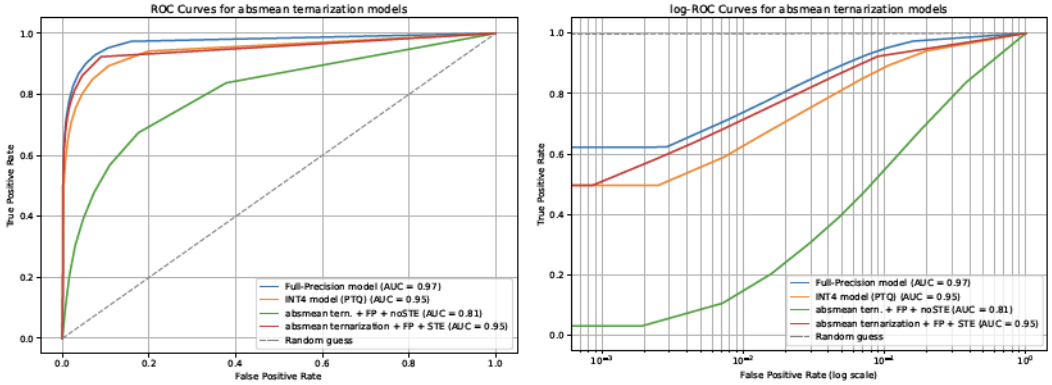
The ROC curves shown in Figure 5.6, 5.7 and 5.8 highlight the key differences between the ternarization models. For each figure, the left plot represents the complete ROC curve (plot of the True Positive Rate (TPR) against the False Positive Rate (FPR)), while the right plot represents log-ROC curves, where the False Positive Rate (FPR) is plotted on a logarithmic scale. This visualization is particularly useful as it expands the low-FPR region, allowing for a clearer comparison of model performance in this critical zone. Minimizing false alarms is crucial in operational scenarios to avoid the erroneous, aggressive compression of valuable clear-sky data.

The Area Under the Curve (AUC) is also reported for each case, providing a single scalar value summarizing overall classification performance. It is calculated as

$$\mathrm{AUC} = \int_0^1 \mathrm{TPR}(\mathrm{FPR})\, d(\mathrm{FPR}) \approx \sum_{i=1}^{n-1} (\mathrm{FPR}_{i+1} - \mathrm{FPR}_i) \cdot \frac{\mathrm{TPR}_{i+1} + \mathrm{TPR}_i}{2} \quad (5.6)$$

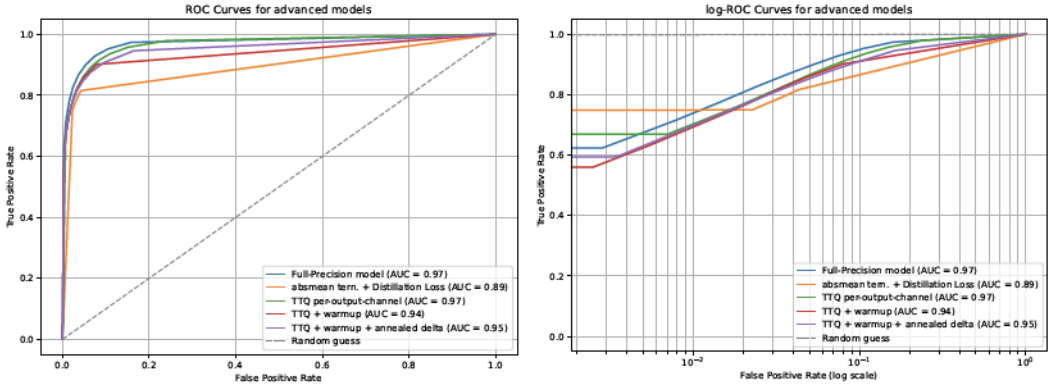
.

Figure 5.6 evaluates the baseline Modified TTQ and its derivations. The results clearly demonstrate the value of both STE, for stable training, and FP initialization for providing a strong starting point. The TTQ + FP + STE model (AUC = 0.93) significantly outperforms all other combinations, establishing it as the most robust of the simple TTQ variants. The symmetric TTQ approach performs almost as well as the corresponding asymmetric version (AUC = 0.92) only but slightly lagging, highlighting the benefit of learning



(a) ROC curve. (b) log-ROC curve.

Figure 5.7: ROC curves comparing the absmean ternarization method against the Full-Precision and INT4-PTQ baselines.



(a) ROC Curve. (b) log-ROC curve.

Figure 5.8: ROC curves for the advanced QAT techniques, demonstrating the highest performance levels among the quantized models.

independent scaling factors.

Figure 5.7a compares the absmean ternarization against similar models. The absmean + STE model (AUC = 0.95) achieves performance nearly identical to the INT4-PTQ model. However, its superiority is evident by looking at the log-ROC curve (Figure 5.7b), which shows that the model maintains higher TPRs than the INT4-PTQ model, making it a more suitable choice for operational deployment. The plot also reaffirms the critical role of the STE, as the absmean + noSTE model's performance collapses (AUC = 0.81).

Figure 5.8 shows the more advanced QAT strategies. The per-channel TTQ model emerges as the clear top performer, achieving an AUC of 0.97, virtually indistinguishable from the Full-Precision baseline. This highlights the substantial benefit of learning channel-wise scaling factors, which allows the model to adapt to the unique statistical properties of each feature map. The distillation and annealed delta methods also show strong performance (AUC = 0.89 and 0.95, respectively), proving to be effective optimization strategies.

