Attention is Needed to Fight Wildfires from Space!

Instance segmentation · Hyperspectral images · Wildfire detection · Attention · Deep learning

Henrique Duarte Moura
Department of Computer Science,
imec University of Antwerp,
2000 Antwerp, Belgium
henrique.duartemoura@imec.be

Phil Reiter
Department of Computer Science,
imec University of Antwerp,
2000 Antwerp, Belgium
phil.reiter@imec.be

ABSTRACT

Wildfires are destructive natural disasters that severely impact ecosystems, wildlife, and air quality. They emit large quantities of greenhouse gases, thus accelerating climate change. In the U.S., wildfires have become more frequent and severe, costing billions in property damage, infrastructure losses, and firefighting expenses. In 2022, 7.2 million acres caught fire costing around 2.5 billion dollars for Federal Firefighting, *i.e.*, direct firefighting costs. These costs exceeded \$89 billion in lost economic output for the U.S., in 2024 alone, and will cost more than $466,000$ jobs. Europe also faces rising wildfire threats due to hotter, drier summers, with an estimated 30,000 wildfires yearly, in particular in southern regions like Greece, Portugal, Spain, and Italy. Early detection of wildfires can mitigate the environmental, economic, and health impacts.

Satellites play a crucial role in quick detection, providing broader, continuous monitoring compared to traditional methods like ground-based sensors and aerial patrols, which are labor-intensive and limited in range. Satellite systems, including NASA's MODIS and ESA's Sentinel-2, detect fires through visible and infrared light, allowing observation of fire hotspots even through thick smoke. These systems can operate continuously across day and night, and cover vast areas, enabling authorities to manage evacuation and health advisories promptly. Deep learning in computer vision has revolutionized object detection and image segmentation, enabling precise partitioning of images into meaningful regions or objects with superhuman capabilities. These models excel at learning complex patterns and features from images, even in challenging scenarios like occlusions or varying lighting conditions. Their ability to generalize across diverse domains makes them a cornerstone of modern computer vision systems. Thus, we argue that satellite-based deep-learning models can provide an advanced tool for wildfire detection. These models can even run onboard the satellites, identifying thermal anomalies in large-scale images. However, the success of these models is challenged by the imbalanced nature of satellite data - where fire spots cover a small fraction of an image relative to the background - making training a model a hard process. The imbalance creates difficulties in accurately detecting wildfires, with models often misclassifying or missing fire pixels.

To address this, we propose balancing datasets using data-level techniques (*e.g.*, resampling) and algorithm-level techniques (*e.g.*, adjusting the loss function, performing class weight adjustment, or adding attention blocks to the neural network). We experimented with random oversampling methods, *i.e.*, increasing the representation of minority classes by duplicating existing instances, as well as used class weight adjustment to make the model more sensitive to underrepresented classes (fire-affected pixels). In this paper, we show that data augmentation is important for training a deep learning model, and that data imbalance using oversampling also helps obtain better results. We also tested the importance of the frequency bands used to train the model. Our results confirm that IR bands must be considered for fire detection from satellite images since they have greater thermal sensitivity and can propagate through smoke. Loss function optimization is critical for improving model accuracy on imbalanced datasets, allowing control over the trade-offs between false positives and negatives. Also, attention mechanisms can be crucial as they focus on learning the most relevant features, allowing models to prioritize minority class instances without requiring significant data augmentation or resampling. The proposed approach includes a workflow pipeline for wildfire detection that adjusts for dataset imbalances and uses specialized loss functions and attention modules, enhancing detection reliability in satellite imagery. We experimented with different attention blocks and loss functions. Our results show that using an attention block and an appropriate loss function helps improve the model's convergence time and the final performance in terms of F1 score while keeping the precision and recall of the model balanced. Our best model obtains an F1 score of 0.9786 on the test set, while the accuracy reached 0.999999 on a dataset with less than 0.5% positive cases. The combination of techniques used in the paper holds promise for advancing wildfire monitoring, enabling authorities to act swiftly in reducing wildfire impacts and protecting at-risk regions and populations.

1. INTRODUCTION

Wildfires are devastating natural disasters that can obliterate ecosystems, displacing wildlife. They emit vast amounts of greenhouse gases, worsening climate change. They can also lead to soil erosion, degraded water quality, and loss of biodiversity. Rapid detection and monitoring allow for a quicker response, potentially containing fires before they spread too widely, reducing environmental destruction, and helping protect vulnerable species and habitats. Domestic fires cause direct property damage, infrastructure costs, firefighting expenses, and economic losses in affected industries. In the United States, wildfires have become more frequent and intense. The National Interagency Fire Center (NIFC) recorded approximately 50,000 wildfires on average per year from 2014 to 2023, which burned approximately 6.4 million acres per year (NIF, 2024). IMPLAN research (Clouse and West, 2024) estimates that wildfires could cost 466,000 jobs, contribute \$52.2 billion to the gross domestic product and account for \$89.6 billion lost economic output for the US in 2024. In Europe, the number of wildfires is somewhat lower, although they are on the rise due to warmer and drier summers linked to climate change. The European Forest Fire Information System (EFFIS) reported that on average around 30,000 wildfires occur annually on the continent (San-Miguel-Ayanz et al., 2022). These fires can burn hundreds of thousands to millions of acres, especially in Southern Europe (*e.g.*, Greece, Spain, Portugal, and Italy). The 2023 wildfire season was particularly severe, with more than 1 million hectares burned, the highest in recorded European history, and a conservative estimated economic impact of €10 billion. This figure includes damage to infrastructure, emergency response costs, and losses in tourism, agriculture, and forests.

Quick detection minimizes these losses by giving authorities a head start in controlling and extinguishing fires. It also saves costs on emergency response and rehabilitation, which can be enormous after large fires. In addition, public health benefits significantly from faster containment, as wildfires release harmful particles and pollutants, affecting air quality and increasing respiratory illnesses. Compared to traditional methods, such as lookout towers, aerial patrols, and ground-based sensors, satellite imagery has notable advantages. Ground and air patrols are effective for local detection, but are limited in range, require significant human resources, and are less feasible in remote areas or harsh terrains. Although drones offer a more flexible solution than ground-based methods and can cover specific areas in detail, they lack the broad coverage that satellites provide and typically need frequent refueling or battery recharges. Additionally, drones are generally controlled manually or semi-autonomously, limiting their real-time efficacy across large regions.

Satellite-based wildfire detection is crucial for environmental, economic and safety reasons. Satellite systems like the NASA Moderate Resolution Imaging Spectroradiometer (MODIS), Landsat, and Sentinel-2 from the European Space Agency (ESA) capture images in different spectral bands. These satellites monitor both visible and infrared light, allowing for the detection of thermal anomalies, which indicate unusually high temperatures characteristic of fires. Infrared (IR) imaging is especially useful because it can detect heat through smoke, enabling observation of the core of the fire even when smoke plumes obscure it in visible wavelengths. Satellites provide comprehensive coverage over vast and inaccessible regions, allowing for continuous and automated monitoring. Satellite detection is also more efficient in terms of time and cost, especially in areas prone to frequent fires. Satellite data helps authorities warn populations to evacuate and avoid smoke-affected areas, reducing health costs and enhancing public safety. Satellite detection can help reduce these impacts by allowing faster response times, which can prevent fires from spreading out of control. Nonetheless, as climate conditions shift, the risk of fire-related losses continues to grow, emphasizing the importance of robust wildfire management systems.

