Deep Neural Network for Wildfire Detection by Autoencoder Pre-training

Elaborated by :

Mariem DOGGAZ

Tijani SIDI ALI

Paulo Roberto DE MOURA JÚNIOR

Gabriele LORENZO

Supervised by : Gianni FRANCHI

Promotion : 2025

Table des matières

1 Introduction

Early detection of wildfires from satellite images is a critical challenge in disaster management and environmental conservation. This project aims to design an efficient detection model under a significant constraint : limited access to labeled training data. To address this challenge, a hybrid approach combining self-supervised and supervised learning was adopted.
The main objective is to extract meaningful representations from the unlabeled training data using an autoencoder neural network, and then leverage these representations to train a classification model capable of distinguishing satellite images containing wildfires from those without wildfires.
The following sections provide a detailed explanation of the methodology, the choices made, and the results achieved.

2 Dataset Description

2.1 Dataset Contents

The dataset used was sourced from Kaggle and consists of 350x350 satellite images containing wildfires and without wildfires, approximately equally divided between these two classes. The whole set is originally divided into three primary subsets :

1. **Training Set :** Contains approximately 10,000 labeled images, but our approach avoids using these labels.
2. **Validation Set :** Comprises about 2,000 labeled images, which will be split between a New Training Set (80%) and a New Validation Set (20%).
3. **Testing Set :** Contains around 3,000 labeled images used solely for final performance evaluation.

2.2 Data Preparation

To ensure the data quality and adhere to project constraints, the following steps were taken :

— Random split of the original Validation Set between a New Training Set (80%) and a New Validation Set (20%).

— Compliance with the constraint of not using labels from the original training set.

3 Methodology

3.1 Data Preprocessing

Before training the models, the raw images underwent the following transformations :

1. **Resizing :** All images were resized to 256x256 pixels, which was found to be a good size to preserve the original image details and reduce the computational costs for training the neural networks.

2. **Data Augmentation :** To increase data diversity and reduce overfitting (not applied during pre-training) :
   — Random rotations in 10 degrees maximum.
   — Random horizontal flips.

3.2 Model Architecture

To address the shortage of labeled data, we initially trained a CNN autoencoder in a self-supervised fashion for image reconstruction. Afterward, we retained only the encoder portion and attached a classification head to its output. We then fine-tuned this classifier using the labeled data from the New Training Set. This approach, known as transfer learning, involves training a model for one task and leveraging its learned weights in another model to perform a different task, thereby reducing the need for expensive pre-training.

3.2.1 Autoencoder

The autoencoder is a self-supervised architecture that encodes input images into compact representations through an encoder and attempts to reconstruct the same images through a decoder. Its role in this project was to extract high-level features from the images using the encoder, projecting them into a lower dimensional latent space, capturing important details while discarding noise or redundant information. By learning how to reconstruct images, the encoder becomes a robust feature extractor which can be further used for several tasks, such as classification in our case.

Structure of the Autoencoder :

1. Encoder : three convolutional layers with increasing filters (64, 128, 256), batch normalization, ReLU activation, and dropout.
2. Decoder : three transposed convolutional layers, reducing filters (256, 128, 64, 3) while applying batch normalization, ReLU, and dropout, with a final sigmoid activation to normalize the output between 0 and 1.

We decided to use 3 convolutional layers in the encoder/decoder in order to maximize performance but avoid increasing the computational cost to train the model, thus reducing the training time, as we have limited access to GPUs. We use batch normalization to speed up training and dropout to help to avoid overfitting. Figure 1 shows the complete autoencoder architecture implemented on PyTorch.

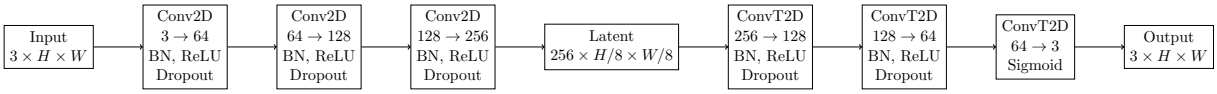

FIGURE 1 – Autoencoder Architecture

3.2.2 Classification head

In order to perform classification, after training the autoencoder to reconstruct images, we save only the encoder weights to use them as initialization for the final classification model, which is composed by the encoder layer (initialized with pre-trained weights), an

adaptive average pooling layer to reduce the number of channels and a fully connected layer with 1 output neuron. Figure 2 shows a representation of the final classifier model.