5.6.2 Quantitative Analysis of Key Metrics

To provide a more granular comparison, Table 5.3 consolidates key performance metrics for the top-performing QAT models at CPTs of 0.5, 0.7, and 0.9. These thresholds represent low, medium, and high-confidence operating points, respectively.

Table 5.3: Performance Summary for Top Quantization-Aware Training Models.

Model	CPT	TPR	FPR	BOA	IoU
Full-Precision (Baseline)	0.5	0.831	0.022	0.905	0.788
	0.7	0.750	0.009	0.870	0.733
	0.9	0.608	0.002	0.803	0.604
INT4-PTQ (Reference)	0.5	0.877	0.051	0.918	0.789
	0.7	0.817	0.025	0.896	0.770
	0.9	0.675	0.006	0.835	0.666
TTQ + FP + STE	0.5	0.712	0.021	0.845	0.677
	0.7	0.643	0.011	0.816	0.626
	0.9	0.545	0.004	0.770	0.540
absmean + STE	0.5	0.726	0.011	0.857	0.707
	0.7	0.638	0.004	0.817	0.632
	0.9	0.497	0.001	0.748	0.496
per-channel TTQ	0.5	0.891	0.065	0.913	0.769
	0.7	0.820	0.033	0.894	0.759
	0.9	0.669	0.007	0.831	0.658

The quantitative results reinforce the findings from the ROC analysis. The per-channel TTQ model demonstrates remarkable robustness, achieving an IoU of 0.759 at a CPT of 0.7, a result that significantly outperforms other QAT methods and is competitive with the INT4-PTQ baseline, all while using a more hardware-friendly ternary representation. Notably, at a CPT of 0.9, per-channel TTQ maintains a very low FPR of 0.007 (0.7%) while still correctly identifying 66.9% of clouds, making it an excellent candidate for missions where preserving clear-sky data is the highest priority.

5.6.3 Visual and Qualitative Assessment

This analysis focuses on the visual quality of the generated cloud masks and the specific trade-offs each model makes as the Cloud Probability Threshold (CPT) is adjusted.

Visual Comparison of Cloud Masks

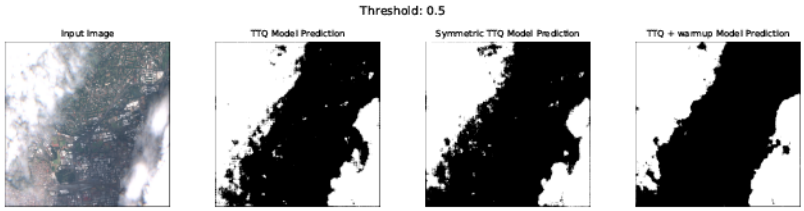


Figure 5.9: From left to right: input RGB composite, TTQ + FP + STE cloud mask, symmetric TTQ cloud mask, TTQ + warmup cloud mask with threshold=0.5.

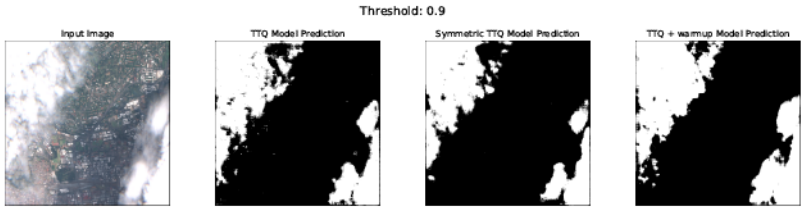


Figure 5.10: From left to right: input RGB composite, TTQ + FP + STE cloud mask, symmetric TTQ cloud mask, TTQ + warmup cloud mask with threshold=0.9.

The models can be additionally compared by looking at the generated cloud masks at different cloud detection thresholds. Figures 5.9 and 5.10 show the cloud masks generated by the models TTQ + FP + STE, symmetric TTQ and TTQ + warmup, which become more precise when increasing the threshold from 0.5 (first figure) to 0.9 (second figure). The Symmetric TTQ and TTQ + Warmup models, while effective, tend to generate slightly softer or more fragmented boundaries, aligning with their slightly lower quantitative scores.

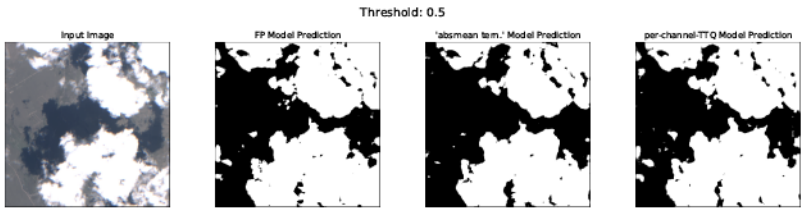


Figure 5.11: From left to right: input RGB composite, Full Precision cloud mask, absmean ternarization cloud mask, per-channel TTQ cloud mask with threshold=0.5.