Recently, deep learning-based satellite image segmentation approaches have gained significant popularity. Several deep Convolutional Neural Network (CNN) models, particularly encoder-decoder-based networks, have been proposed for various domains, including medical, agricultural and satellite imagery. These models, such as 2D U-Net, 2D SegNet, and Mask R-CNN are designed to learn features and perform classification or segmentation in an end-to-end manner.

In this paper, we propose a workflow pipeline that detects fire-affected pixels in satellite images. But our fire detection approach must also deal with imbalanced data issues (Asgari Taghanaki et al., 2021) and dealing with them in deep learning is crucial to improve the performance and robustness of the model. In this work, some methods are applied to address this issue, which can mainly be divided into two: (a) data-level and (b) algorithm-level techniques. The choice of methods depends on the specific problem, the characteristics of the data, and the chosen model. We argue that a combination of these techniques yields the best results in handling imbalanced datasets in deep learning.

Our contributions are as follows:

- **Workflow pipeline fire detector**: We propose a workflow pipeline to detect fire-affected pixels in satellite images. This is a three-stage pipeline. First, it filters out patches that are formed from artifacts in the projected images. Second, the patches filled only with water are also filtered. Finally, the pipeline processes the remaining patches to identify fire-affected pixels. This workflow pipeline detector obtains high F1 score performance (0.9786) and high accuracy (0.999999), while maintaining good balance between precision and recall (0.9763 and 0.9499, respectively).

- **Data-level imbalanced dataset solution**: We evaluate techniques to deal with a highly imbalanced dataset. Only 0.33% of the patches contain positive cases, *i.e.*, at least one pixel represents fire. Even patches that contain fire, less than 10% of the patch area have pixels with fire. We applied random oversampling to increasing the representation of minority classes by duplicating existing instances and then using data augmentation on the duplicated to avoid overfitting. We refrained from synthetically generating new ones due to complexity of the features that characterize fire-affected pixels. However, augmentation techniques such as copy-and-paste (Ghiasi et al., 2021) are left to future work.

- **Algorithm-level imbalanced dataset solution**: We evaluate different approaches: (a) different loss function; (b) attention blocks; (c) simpler versions of some attention blocks to speed up the workflow pipeline's runtime; and (d) class weight adjustment, *i.e.*, by assigning higher weights to minority class samples during model training, we aim to make the model more sensitive to underrepresented classes.

- **Prototype testing**: We deploy our model on an Intel NUC, a Raspberry Pi platform, and a prototype neural network accelerator, developed to be installed on-board satellites by MAGICS. The latter setup, despite being an early-stage prototype, can process all 7,000 64x64 patches in a minute (*i.e.*, the whole Sentinel-2 Level-2A (L2A) image), which is well below the 5-minute requirement for real-time operation for this use-case.

We continue this paper by describing in Section 2 the datasets with the satellite images used by the detection models. Afterward, we enumerate in Section 3 the performance metrics used in this article and several loss functions tested to deal with data imbalance. Section 4 contains a description of the architecture of the proposed workflow pipeline. The experimental results are shown in Section 5. Finally, our conclusions are drawn in Section 6.

2. DATASET

In this paper, we use images processed by Python for RAW Sentinel-2 data (PyRawS), which is an open-source Python package designed for analysis of Sentinel-2 raw data (Meoni et al., 2024), particularly for detecting and classifying warm-temperature hotspots such as wildfires and volcanic eruptions. Sentinel-2 satellites carry the MultiSpectral Instrument (MSI), which captures data across 13 spectral bands, ranging from visible and near-infrared (VNIR) to shortware near-infrared (SWIR). We employed in our experiments L2A products, which are satellite images designed to provide accurate surface reflectance values. The images are corrected for atmospheric disturbances, such as aerosols, water vapor, and ozone, which can distort the satellite's observations. The result is a "bottom-of-atmosphere" reflectance, meaning that users can observe the actual surface characteristics of Earth, as if the atmosphere were clear. This makes the data particularly useful for analyzing land cover, vegetation, and environmental changes. Data are projected on Universal Transverse Mercator system (UTM) with WGS84 datum, ensuring accurate georeferencing and easy integration into Geographic Information Systems (GIS) for mapping and analysis. L2A products are widely available through the Copernicus Open Access Hub.

Sentinel-2 has 3 different image resolutions: 10 m, 20 m and 60 m. In this article, we use 20-m resolution images as the model's input. These L2A images have dimensions: 5,490 x 5,490 pixels. Sentinel-2 provides 6 bands with this resolution. However, there are 4 more bands with 10 m resolution. Using AveragePool with a (2, 2) kernel and a stride equal to 2, we can reduce the resolution of these bands to 20 m and therefore, they can be combined as extra channels in the input images of the detection model. PyRawS dataset contains 20 images from Sentinel-2 covering areas in Europe, South America, Africa, and Oceania that have the ground truth for wildfire (Meoni et al., 2024). PyRawS provides the real L2A and a ground truth (GT) image where each pixel has a binary label that represents whether it is a fire-affected pixel. This pair of images can be used for supervised learning training. However, instead of feeding the model with such huge images, we use smaller non-overlapping squares (or patches) from each satellite image and its GT. Figure 1 shows examples of patches from images using bands B8, B11, and B12 (as RGB channels).

The dataset is highly imbalanced as shown in Figure 2, *i.e.*, less than 0.5% of the patches contain at least one pixel that represents fire. Figure 3 shows that among the image patches containing fire the distribution of affected pixels has a long tail, *i.e.*, most of them are concentrated below the 350-pixel threshold (out of 4096 pixels in the patch). Thus, data-level techniques must be used to address this imbalance. In deep learning those techniques primarily involve resampling and data augmentation. The dataset was divided into two parts: training/validation and test sets. Thus, no image in the test set is used for model training. This unseen data allows us to get an unbiased estimate of the model generalization ability.

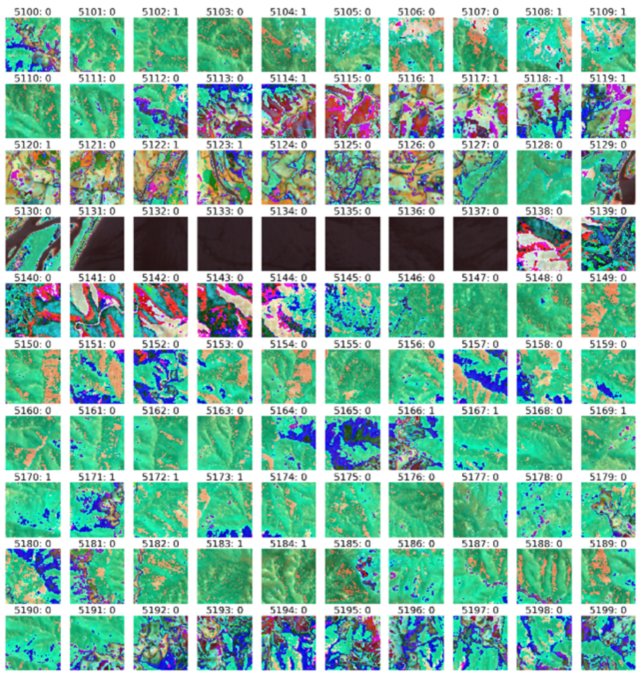

Figure 1: *Example of 64x64 patches from selected images in the training dataset. The colors are visual representations of three infrared bands – B8, B11, and B12.*