Figures 5.11 and 5.12 show the top-performing models against the Full-Precision baseline at different CPT: from 0.5 (first figure) to 0.9 (second figure). The per-channel TTQ model produces an output nearly identical to that of the unquantized model, accurately capturing fine details like wispy cloud edges and

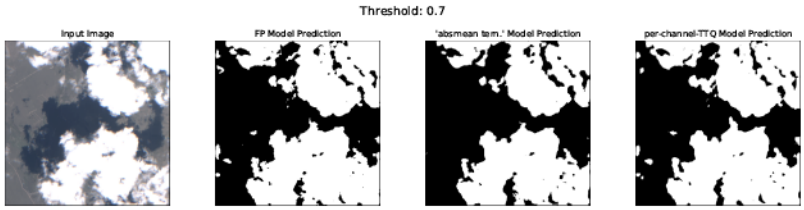


Figure 5.12: From left to right: input RGB composite, Full Precision cloud mask, absmean ternarization cloud mask, per-channel TTQ cloud mask with threshold=0.7.

small gaps between cloud formations. This visual evidence confirms that the channel-wise flexibility of this method allows it to preserve a very high degree of segmentation accuracy.

The same behavior is shown by Figures 5.11 and 5.12, which display the cloud masks generated by the models Full Precision, absmean ternarization and per-channel TTQ. In this case the threshold is increased from 0.5 (first figure) to 0.9 (second figure)

Analysis of Error Types

To understand the nature of the errors, it is useful to visualize the spatial distribution of True Positives (TP), False Positives (FP), and False Negatives (FN). For the onboard compression use-case, minimizing False Positives is fundamental, as each FP represents a clear-sky pixel that would be erroneously subjected to aggressive, lossy compression, leading to a loss of valuable scientific data. False Negatives, while not ideal, are less critical as they result in cloudy pixels being compressed losslessly, which only reduces the overall compression ratio without degrading the quality of clear-sky areas.

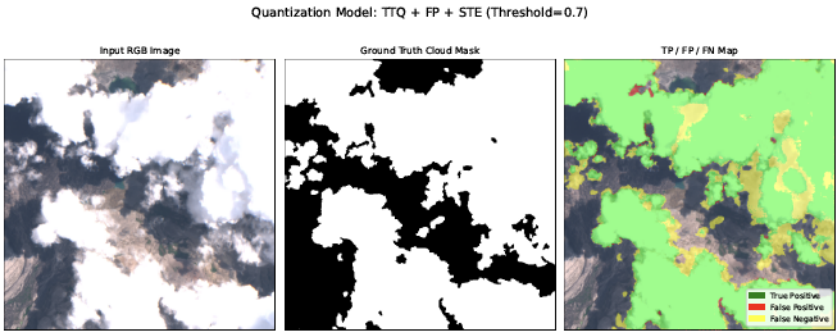


Figure 5.13: Error map for the TTQ + FP + STE model at CPT=0.7. Green indicates correctly identified clouds (TP), red highlights erroneously flagged clear pixels (FP), and yellow shows missed clouds (FN). The low number of red pixels indicates a low rate of critical errors.

Figure 5.13 shows a representative error map. The model exhibits a low number of False Positives (colored in red), which are mostly confined to the

transition zones around cloud edges. In contrast, False Negatives (colored in yellow) are more prevalent, typically occurring in the thinner, semi-transparent parts of clouds where the confidence score is lower.

Figures 5.14 and 5.15 show the effects of changing the Cloud Probability Threshold for detecting the clouds through maps of True Positives, False Positives and False Negatives. The image used for testing is the same, as the model used to create the Cloud Mask: TTQ + FP + STE. By increasing the threshold from 0.5 in the first figure, to 0.9 in the second, the amount of True Positives and False Positives decreases, while the amount of False Negatives increases.

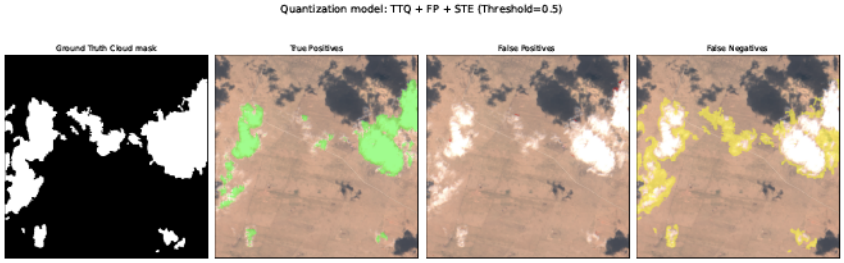


Figure 5.14: From left to right: Ground truth cloud mask, True Positives, False Positives and True Negatives map of the clouds created with the model TTQ + FP + STE and threshold=0.5.

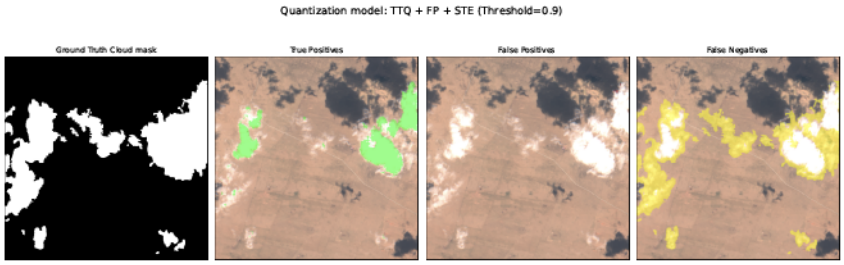


Figure 5.15: From left to right: Ground truth cloud mask, True Positives, False Positives and True Negatives map of the clouds created with the model TTQ + FP + STE and threshold=0.9.