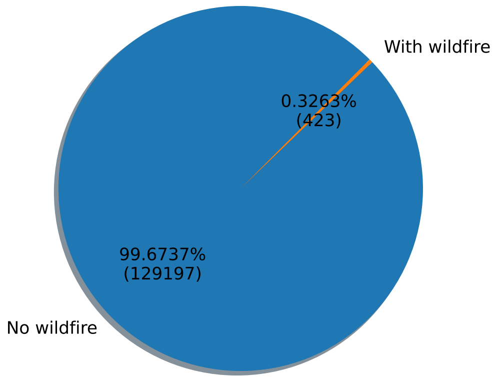

Figure 2: *Proportion of patches with some fire-affected pixel and patches without fire (i.e., zero pixels classified as fire-affected). The graph, above, illustrates the scale of the dataset imbalance. Less than 0.5% of the patches have an active pixel.*

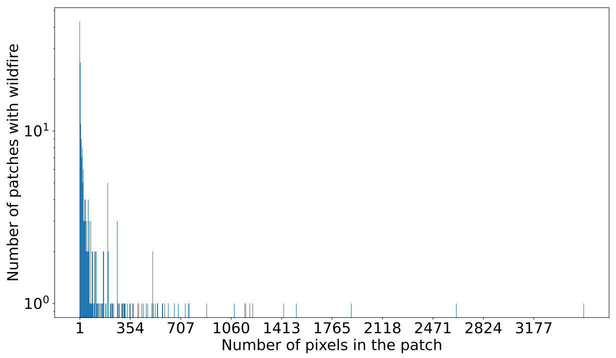

Figure 3: *Plot showing the number of pixels with fire in the patches considering only patches with active pixels. The number of pixels is shown in the Y-axis, which is in logarithmic scale. The number of pixels is shown in the X-axis.*

(i). Resampling

Resampling includes oversampling and undersampling methods. Oversampling increases the number of instances in the minority class using several techniques, while the latter reduces instances of the majority class. We used random oversampling. Existing minority instances are duplicated, which are then augmented. We tried undersampling, however, the initial results were much worse than oversampling, due to loss of information (due to removing instances from the majority class), and overfitting, *i.e.*, the model is not good to identify the minority class in the validation set.

(ii). Data augmentation

The training and validation data were generated by first extracting 64x64 patches from every satellite image. Since the dataset is imbalanced, only patches that contain fire-affected pixels are augmented. Additionally, minority examples are selected randomly and augmented using only geometric transformations. The test set was not augmented. There are several ways to augment image datasets (Xu et al., 2023; Kumar et al., 2024), such as geometric transformations, color adjustments, noise injection, blur techniques, affine and perspective transformations, random erasing and occlusion and other more advanced augmentation techniques (*e.g.*, MixUp, CutMix, Grid Mask, etc.). To avoid loss of information, in this paper we focus only on geometric transformations. More specifically, we used rotation, translation, and flipping. Rotations are performed at any angle. Translations are performed in any direction. No pixels are lost during those transformations, as shown in Figure 4 (D) and (E). During training, there is a probability of 90% that an image is augmented by one of these methods (or a combination of them).

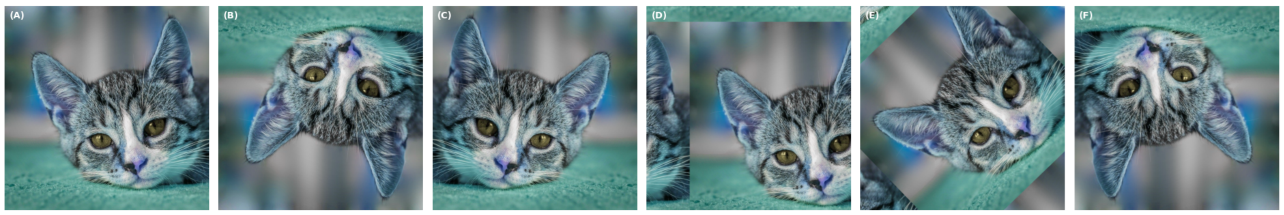

Figure 4: *Example of geometric transformations used in patches. For clarity, we show the transformation on a photo of a cat. (A) Original image, (B) Horizontal flip, (C) Vertical flip, (D) Free translation, (E) Free rotation, (F) Rotation 90°, 180°, or 270°.*

3. THEORETICAL BACKGROUND

In this section we discuss two important topics necessary for model selection: performance metrics and loss functions.

3.1. Performance metrics

Performance metrics are crucial to assess the performance of a model by measuring the difference between predictions and actual results. They help select, optimize, and compare models, ensuring that a model generalizes well with new data. Using appropriate metrics, we can assess model accuracy, efficiency, and reliability, guiding improvements and ensuring that models are fit for their intended purposes. In this section, we present the metrics used in our study.

- **Confusion Matrix** is a table that summarizes the performance of a classification model by displaying the actual and predicted labels together, as shown in Table 1. The table provides a comprehensive way to evaluate the model's accuracy and identify its strengths and weaknesses by pointing out the number of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) when we compare the actual class with the predicted class.

Table 1: Confusion Matrix

Actual / **Predicted**		**Actual class**	
		Positive	Negative
Predicted class	Positive	TP: Correctly predicted positives	FP: Incorrectly predicted (Type I error)
	Negative	FN: Incorrectly predicted (Type II error)	TN: Correctly predicted negatives

Accuracy (Acc) is the ratio of correctly predicted observations to total observations. Notice that accuracy alone can be misleading, especially with imbalanced class distributions. For example, in a binary classification problem with a class distribution of 90:10, predicting all samples as the majority class **A** yields a 90% accuracy, while failing to identify any minority class **B** instances correctly. Thus, accuracy does not provide a complete picture of model performance.

- **Precision** (P) measures the accuracy of positive predictions. It indicates the proportion of TP results in the predicted positives, reflecting the model's ability to correctly identify positive instances.

- **Recall** (R), also known as sensitivity or True Positive Rate (TPR), measures the ability of the model to identify all relevant instances. It indicates the proportion of actual positives that were correctly identified by the model. In imbalanced datasets, high recall for the minority class indicates that the model is good at identifying positive cases, even if it means some false positives.
- **F1 Score** is the harmonic mean of precision and recall, providing a single metric that balances both. It is particularly useful when you need to balance the trade-off between precision and recall, ensuring that neither metric dominates the evaluation. In imbalanced datasets, high values ensure that the model performs well on both precision and recall for the minority class.

Precision, recall, and F1 Score offer a more nuanced evaluation by focusing on the performance of positive predictions and the model's ability to capture all relevant instances. The latter is the main metric used in this paper, but also recall plays an important role in model evaluation.

3.2. Loss functions

A loss function measures the error between a model's predictions and actual values, guiding model improvement during training. By minimizing this error, the model learns to make more accurate predictions over time. There are several classical loss functions in the literature. We checked some, listed below.

Cross entropy (CE) loss, also known as log loss, quantifies the difference between predicted probabilities and actual class labels (Mao et al., 2023). CE produces non-negative values and reaches 0 when the model's predictions perfectly match the actual labels. Lower CE indicates that the predicted probabilities are closer to the actual labels, meaning better model performance. A higher loss suggests poor predictions. In essence, CE loss penalizes incorrect predictions more when the model is confident in its wrong predictions, thus encouraging the model to output probabilities that reflect true likelihoods more accurately. **Weight Cross entropy (WCE)** is a variation of CE that applies different weights to different classes (Aurelio et al., 2019). This is particularly useful in situations such as ours where the class distribution is imbalanced. By applying weights, the loss function can be adjusted to give more importance to the less frequent classes, helping the model learn more effectively. It is calculated as shown in Table 2, where α_i is the i^{th}-class weight. Classes with fewer samples might be assigned higher weights to compensate for their scarcity, ensuring that the model pays more attention to these classes during training. Thus, WCE addresses class imbalance by scaling the loss contribution of each class, thereby improving the model's performance on underrepresented classes.

Focal loss is a loss function designed to address the issue of class imbalance (Ross and Dollár, 2017). It is a variation of CE by adding a modulating factor, which emphasizes hard-to-classify examples and reduces the contribution from easy-to-classify examples. It's formula is shown in Table 2, where p_i is the predicted probability for the target class, α_i is a weighting factor to balance the importance of positive/negative examples, and γ is the focusing parameter that adjusts the rate at which easy examples are down-weighted (typically set to a positive value, such as 2). A higher γ increases the focus on hard examples. The parameter α_i can be used to address i^{th}-class imbalance by assigning different weights to positive and negative examples.

Dice loss, also known as the Sørensen-Dice coefficient loss (Dice, 1945; Sorensen, 1948), measures the overlap between the predicted segmentation and the ground truth (Li et al., 2019), making it highly effective for tasks where the goal is to accurately delineate the boundaries of objects within an image. It is based on the Dice similarity coefficient (DSC), which is a measure of overlap between two sets. It ranges from 0 to 1, where 1 indicates perfect overlap and 0 indicates no overlap. For binary segmentation, $DSC = \frac{2|A \cap B|}{|A|+|B|}$, where $|A \cap B|$ is the number of TP (the intersection between the predicted and ground truth sets), $|A|$ is the number of elements in the predicted set, and $|B|$ is the number of elements in the ground truth set. Dice loss is simply 1 minus the Dice coefficient, but for differentiability and computational purposes, the Dice loss' formula is often written in a form that is suitable for use with continuous predictions (probabilistic outputs) rather than binary labels. Dice loss inherently accounts for class imbalance by focusing on the overlap between predictions and ground truth, making it less sensitive to the dominance of the background class.

Focal Tversky loss is an advanced loss function that combines the concepts of focal loss and the Tversky index to handle class imbalance and improve segmentation quality in imaging tasks (Abraham and Khan, 2019). It emphasizes hard-to-classify examples and accounts for FP and FN differently, making it particularly effective for segmentation problems where precision and recall need to be balanced. The Tversky index, the base of this loss, is a generalization of the Dice coefficient and the Jaccard index, which are commonly used to compare the similarity and diversity of sample sets. The Tversky index introduces two parameters, α and β, to control the penalty for FP and FN, respectively.

Typically $\alpha + \beta = 1$. Focal Tversky loss builds on the Tversky index by adding a focusing parameter γ to emphasize harder-to-classify examples, similar to focal loss, as shown in Table 2.

Combo loss is a hybrid loss function used primarily in image segmentation tasks to leverage the strengths of multiple loss functions. The goal is to improve the performance of segmentation models by combining different loss functions that address various aspects of the task. It combines CE and Dice loss, capitalizing on the benefits of both. Dice loss helps in handling class imbalance and improving boundary precision, while CE ensures robust pixel-wise classification. The formula is shown in Table 2, where α and β are weighting coefficients that balance the contributions of each component. These coefficients are usually set such that $\alpha + \beta = 1$, ensuring a normalized combination.

Table 2: Comparison of loss functions used in this paper

Loss function	**Best use case**	**Imbalance handling**	**Drawback**	**Params**	**Formula**
Weighted BCE loss	Simple imbalanced binary tasks	Moderate	Sensitive to weight choice	α_i	$\mathcal{L}_{WCE} = -\sum_{i=1}^{N} \alpha_i y_i \log p_i$
Focal loss	Imbalanced classification	Strong	Overfitting hard examples	γ, α_i	$\mathcal{L}_{FL} = -\sum_{i=1}^{N} \alpha_i (1-p_i)^\gamma \log p_i$
Tversky loss	Imbalanced segmentation	Strong	Hyperparameter tunning	α, β	$\mathcal{L}_{TV} = 1 - S = 1 - \frac{TP}{TP+\alpha FP+\beta FN}$
Focal Tversky loss	Extreme imbalance, hard examples	Strong	Complex hyperparameter tuning	α, β, γ	$\mathcal{L}_{FTV} = (1-S)^\gamma$
Dice loss	Binary or multi-class segmentation	Moderate	Poor for small regions	None	$\mathcal{L}_{Dice} = 1 - \frac{\sum_{i=1}^{N} p_i \log p_i}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} y_i}$
Combo loss	Imbalanced classification	Moderate	Hyperparameter tunning	α, β	$\mathcal{L}_{COMBO} = \alpha \mathcal{L}_{CE} + \beta \mathcal{L}_{Dice}$

4. PROPOSED SOLUTION

This section introduces the components of our approach to high-resolution hyperspectral image segmentation in remote sensing. We proposed a workflow pipeline for fire detection in satellite images shown in Figure 5. The pipeline performs a binary classification of each pixel in the image, indicating whether the pixel represents fire or not. As a requirement, the model must process the entire image in less than 5 minutes on low-power hardware so that this model can be processed directly on the satellite, *i.e.*, the proposed pipeline should run on embedded hardware without a power-hungry GPU and with a small memory footprint. There are two cases easily identified while processing an image: (a) areas with invalid may appear on each image's edges because of the UTM projection, and (b) some patches are composed only of water (e.g., when extracted from large bodies of water). Thus, the proposed workflow pipeline has two filtering stages before the semantic segmentation model that identifies fire-affected pixels in the patch. These filters perform border and all-water detection and are used to remove patches that have high probability of not containing fire-affected pixels from the fire segmentation model's processing list. We want the first two stages to be small and fast (even at the expense of some accuracy, but keep high recall).

4.1. Border-detection algorithm

This stage identifies the artifacts generated at the edges of the projected L2A image. These artifacts correspond to dark bands that appear because the Sentinel-2 orbit is tilted around the meridians and parallels. The pixels in this artifact have the same value in all bands, which is highly unlikely to occur elsewhere. This is a simple algorithm that considers the sum of all pixels in the patch and compares it with a fixed value, which varies only depending on the selected L2A bands. If the sum is equal to this constant, the patch is considered to be completely in the dark band that does not correspond to the terrain image.

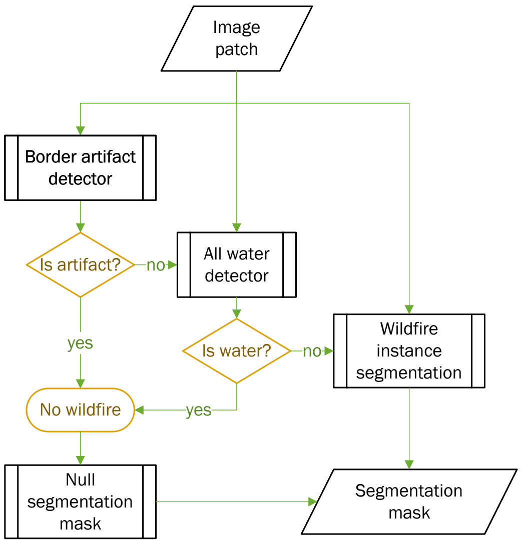

Figure 5: *Wildfire detection workflow pipeline.*

4.2. All-water detector

The second stage of the pipeline identifies areas composed solely of water. Thus, avoiding trying to segment fires in large bodies of water, such as oceans and lakes. The identification threshold is set to a high value ($\delta \gg 0.5$) so only TP is retained (high recall). This stage contains a simple neural network formed by sandwiched convolutional layers, max/average pool layers, and non-linear activation layers (typically ReLU). It performs binary detection, identifying whether the whole input (*i.e.*, patch) is solely water. The proposed architecture is shown in Figure 6. This model has only 641 parameters, an important factor for speed and size.