Performance Trade-offs vs. Cloud Probability Threshold

Figure 5.16 illustrates how the FPR evolves as the CPT is varied from 0.1 to 0.9 for the advanced QAT models. The FPR represents the fraction of clear-sky pixels that are incorrectly classified as clouds. A lower FPR is better, as it means fewer false alarms. As expected, all models exhibit a decreasing FPR as the Cloud Probability Threshold increases. This indicates that being more stringent with the classification criteria, requiring a higher confidence score to

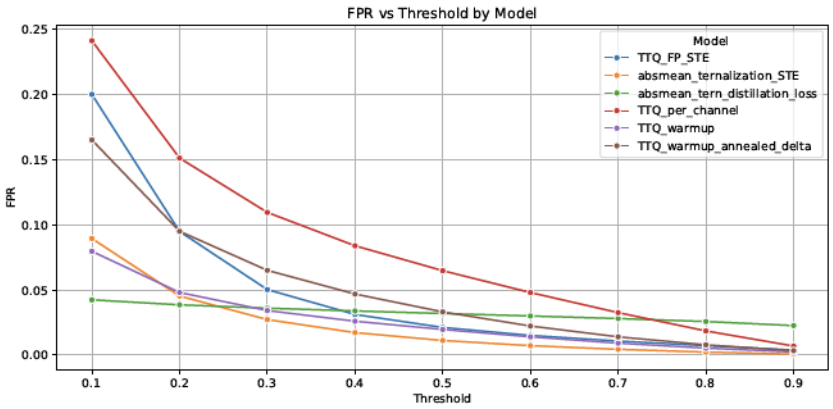


Figure 5.16: False Positive Rate (FPR) vs. Threshold for advanced QAT models. All models exhibit a sharp drop in FPR, with absmean + distillation loss achieving the lowest false alarm rate at higher thresholds.

label a pixel as a cloud, effectively reduces the number of false alarms. The *absmean* + distillation loss model (green curve) consistently demonstrates the best performance, maintaining the lowest FPR across all thresholds and the simple *absmean* ternarization model (orange curve) has the second-lowest FPR. In contrast, the per-channel model (red curve) has the highest FPR, making it the most prone to misclassifying clear sky pixels as clouds compared to the others.

Figure 5.17 plots the F1 Score, which represents the harmonic mean of Precision and Recall, for different advanced models at different CPTs (0.1 to 0.9). A higher F1-score is better. The F1-score for most models peaks at an optimal threshold before decreasing. This peak represents the spot where the model achieves the best balance between correctly identifying all clouds (high recall) and not mislabeling clear sky as cloud (high precision).

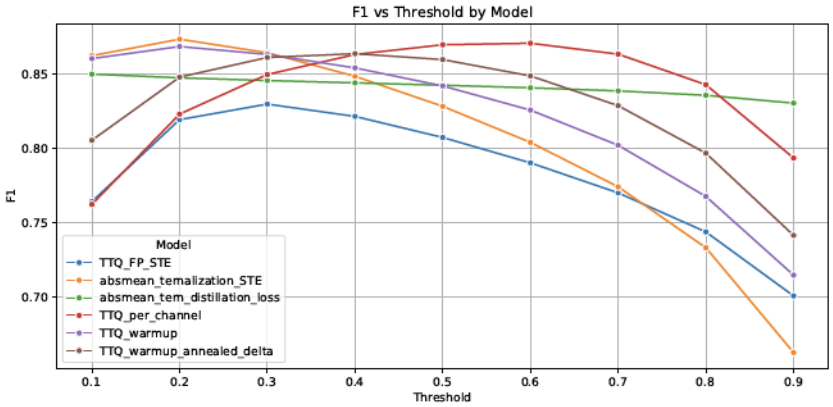


Figure 5.17: F1 Score vs. Threshold for advanced QAT models. Most models achieve their peak F1 Score between a CPT of 0.1 and 0.9.

The per-channel TTQ (red curve) and TTQ + warmup + annealed delta

(brown curve) models achieve the highest peak F1-scores, both reaching approximately 0.87. This indicates they provide the best overall balance between precision and recall, though they achieve this at different optimal thresholds (around 0.5-0.6 for per-channel TTQ and 0.3-0.4 for TTQ + warmup + annealed delta). The simple *absmean* ternarization model (orange curve) also performs well, but it peaks early at a threshold of 0.2. The *absmean* + distillation loss model (green curve) is notable for its stability. While it doesn't reach the highest peak F1-score, its performance is very consistent and degrades much more slowly at higher thresholds compared to others.

The best choice would be the *absmean* + distillation loss model with a high threshold, such as 0.7 since it prioritizes the safety of the scientifically valuable data by ensuring an extremely low False Positive Rate. While its peak F1-score isn't the absolute highest, it remains very high (0.83), indicating that it is still a very effective cloud detector.

5.7 Sliding Window approach on Ternarization models

To emulate more realistic onboard inference conditions, some ternarization models are trained and evaluated using a sliding window approach: *absmean* ternarization model, TTQ + FP + STE and per channel TTQ.

To test the models, a fixed window size of 16 lines is used across all models, while different shift values (4 and 8 lines) are employed during inference.

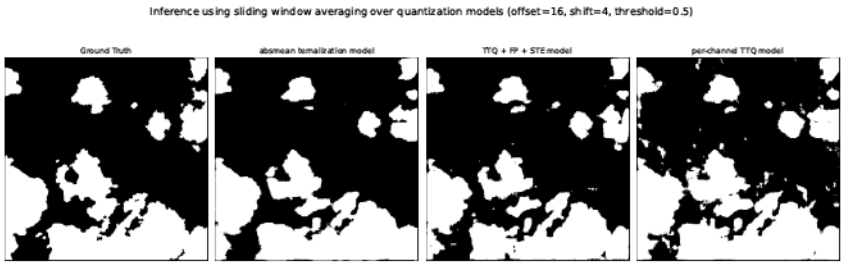


Figure 5.18: Different ternarization models trained and tested though the sliding window approach: window size = 16, shift = 4, CPT = 0.5.