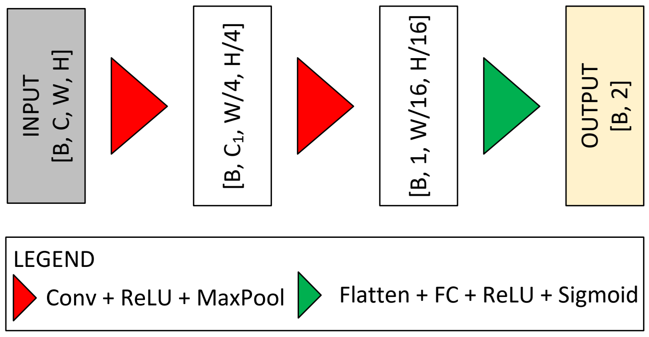

Figure 6: *Proposed architecture for the all-water detector:* ***IsWaterClassifier_512****.*

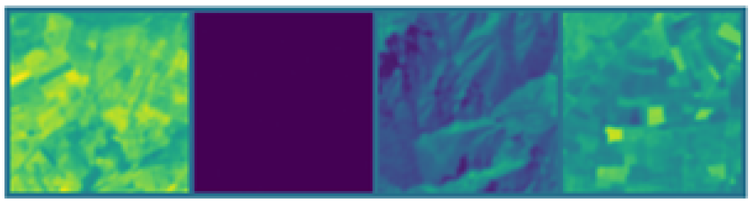

Figure 7: *The figure shows an example of an image using bands 8A, 11 and 12 of the Sentinel-2 L2A image. Only the second image (from left to right) corresponds to a patch composed entirely of water.*

Figure 7 shows four 64x64 patches from the training set, where only the second one (from left to right) is composed solely of water (dark blue patch). These images are formed by infrared bands. The training is supervised, so each patch in the training dataset receives a corresponding label. Table 3 shows the relationship between the number of patches that contain only water and the rest of the patches in the dataset. Only the patches where all pixels correspond to water are considered in the row named "All water". This set was divided into two subsets, for training and validation. We can see in Table 3 that there is a ratio of one patch filled with water for every four patches containing a mixture of land and water. To cope with this imbalance we used a loss function that combines F1 score and recall. *IsWaterClassifier_512* showed an accuracy of 0.9882 and an F1 score of 0.9717. By choosing the loss function above, we could balance precision and recall, which amounted to 0.9470 and 0.9976, respectively.

Table 3: Comparison between patch that contains only only water in PyRawS dataset to the rest.

Type of patch	**Number of patches**	**Proportion**
All water	20,794	20.13%
Land or Land+Water	82,485	79.87%
All patches	103,279	100%

4.3. Wildfire instance segmentation model

Satellite images are typically large and high-resolution, requiring significant computational power and memory for training and inference. The deep learning network must be accurate, small, and fast. Figure 8 shows the final architecture of the fire segmentation model. The input consists of a 64x64 pixel patch with C channels, while the output consists of a 64x64 mask with the probabilities of each pixel being a fire-affected pixel. The architecture was tested with different attention modules (described in the following section), indicated by the circle with the letter A in the middle. Attention mechanisms were introduced in the literature to improve model performance (Hafiz et al., 2021; Hassanin et al., 2024). Its main idea is to focus the model on the most relevant parts of the input sequence in a flexible manner, by a weighted combination of all the encoded input vectors, with the most relevant vectors being attributed the highest weights.

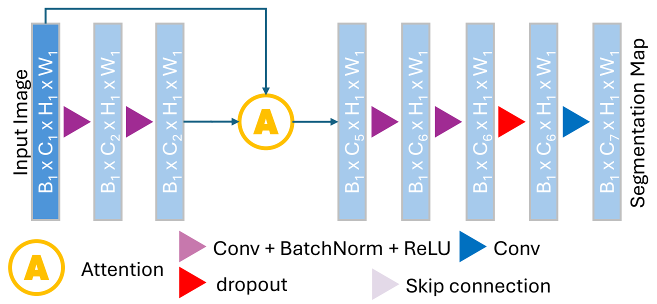
Input Image
$B_1 \times C_1 \times H_1 \times W_1$
$B_1 \times C_2 \times H_1 \times W_1$
$B_1 \times C_2 \times H_1 \times W_1$
A
$B_1 \times C_5 \times H_1 \times W_1$
$B_1 \times C_6 \times H_1 \times W_1$
$B_1 \times C_6 \times H_1 \times W_1$
$B_1 \times C_6 \times H_1 \times W_1$
$B_1 \times C_7 \times H_1 \times W_1$
Segmentation Map
A Attention
Conv + BatchNorm + ReLU
dropout
Conv
Skip connection

Figure 8: *Proposed fire detector.*

4.4. Attention blocks

We tested several attention blocks as described in this section. Depending on the attention module tested, it will use only the feature map generated by the blue block to the left of it in Figure 8 or it may also use the input image of the network.

We tested Squeeze-and-excitation module (eSE) proposed by (Lee and Park, 2020) as an anchor-free instance segmentation, Efficient Channel Attention (ECA) proposed by (Wang et al., 2020) to do channel attention, and a spatial attention modules called Spatial Attention Module (SAM) (see Figure 9) was proposed by Guo et al. (Guo et al., 2021). We modified SAM, proposing a simpler module in Figure 10.

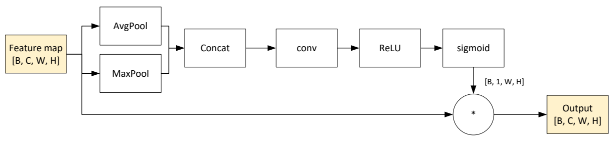

Figure 9: *Spatial Attention Module (SAM).*

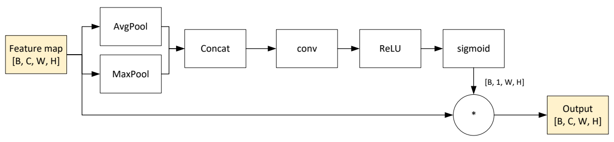

Figure 10: *Spatial Attention Simplified (SAS).*

A more complex structure that sequentially infers attention maps along two separate dimensions (channel and spatial) called Convolutional Block Attention Module (CBAM) (Woo et al., 2018) was also tested. The attention maps are multiplied by the input feature map for adaptive feature refinement. This module is depicted in Figure 11.

Due to the similarity of the problems, *i.e.*, identifying small structures in a large volume of pixels in an image, we tested a simple attention module proposed by (Oktay et al., 2018), called additive Additive Attention Gate (AAG) shown in Figure 12. The spatial regions of the image are selected by the activations and contextual information provided by the gating signal (g), which is collected from a coarser scale.

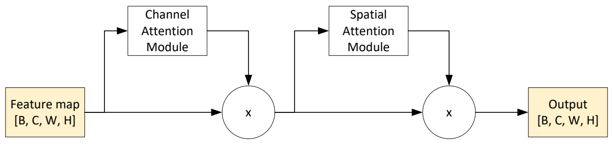

Figure 11: *Convolutional Block Attention Module.*

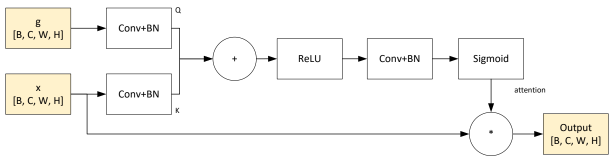

Figure 12: *Additive Attention Gate.*

We checked the Multi-Head Cross Attention (MHCA) module proposed by (Petit et al., 2021). It combines spatial and channel information to turn off unimportant areas from the skip connection features and highlight regions that present a significant interest for the application. Since it is quite complex and large, we proposed a simpler architecture called Simple MHCA (SMHCA) as shown in Figure 13.

Figure 14 shows Basic Linear Attention module (BLA), an attention block that uses Einstein summation to operate on multidimensional linear algebraic matrices combining the features obtained by the neural network with the original image. The result of this operation passes through a nonlinear function and via SoftMax provides the attention mask. This mask is used to highlight the main points of the original image.

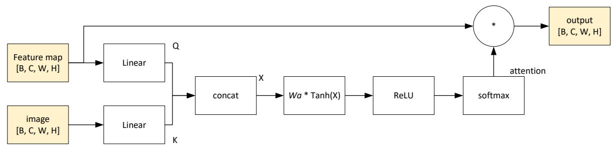

Figure 13: *Simple MHCA.*

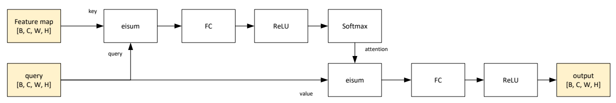

Figure 14: *Basic Linear Attention module.*

Table 4 shows the number of trainable parameters for each attention network tested in this paper. This number gives the reader an idea of their complexity because the number of trainable parameters in a model directly affects its size, speed, and performance. A larger number of parameters increases the model's size, as each parameter requires memory storage, resulting in larger storage requirements for deployment. This also impacts computational speed since training and inference require more operations, leading to increased training time and slower inference, especially on less powerful hardware. Conversely, a larger number of parameters can enhance a model's ability to learn complex patterns, potentially improving performance if sufficient data is available. However, this also raises the risk of overfitting when the model memorizes training data instead of generalizing, particularly when data is limited. Thus, the choice of parameter count must balance accuracy, efficiency, and resource constraints.

Table 4: Number of parameters of the attention block.

Model	**Number of parameters**
CBAM	227,908
BLA	170,369
AAG	159,236
SMHCA	154,000
eSE	153,921
SAS	150,857
SAM	149,860
ECA	149,764

5. EXPERIMENTAL RESULTS

In this section, we describe the equipment used in training and testing the models. We present the results with the fire detection pipeline, where we experimented with several different attention blocks and loss functions.

5.1. Experimental setup

We used several devices to train and test the models used in this paper, as shown below:

- An Intel server with a GPU was used to train the models. It was configured as a Virtual Machine (VM) with 2 Intel Xeon(R) Gold 6240R CPU @ 2.40GHz with 16GB RAM. We call this configuration **CPU**. The VM can also be instantiated with one GPU NVIDIA A100 80GB PCIe with 80 GB RAM. We call this configuration **SRV**;
- A Raspberry Pi (RPi) 4B with a Quad-core Cortex-A72 ARM @ 1.5 GHz, with 4GB RAM. Due to this memory restriction, the default swap file is set to 2GB.
- A RPi 5B with a Quad-core Cortex-A76 ARM @ 2.0 GHz, with 8GB RAM. Both RPi's were configured with a 32GB microSD installed with Raspbian; and
- A **NUC** configuration with one Intel(R) Core(TM) i5-4250U CPU @ 1.30GHz, 8GB RAM, 200GB SSD HD.

We also performed tests with a neural network (NN) accelerator in development by Magics Technologies NV (`https://www.magics.tech/`). This accelerator is still in a very early stage of development. Thus, not all features are available. However, it can give us a glimpse of how an actual hardware will perform. This accelerator was accessed through a proprietary API provided by MAGICS. This API allows the model developed using PyTorch to be quantized, compiled, and submitted to the accelerator. After this step, the image patches can be sent by the local program for processing using another API call. The final prototype will be compatible with SpaceWire Network, which is a high-speed data handling network protocol standard, developed by ESA, that connects various onboard subsystems on spacecrafts.

5.2. Wildfire instance segmentation

Table 5 shows the hyperparameters used to train the fire detector model. By default, the loss function is Binary Cross entropy (BCE), but we also evaluate other loss functions in a later section of this paper. Since our dataset is very imbalanced, we applied a technique called Random Oversampling (Fernández et al., 2018), which randomly duplicates samples from the minority class and to increase the quality of those instances, these samples are augmented. Using **SRV** with the GPU described in "Experimental setup", it takes around 1.8 ms to train each patch. Thus, the whole training takes around 5.5 hours. One whole image (with the border and all-water detector deactivated) takes less than 15 seconds. We also tried this model using an experimental NN accelerator. In this stage of development, it provides an output of 7000 patches per minute, *i.e.*, we can process the whole image in less than a minute with the whole pipeline active.

Table 5: Fire detector hyperparameters.

Hyperparameter	Value
Loss function	default is BCE
Optimizer	Adam (with weight decay = 0.00001)
Adam's beta coefficients	$\beta_1 = 0.9$, $\beta_2 = 0.999$
Learning rate (LR)	0.0001
Number of epochs	100
Batch size	64
Data sampler	Random over sampler
Augmentation	*see text*

Our best model obtains an F1 score equal to 0.9786 as shown in Table 6 on **CPU** using the FP16 data format. This model uses the CBAM block and is trained using BCE loss function (no weights configured). The table also shows the second-best model, which achieves a similar F1 score in a much earlier stage (epoch 13), but does not improve better than the first model, even running for 200 epochs. Notice that BCE is one of the components of the combo loss. Thus, this results may reflect a poor choice in the parameters related to the DICE loss (second component). In summary, the first-ranked model obtain an F1 score that is less than 1% higher than the second, but the second converges faster.

Table 6: Performance of the top 2 ranked models.

Rank	Epoch	Accuracy	Precision	Recall	F1 score	Improv. F1 (%)	Model
1	142	0.9999	0.9763	0.9499	0.9786	0.57%	CBAM + BCE Loss
2	13	0.9999	0.9788	0.9487	0.9730	-	CBAM + Combo Loss

5.3. Experimenting with loss functions

Table 7 has four columns. The first identifies the loss functions (described in Section 3.2) used during training. The second is the epoch the best value of the metric indicated in the third column (F1 score) is obtained. The fourth column shows the percentage improvement of the performance metric in relation to the reference model, *i.e.*, the model trained with Focal loss (the worst model). All models were trained with an attention block. It's possible to trade off recall and precision by adding weights to positive examples on the BCE loss function. Since all results shown are trained using oversampling, this can explain why using weights (as shown by setting pos_weight equal to 10 or 100 at the bottom of the table) produces worse results than the balanced option (pos_weight equal to 1). Combo is a combination of BCE and Dice. We see that by giving more importance to Dice ($\alpha = 0.4$ and $\beta = 0.6$) the result worsens compared to a more balanced setup ($\alpha = \beta = 0.5$), confirming that BCE is more effective in obtaining better results. We also observe that, in general, a combined version of a loss function outperforms the single one, *i.e.*, Combo (*i.e.*, Dice + BCE) over Dice, Focal Tversky over Tversky or Focal.