Figures 5.18 and 5.19 display visual comparisons of the predicted cloud masks generated by the three ternarized models evaluated under the same window and shift settings, but changing the cloud detection thresholds from 0.5 to 0.7. The shift of the sliding window is fixed to 4 for the test.

Figures 5.20 and 5.21 display a different image tested under the same conditions except from the shift of the window, which is equal to 8.

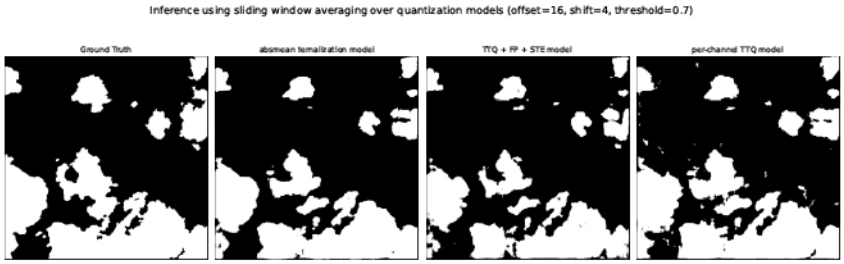


Figure 5.19: Different ternarization models trained and tested though the sliding window approach: window size = 16, shift = 4, CPT = 0.7.

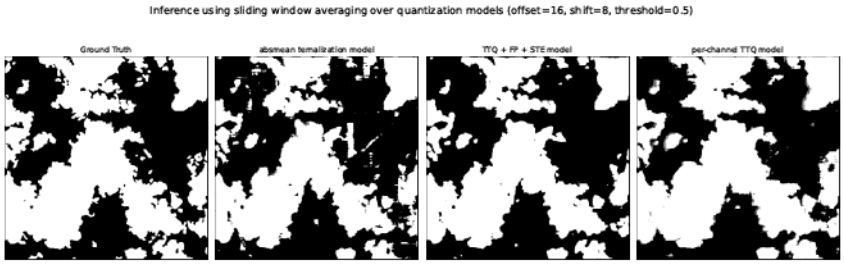


Figure 5.20: Different ternarization models trained and tested though the sliding window approach: window size = 16, shift = 8, CPT = 0.5.

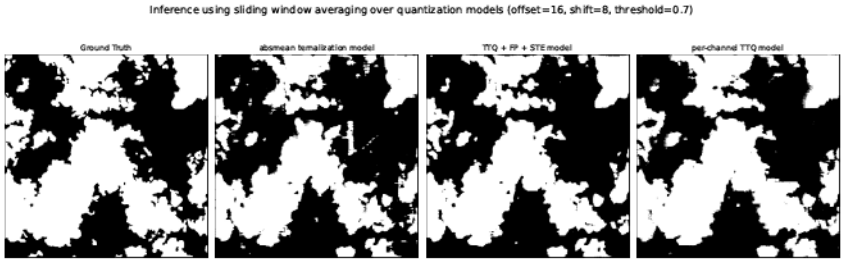


Figure 5.21: Different ternarization models trained and tested though the sliding window approach: window size = 16, shift = 8, CPT = 0.7.

The predictions from the Absmean and TTQ models are smoother and better aligned with ground truth masks at threshold 0.5, while threshold 0.7 sharpens predictions at the cost of increased false negatives.

Table 5.4: Performance of Ternarized Models with a 16-Line Sliding Window.

Model	Shift	CPT	TPR	FPR	BOA	IoU
absmean + STE	4 lines	0.5	0.839	0.045	0.897	0.755
		0.7	0.757	0.022	0.868	0.719
	8 lines	0.5	0.728	0.031	0.849	0.677
		0.7	0.645	0.012	0.816	0.626
TTQ + FP + STE	4 lines	0.5	0.821	0.041	0.890	0.746
		0.7	0.698	0.012	0.843	0.678
	8 lines	0.5	0.760	0.021	0.870	0.723
		0.7	0.668	0.008	0.830	0.654
per-channel TTQ	4 lines	0.5	0.860	0.063	0.899	0.746
		0.7	0.788	0.030	0.879	0.735
	8 lines	0.5	0.876	0.076	0.900	0.738
		0.7	0.847	0.058	0.895	0.742

The results in Table 5.4 confirm the trends observed with the full-precision models: smaller shifts lead to better performance due to increased overlap, at the cost of higher computational load. The per-channel TTQ model again demonstrates its superior robustness, maintaining the highest IoU and BOA across both shift configurations. This indicates that its strong performance holds up under realistic, memory-constrained inference conditions.

By considering per-channel TTQ, with a 16-line window and a 4-line shift at a CPT of 0.7, it is noticeable that this model achieves a BOA of 0.879 and an IoU of 0.735. As shown in Table 5.2, the full-precision model, under the exact same configuration (16-line window, 4-line shift, CPT=0.7), yields a BOA of 0.886 and an IoU of 0.758. This represents a performance gap of 0.007 in BOA and 0.023 in IoU, which is a minimal degradation with respect to the savings in the memory footprint. It shows that the model successfully learns to compensate for the precision reduction during the Quantization-Aware Training process, achieving only a marginal loss in segmentation accuracy.

5.8 Satellite images compression using cloud masks

To reduce data transmission loads and optimize the compression process while preserving essential scientific information, cloud segmentation is used with

different compression strategies for cloudy and non-cloudy regions. In this context, lossy compression is applied to cloud-covered pixels, where radiometric precision is less critical, while non-cloudy areas are preserved using lossless encoding.

This section presents a comparative analysis of various models generating cloud masks and evaluates their impact on compression efficiency, measured in terms of Compression Rate and Mean Squared Error (MSE). The evaluation highlights the trade-offs between segmentation accuracy and compression performance.

The compression rate is evaluated as shown in eq 5.7.

$$\text{Compression Rate (bps)} = \frac{\text{Compressed data size}(bits)}{N_x \times N_y \times N_z} \tag{5.7}$$

where N_x, N_y and N_z are the dimensions of the compressed image.

The Mean Squared Error (MSE) is evaluated as shown in eq 5.8.

$$\text{MSE} = \frac{1}{N_x N_y N_z} \sum_{x,y,z} (s_{xyz} - \hat{s}_{xyz})^2 \tag{5.8}$$

where s_{xyz} is the original sample and $\hat{s}_{xyz}$ is the reconstructed sample.

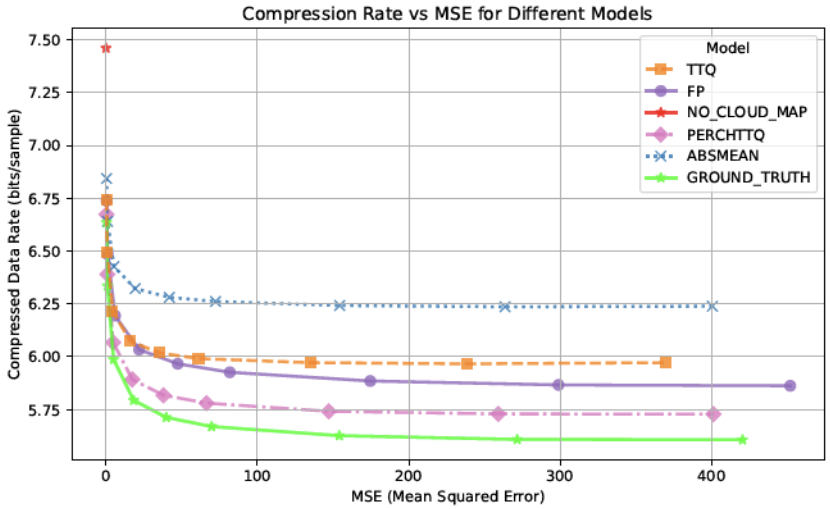


Figure 5.22: Compression Rate vs. Mean Squared Error (MSE) for different cloud masking models at a CPT of 0.7. Lower is better.

Figure 5.22 illustrates the rate-distortion performance of various cloud segmentation models (FP, TTQ, *absmean* and per-channel TTQ). The x-axis, Mean Squared Error (MSE), quantifies the average error or distortion introduced by the compression: a higher MSE value means the reconstructed image is less faithful to the original. The y-axis, Compressed Data Rate, represents the final size of the data in bits per sample: a lower rate means more effective

compression. The precision threshold is set to 0.7 for all the models.

The "Ground Truth" line represents the theoretical lower bound generated using the true label of each pixel, while the "No Cloud Map" star shows the high bit rate required without any adaptive strategy. This is generated without providing any cloud mask and performs the worst in terms of rate (approximately 7.5 bits/sample), showing the importance of cloud segmentation. All tested models generating cloud masks provide a significant reduction in the compression rate with respect to the "No Cloud Map" compressed image.

The plot reveals a crucial trade-off: to achieve a lower data rate (meaning to hardly compress the data), a higher level of distortion (a higher MSE) must be tolerated. To create each curve, the compression algorithm is run multiple times with increasingly stronger quantization settings. Moving from left to right along a curve corresponds to increasing the quantization strength. This discards more data, which naturally increases the reconstruction error (higher MSE), but in return, it allows the data to be stored with fewer bits (lower data rate).

Table 5.5: False Negative Rate (FNR) at CPT=0.7 for Top Performing Models.

Model	**FNR (%) at CPT=0.7**
Full-Precision	0.223
TTQ + FP + STE	0.357
absmean + STE	0.362
per-channel TTQ	0.180

The main performance difference between the models is driven by their False Negatives (cloudy pixels classified as non-cloudy pixels). In the implemented pipeline, any pixel labeled as "non-cloud" is passed to a high-quality, lossless compressor to preserve its scientific value. When a cloudy pixel is mistakenly identified as clear (a False Negative), it is unnecessarily encoded with this inefficient lossless method instead of being aggressively compressed. Consequently, models with higher FNR produce larger final data volumes, as more of the image is forced through the less efficient compression path. While this reduces the overall compression ratio, it is considered a less critical error than the opposite case. A False Positive, where clear-sky data is mislabeled as cloud and aggressively compressed, would result in an irreversible loss of valuable information, which is a much more severe outcome for most mission objectives.

As shown in Table 5.5, the per-channel TTQ model achieves the lowest FNR (0.180) among all tested models. This superior ability to correctly identify clouds translates directly into the best compression performance, as seen in Figure 5.22,

where its curve is closest to the Ground Truth lower bound (outperforming also the Full Precision model). This definitively establishes the per-channel TTQ model as the most effective solution for the end-to-end cloud-aware compression pipeline developed in this thesis.