Table 7: Effect of the loss function in the model performance.

Loss function	Epoch	F_1 score	F_1 Improv. (%)	Accuracy	Precision	Recall
BCE (pos weight = 1)	77	0.9597	25.9%	0.999998	0.9627	0.9268
CDB-CE	83	0.9595	25.9%	0.999998	0.9727	0.9153
Focal Tversky	48	0.9581	25.7%	0.999998	0.9559	0.8909
Combo ($\alpha = \beta = 0.5$)	92	0.9510	24.8%	0.999998	0.9508	0.9435
Combo ($\alpha = 0.4, \beta = 0.6$)	73	0.9508	24.7%	0.999998	0.9696	0.8999
Dice	74	0.9503	24.7%	0.999998	0.9553	0.9320
Tversky	32	0.9474	24.3%	0.999998	0.9086	0.9320
BCE (pos.weight = 10)	95	0.9459	24.1%	0.999998	0.8977	0.9910
BCE (pos.weights = 100)	97	0.8630	13.2%	0.999995	0.7549	0.9961
Focal	93	0.7622	0.0%	0.999988	0.5501	1.0000

Tversky loss tries to balance FP and FN but it implies adjusting the hyperparameters — α and β. This adjustment extends the training time of the model. Our results are in line with the literature, which indicates that simpler loss functions like Dice might perform similarly with fewer tuning requirements. On the other hand, Focal Tversky loss shows better results than Tversky, despite having an additional parameter (γ). However, as we see in the table, the results align with the theory, since it is more effective than Tversky for very imbalanced datasets by emphasizing hard-to-classify samples. Focal loss is a general loss function, and it might not capture spatial relationships, as we can see in the table, since it obtained the worst result among the loss functions tested. The recall with Focal loss indicates that this function overly focuses on hard examples, ignoring easy ones, potentially leading to overfitting.

5.4. Effect of attention modules on performance

Table 8 shows the best results based on F1 score among models trained with the attention modules described in Section 4.4. These models were trained using the BCE loss function. Different attention blocks provide almost negligible improvements over the final F1 score obtained by a model without attention, considering training for 100 epochs. However, as shown in Table 8, most of these modules improve the convergence speed. In our experiments, we observe that adding a loss function is responsible for the improving the performance of the model, while adding the attention module speeds up the process of finding better weights. Adding an additional layer (attention) to the fire detection model adds additional image processing time. This is an important point, as there is a maximum execution window for the pipeline. Table 9 shows the average inference times per patch with the respective confidence interval (95%) using the same models listed in Table 8. The times shown correspond to the inference on the test dataset patches executed on the **NUC**. The right column shows the deterioration of the execution time compared to the model without the attention block. The model with the best performance, considering the F1 score, is 1.7 times slower than the model without attention.

Table 8: Effect of attention modules on model's performance.

Attention	F1 score	Improv. (%)	Epoch
CBAM	0.9532	0.78%	92
BLA	0.9510	0.54%	95
SMHCA	0.9496	0.39%	98
SAM	0.9482	0.25%	53
ESE	0.9480	0.22%	73
AAG	0.9473	0.15%	78
SAS	0.9470	0.12%	72
No Attention	0.9459	-	95
ECA	0.9427	-0.37%	65

Table 9: Effect of attention modules on model inference runtime.

Attention	Runtime (ms)	Deterioration
No Attention	10.8 ± 0.1	-
SAS	11.8 ± 0.1	1.1x
SAM	12.9 ± 0.1	1.2x
ESE	13.4 ± 2.5	1.2x
ECA	13.4 ± 2.5	1.2x
CBAM	18.1 ± 1.4	1.7x
BLA	35.4 ± 0.2	3.3x
SMHCA	45.3 ± 0.2	4.2x
AAG	57.1 ± 0.3	5.3x

5.5. Effect of quantization

Table 10 shows a comparison between the maximum runtime of various hardware setups – **CPU**, **NUC**, 4B, and 5B – for dynamic processing (100 samples) and training (50 epochs) of quantized models. The CPU consistently delivers the fastest performance, with runtimes of 60.2 seconds and 53.0 seconds for dynamic and training quantization scenarios, respectively, followed closely by the **NUC**. In contrast, RPi models exhibit significantly higher runtimes, highlighting their limited processing power. In particular, RPi 5B outperforms RPi 4B, as expected by their specs. RPi 5B demonstrates approximately 50% reduction in runtime for both scenarios against RPi 4B. Across all devices, the runtime differences between dynamic and training tasks are minimal. Overall, Intel-based computers are best suited for time-sensitive or resource-intensive tasks, while RPi devices, particularly the 5B, are better suited for applications where compactness and energy consumption outweigh speed.

Table 10: Effects of hardware on runtime of quantized models.

	Max. runtime (seconds)			
	CPU	NUC	Raspberry Pi	
			4B	5B
Dynamic*	60.2	265.3	1870.1	941.9
Training**	53.0	265.8	1872.5	946.1

* Value with 100 samples

** Training for 50 epochs

Our fire detection model uses several layers of CNN. Depthwise separable convolutions provide a lightweight CNN architecture (Howard et al., 2017) by replacing the sum of multiple convolutional operations by a weighted sum of results obtained from a single convolutional operation. This type of architecture allows one to reduce the size of the model and also the number of computations. We will explore it in future work as well as model compression techniques (Li et al., 2023).

5.6. Power consumed for inference

This section gives an idea of the energy consumption for one inference using the workflow pipeline. We used a DC analyzer (Joulescope JS220) to identify energy consumption due to the intensive use of computational resources related to the operations required by the inference model. It is placed between the RPi power input plug and the power source, thus being able to obtain information about voltage, current and power. The 5V DC power line is sampled at 2KHz. In addition, the equipment allows 2 connections to the RPi GPIO. Then, our code can trigger those GPIO pins within our code to create a binary signal (four different values). This signal is also recorded by the analyzer. As can be seen in Table 11, we identified three cases: (a) the RPi is not executing our code, (b) our code is executing, but the inference model is not executing, and (c) it is performing inference. Inference causes causes a power increase of 36.8% (around 0.8 W). The solar array on the Sentinel-2 satellites (Sentinel-2A and Sentinel-2B) has a power output of approximately 1,700 watts (eoPortal, 2024). Thus, the model consumption is less than 0.2%.

Table 11: Power consumption on RPi 4B.

Activity	**Consumed power (W)**	**Increase**	
		(W)	**(%)**
Pipeline not loaded	2.2577 ± 0.0229	–	–
Python code running (no inference)	2.5944 ± 0.0144	0.3367	14.9%
Python code running (during inference)	3.0893 ± 0.0027	0.8316	36.8%

6. CONCLUSION AND FUTURE WORK

This study demonstrates the high performance of the proposed deep learning model for wildfire detection, achieving an F1 score of 0.9786 on the test set with an accuracy of 0.999999. The best-performing configuration integrated the CBAM loss, striking a balance between accuracy and computational efficiency. It is important to note that it was not necessary to use a very deep neural network to obtain good performance, which is important for embedding this type of specialized model in a NN accelerator on a satellite. Despite working with an extremely imbalanced dataset, where fire-affected pixels make up less than 0.5% of the data, the model successfully identified fire hotspots. Addressing data imbalance was critical to improving the model's performance. Techniques such as random oversampling and data augmentation, particularly geometric transformations, proved effective in enhancing the representation of fire-affected pixels. The study confirmed that infrared spectral bands (B8, B11, and B12) are essential for fire detection, as they provided better thermal sensitivity and allowed fire hotspots to be identified even through thick smoke.