Chapter 6

Conclusions and Future Work

The exponential growth in Earth Observation (EO) data during the last years has created a significant bottleneck for satellite missions, where onboard storage and downlink bandwidth are severely constrained. This challenge is linked to the presence of clouds in the sky, which obscure large portions of imagery and reduce the scientific value of the collected data. This thesis addressed these issues by developing and evaluating an integrated framework for onboard cloud screening and adaptive compression, designed to enhance data transmission efficiency by prioritizing clear-sky pixels.

The core of this work was a two-stage pipeline that first used a Lightweight U-Net to perform real-time, pixel-level cloud segmentation. The resulting binary masks then guide an adaptive compression scheme compliant with the CCSDS 123.0-B-2 standard, applying aggressive lossy compression to uninformative cloudy regions while preserving the scientific integrity of clear areas through near-lossless encoding. The experimental results confirm that this cloud-aware approach provides a substantial improvement in compression rates compared to non-adaptive methods. A central focus of this research was the optimization of the neural network for resource-constrained hardware through ternary weight quantization. The conducted evaluation included multiple Quantization-Aware Training (QAT) strategies and the findings showed that with the right training methodology, the performance gap between a full-precision model and its highly compressed ternary counterpart can be significantly narrowed. Among the tested techniques, the per-channel Trained Ternary Quantization (TTQ) model emerged as the most robust solution. It achieved an Area Under the Curve (AUC) of 0.97, nearly identical to the full-precision baseline, and delivered the best end-to-end compression performance by maintaining the lowest False Negative Rate. This confirms that learning channel-specific scaling factors is a highly effective strategy for preserving model accuracy even under extreme quantization. The investigation also captured key insights into the practical application of memory-efficient inference strategies. For the non-overlapping

slice-based approach, performance was directly correlated with the amount of spatial context, with larger slices yielding better results. In contrast, the sliding-window method demonstrated a more complex relationship, where a smaller, more focused window sometimes outperformed larger ones. This suggests that for a lightweight architecture, forcing the model to specialize in local feature detection, while relying on the averaging of overlapping predictions to reconstruct global context, can be a highly effective strategy.

While this thesis successfully demonstrates a viable solution for onboard cloud-aware compression, it also opens several avenues for future research and enhancement:

- Full-Model Quantization: This work focused exclusively on weight quantization. The next logical step is to investigate the quantization of network activations. A fully quantized model, where both weights and activations are low-precision, would further reduce computational complexity and power consumption by enabling the use of pure integer arithmetic throughout the network.

- The extension of the model from a binary classifier to a multi-class segmentation network capable of distinguishing between clouds, thin cirrus, haze, smoke, and cloud shadows. This technique might enable even more sophisticated and fine-grained adaptive compression schemes.

In summary, this research establishes a strong foundation for the development of autonomous, efficient data processing systems for next-generation satellite missions. By integrating intelligent sensing directly at the data source, the methodologies explored in this thesis offer a practical pathway to mitigate data bottlenecks and maximize the scientific return of Earth Observation missions.

Bibliography

[1] Miguel Hernández-Cabronero, Aaron B. Kiely, Matthew Klimesh, Ian Blanes, Jonathan Ligo, Enrico Magli, and Joan Serra-Sagristà. "The CCSDS 123.0-B-2 "Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression" Standard: A comprehensive review". In: *IEEE Geoscience and Remote Sensing Magazine* 9.4 (2021), pp. 102–119. DOI: 10.1109/MGRS.2020.3048443 (cit. on pp. 4, 9, 11).

[2] Antonio Plaza et al. "Recent advances in techniques for hyperspectral image processing". In: *Remote Sensing of Environment* 113 (Sept. 2009), S110–S122. ISSN: 0034-4257. DOI: 10.1016/j.rse.2007.07.028. URL: http://dx.doi.org/10.1016/j.rse.2007.07.028 (cit. on p. 4).

[3] José M. Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot. *Hyperspectral Unmixing Overview: Geometrical, Statistical, and Sparse Regression-Based Approaches*. 2012. DOI: 10.48550/ARXIV.1202.6294. URL: https://arxiv.org/abs/1202.6294 (cit. on p. 4).

[4] *Copernicus: Sentinel-2 - The Optical Imaging Mission for Land Services — copernical.com*. https://www.copernical.com/projects-public/item/20279-copernicus-sentinel-2-the-optical-imaging-mission-for-land-services (cit. on pp. 5, 6).

[5] ESA. *Copernicus: Sentinel-2 - eoPortal*. 2025. URL: https://www.eoportal.org/satellite-missions/copernicus-sentinel-2#overview (cit. on pp. 5–7).

[6] Magdalena Main-Knorn, Bringfried Pflug, Jerome Louis, Vincent Debaecker, Uwe Müller-Wilm, and Ferran Gascon. "Sen2Cor for Sentinel-2". In: *Image and Signal Processing for Remote Sensing XXIII*. Ed. by Lorenzo Bruzzone, Francesca Bovolo, and Jon Atli Benediktsson. SPIE, Oct. 2017. DOI: 10.1117/12.2278218. URL: http://dx.doi.org/10.1117/12.2278218 (cit. on p. 6).

[7] Zhe Zhu and Curtis E. Woodcock. "Object-based cloud and cloud shadow detection in Landsat imagery". In: *Remote Sensing of Environment* 118 (Mar. 2012), pp. 83–94. ISSN: 0034-4257. DOI: 10.1016/j.rse.2011.10.028. URL: http://dx.doi.org/10.1016/j.rse.2011.10.028 (cit. on p. 7).

[8] Marharyta Domnich et al. "KappaMask: AI-Based Cloudmask Processor for Sentinel-2". In: *Remote Sensing* 13.20 (Oct. 2021), p. 4100. ISSN: 2072-4292. DOI: 10.3390/rs13204100. URL: http://dx.doi.org/10.3390/rs13204100 (cit. on p. 7).

[9] Amal Altamimi and Belgacem Ben Youssef. "Lossless and Near-Lossless Compression Algorithms for Remotely Sensed Hyperspectral Images". In: *Entropy* 26.4 (Apr. 2024), p. 316. ISSN: 1099-4300. DOI: 10.3390/e26040316. URL: http://dx.doi.org/10.3390/e26040316 (cit. on pp. 7, 8).

[10] Didier Keymeulen et al. "High Performance Space Computing with System-on-Chip Instrument Avionics for Space-based Next Generation Imaging Spectrometers (NGIS)". In: *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2018, pp. 33–36. DOI: 10.1109/AHS.2018.8541473 (cit. on p. 7).

[11] Jiaojiao Li, Jiaji Wu, and Gwanggil Jeon. "GPU Acceleration of Clustered DPCM for Lossless Compression of Hyperspectral Images". In: *IEEE Transactions on Industrial Informatics* 16.5 (2020), pp. 2906–2916. DOI: 10.1109/TII.2019.2893437 (cit. on p. 7).

[12] Xiaolin Wu, Nasir Memon, and K. Sayood. "A Context-based, Adaptive, Lossless/Nearly-Lossless Coding Scheme for Continuous-tone Images". In: (Sept. 1995) (cit. on p. 8).

[13] Ian Blanes, Enrico Magli, and Joan Serra-Sagrista. "A Tutorial on Image Compression for Optical Space Imaging Systems". In: *IEEE Geoscience and Remote Sensing Magazine* 2.3 (2014), pp. 8–26. DOI: 10.1109/MGRS.2014.2352465 (cit. on p. 8).

[14] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), pp. 106–154. DOI: 10.1113/jphysiol.1962.sp006837 (cit. on p. 12).

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: http://dx.doi.org/10.1038/nature14539 (cit. on p. 12).

[16] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116 (cit. on p. 15).

[17] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. "Efficient BackProp". In: (2012), pp. 9–48 (cit. on p. 15).

[18] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted Boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. 2010 (cit. on p. 15).

[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. ICML* 30.1 (2013), p. 3 (cit. on p. 15).

[20] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (ELUs)". In: *arXiv preprint arXiv:1511.07289* (2016) (cit. on p. 16).

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536 (cit. on p. 16).

[22] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827 (cit. on p. 16).

[23] Harsh Srivastava and Kishor Sarawadekar. "A Depthwise Separable Convolution Architecture for CNN Accelerator". In: *2020 IEEE Applied Signal Processing Conference (ASPCON)*. 2020, pp. 1–5. DOI: 10.1109/ASPCON49795.2020.9276672 (cit. on p. 18).

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597 (cit. on p. 19).

[25] *On-board cloud screening algorithms for satellite imaging - Webthesis — webthesis.biblio.polito.it*. http://webthesis.biblio.polito.it/id/eprint/33085 (cit. on pp. 20, 21).

[26] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. *A White Paper on Neural Network Quantization*. 2021. arXiv: 2106.08295 [cs.LG]. URL: https://arxiv.org/abs/2106.08295 (cit. on pp. 22, 24).

[27] Mark Horowitz. "1.1 Computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323 (cit. on p. 23).

[28] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. *Ternary Neural Networks with Fine-Grained Quantization*. 2017. DOI: 10.48550/ARXIV.1705.01462. URL: https://arxiv.org/abs/1705.01462 (cit. on p. 25).

[29] Fengfu Li, Bin Liu, Xiaoxing Wang, Bo Zhang, and Junchi Yan. *Ternary Weight Networks*. 2022. arXiv: 1605.04711 [cs.CV]. URL: https://arxiv.org/abs/1605.04711 (cit. on pp. 25, 26).

[30] Shuming Ma et al. *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*. 2024. DOI: 10.48550/ARXIV.2402.17764. URL: https://arxiv.org/abs/2402.17764 (cit. on pp. 26, 36).

[31] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. *Trained Ternary Quantization*. 2017. arXiv: 1612.01064 [cs.LG]. URL: https://arxiv.org/abs/1612.01064 (cit. on pp. 27, 34).

[32] M. Makitalo and A. Foi. "Optimal Inversion of the Generalized Anscombe Transformation for Poisson-Gaussian Noise". In: *IEEE Transactions on Image Processing* 22.1 (Jan. 2013), pp. 91–103. ISSN: 1941-0042. DOI: 10.1109/tip.2012.2202675. URL: http://dx.doi.org/10.1109/TIP.2012.2202675 (cit. on p. 31).

[33] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: 10.48550/ARXIV.1503.02531. URL: https://arxiv.org/abs/1503.02531 (cit. on p. 37).

[34] Cesar Aybar et al. "CloudSEN12, a global dataset for semantic understanding of cloud and cloud shadow in Sentinel-2". In: *Scientific Data* 9.1 (Dec. 2022). ISSN: 2052-4463. DOI: 10.1038/s41597-022-01878-2. URL: http://dx.doi.org/10.1038/s41597-022-01878-2 (cit. on p. 39).