The choice of loss function played a crucial role in optimizing the model's accuracy. BCE loss performed best, especially when used without additional weighting and oversampling. The study also tested a combined loss function (Combo Loss), which integrates BCE and Dice loss, but found that its effectiveness depended on the weight distribution between the two components. Introducing attention mechanisms especially helped speed up the model's convergence, though their impact on F1 score was relatively small. CBAM provided the best trade-off between accuracy and efficiency, whereas more complex attention blocks, such as BLA and SMHCA, significantly slowed down inference. Given the constraints of real-time processing, CBAM was the most suitable choice for deployment.

To ensure the model's feasibility for real-world applications, it was tested on various hardware platforms, including Intel NUC, RPi 4B and 5B, and a prototype NN accelerator. The accelerator demonstrated promising results, processing 7,000 patches per minute, well under the required five-minute window for real-time operation. Additionally, power consumption during inference remained minimal, accounting for less than 0.2% of the Sentinel-2 satellite's power output, making it a viable solution for onboard satellite processing.

Future research will focus on further optimizing the model by exploring lighter architectures, such as depthwise separable convolutions, and applying model compression techniques to reduce computational overhead. Our work also highlights the potential of leveraging advanced augmentation methods like MixUp, CutMix, copy-paste, etc. to improve generalization. Expanding the model's capabilities to work with additional multi- and hyperspectral datasets could enhance its ability to detect wildfires more accurately across different environmental conditions. These advancements will contribute to more effective and timely wildfire monitoring, providing authorities with a powerful tool to mitigate fire-related damage.

ACKNOWLEDGMENTS

This work has been supported by the MOVIQ (Mastering Onboard Vision Intelligence and Quality) project funded by Flanders Innovation & Entrepreneurship (VLAIO) and Flanders Space (VRI) and has received co-funding from the European Union NextGenerationEU.

REFERENCES

Abraham, N. and Khan, N.M. (2019): A Novel Focal Tversky Loss Function with Improved Attention U-Net for Lesion Segmentation, in *2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019)*, pp. 683–687, IEEE.

Asgari Taghanaki, S., Abhishek, K., Cohen, J.P. et al. (2021): Deep Semantic Segmentation of Natural and Medical Images: a Review, *Artificial Intelligence Review*, vol. 54, pp. 137–178.

Aurelio, Y.S., De Almeida, G.M., de Castro, C.L. et al. (2019): Learning From Imbalanced Data Sets with Weighted Cross-Entropy Function, *Neural processing letters*, vol. 50, pp. 1937–1949.

Clouse, C. and West, C. (2024): ”The Economic Ripple Effect of Wildfires: An IMPLAN Analysis”, https://blog.implan.com/wildfires, accessed: 25-11-2024.

Dice, L.R. (1945): Measures of the Amount of Ecologic Association Between Species, *Ecology*, vol. 26(3), pp. 297–302.

eoPortal (2024): Copernicus: Sentinel-2, https://www.eoportal.org/satellite-missions/copernicus-sentinel-2, accessed: 29-10-2024.

Fernández, A., García, S., Galar, M. et al. (2018): Data Level Preprocessing Methods, *Learning from Imbalanced Data Sets*, pp. 79–121.

Ghiasi, G., Cui, Y., Srinivas, A. et al. (2021): Simple Copy-Paste Is a Strong Data Augmentation Method for Instance Segmentation, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2918–2928.

Guo, C., Szemenyei, M., Yi, Y. et al. (2021): Sa-Unet: Spatial Attention U-Net for Retinal Vessel Segmentation, in *2020 25th international conference on pattern recognition (ICPR)*, pp. 1236–1242, IEEE.

Hafiz, A.M., Parah, S.A., and Bhat, R.U.A. (2021): Attention Mechanisms and Deep Learning for Machine Vision: A Survey of the State of the Art, *arXiv preprint arXiv:2106.07550*.

Hassanin, M., Anwar, S., Radwan, I. et al. (2024): Visual Attention Methods in Deep Learning: An In-Depth Survey, *Information Fusion*, vol. 108, p. 102,417.

Howard, A.G., Zhu, M., Chen, B. et al. (2017): MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017), *arXiv preprint arXiv:1704.04861*, vol. 126.

Kumar, T., Brennan, R., Mileo, A. et al. (2024): Image Data Augmentation Approaches: A Comprehensive Survey and Future Directions, *IEEE Access*.

Lee, Y. and Park, J. (2020): Centermask: Real-Time Anchor-Free Instance Segmentation, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13,906–13,915.

Li, X., Sun, X., Meng, Y. et al. (2019): Dice Loss for Data-Imbalanced NLP Tasks, *arXiv preprint arXiv:1911.02855*.

Li, Z., Li, H., and Meng, L. (2023): Model Compression for Deep Neural Networks: A Survey, *Computers*, vol. 12(3), p. 60.

Mao, A., Mohri, M., and Zhong, Y. (2023): Cross-Entropy Loss Functions: Theoretical Analysis and Applications, in *International conference on Machine learning*, pp. 23,803–23,828, PMLR.

Meoni, G., Del Prete, R., Serva, F. et al. (2024): Unlocking the Use of Raw Multispectral Earth Observation Imagery for Onboard Artificial Intelligence, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

NIF (2024): Statistics, https://www.nifc.gov/fire-information/statistics, accessed: 29-10-2024.

Oktay, O., Schlemper, J., Folgoc, L.L. et al. (2018): Attention U-Net: Learning Where to Look for the Pancreas, *arXiv preprint arXiv:1804.03999*.

Petit, O., Thome, N., Rambour, C. et al. (2021): U-Net Transformer: Self and Cross Attention for Medical Image Segmentation, in *Machine Learning in Medical Imaging: 12th International Workshop, MLMI 2021, Held in Conjunction with MICCAI 2021, Strasbourg, France, September 27, 2021, Proceedings 12*, pp. 267–276, Springer.

Ross, T.Y. and Dollár, G. (2017): Focal Loss for Dense Object Detection, in *proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2980–2988.

San-Miguel-Ayanz, J., T. Durrant, R.B., Maianti, P. et al. (2022): ForestFiresinEurope,MiddleEastandNorthAfrica2022, https://effis-gwis-cms.s3.eu-west-1.amazonaws.com/effis/reports-and-publications/annual-fire-reports/Annual_Report_2022_final_240126_print.pdf, accessed: 29-10-2024.

Sorensen, T. (1948): A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons, *Biologiske skrifter*, vol. 5, pp. 1–34.

Wang, Q., Wu, B., Zhu, P. et al. (2020): ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11,534–11,542.

Woo, S., Park, J., Lee, J.Y. et al. (2018): Cbam: Convolutional Block Attention Module, in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19.

Xu, M., Yoon, S., Fuentes, A. et al. (2023): A Comprehensive Survey of Image Augmentation Techniques for Deep Learning, *Pattern Recognition*, vol. 137, p. 109,347.