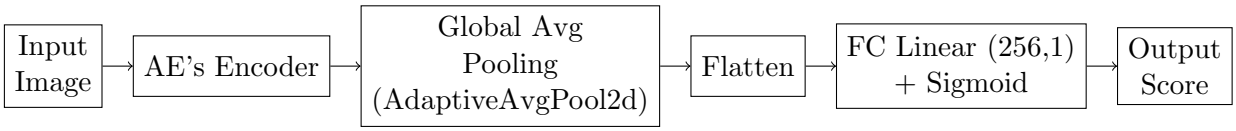

FIGURE 2 – Final classifier architecture

3.3 Training and Optimization

3.3.1 Training the Autoencoder

The autoencoder was pre-trained in a self-supervised manner on the original Training Set by minimizing the Mean Squared Error (MSE) loss between the original and reconstructed images, using Adam optimizer ($lr = 0.01$) and 20 epochs. We perform validation during training and we save the weights of the model with the best (smallest) validation loss to avoid overfitting and ensure we get the best model. In figure 3 we can see that the training is stable as the training loss decreases significantly, and in figure 4 we can see that the final model is reconstructing the images well.

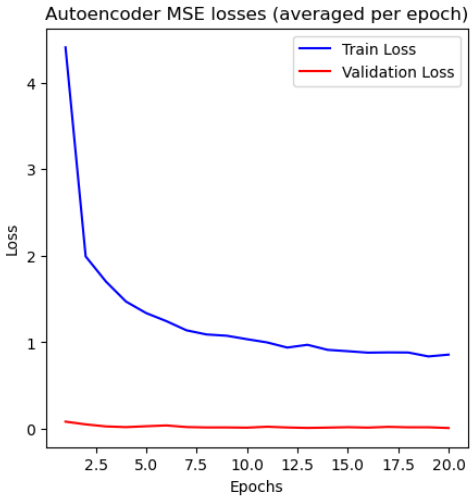

FIGURE 3 – Autoencoder training and validation losses over epochs

FIGURE 4 – Original and reconstructed images by the trained autoencoder

3.4 Training the final classifier

In order to train the complete classifier model, we initialized the encoder with the pre-trained weights trained in the autoencoder for image reconstruction. We then proceeded to train the complete classifier in a supervised manner by minimizing the Binary Cross Entropy (BCE) loss, classifying samples where $output \geq 0.5$ as Wildfire and NoWildfire otherwise. We used Adam optimizer with a smaller learning rate ($lr = 0.001$) and 20 epochs. As before, we perform validation during training, saving the weights of the model with the best (smallest) validation loss to avoid overfitting and ensure we get the best model. In figure 5 we can see that the training is stable as the training loss decreases significantly.

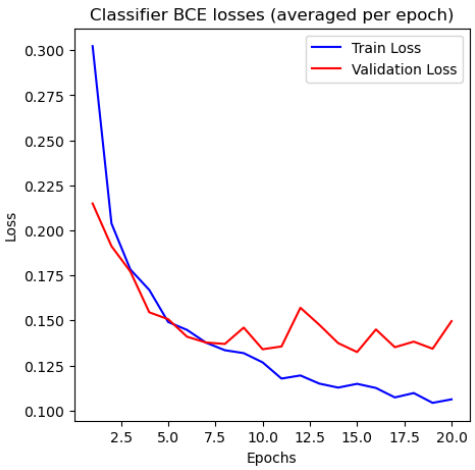

FIGURE 5 – Final classifier training and validation losses over epochs

Finally, we performed hyperparameter fine-tuning for the classification threshold applied to sigmoid output (set to 0.5 during NN training), where we chose the threshold value that maximizes the accuracy on New Validation Set, which was found to be 0.55 in our case.

4 Results

To validate our solution, we performed inference on a Testing Set and computed four classification metrics for this set, which are accuracy, macro precision, macro recall and macro F1. We choose the macro versions of these metrics because the classes aren't perfect balanced, so we decided to give equal importance to them. Figure 6 presents the final results for each metric, showing minimal difference between precision and recall with both metrics having high values, which indicates that the model is well-balanced for classifying both classes, avoiding too many false positives or false negatives.

```
Final Test Accuracy: 96.87%
Macro Precision: 0.9680
Macro Recall: 0.9689
Macro F1 Score: 0.9684
```

FIGURE 6 – Final classification results

5 Conclusion

We successfully demonstrated the effectiveness of combining self-supervised learning with transfer learning to address the challenge of wildfire detection in satellite imagery under limited labeled data constraints. The hybrid approach, utilizing an autoencoder for feature extraction followed by supervised fine-tuning, achieved promising results with macro F1 and accuracy scores of around 97%.

The autoencoder's ability to learn meaningful representations from unlabeled data proved crucial, as evidenced by the quality of image reconstructions and the subsequent classification performance. The balanced precision and recall scores indicate that the model performs consistently across both wildfire and non-wildfire classes, making it suitable for practical applications where false positives and false negatives carry similar costs.

The three-layer convolutional architecture struck an effective balance between model complexity and computational efficiency. In addition, the use of batch normalization and dropout helped prevent overfitting despite the limited labeled dataset. Finally, fine-tuning the classification threshold improved the model's performance beyond the standard 0.5 threshold.

Future work could explore more sophisticated architectures for the encoder or alternative self-supervised learning approaches. Additionally, investigating the model's performance across different geographical regions and seasonal conditions would be valuable for assessing its robustness in real-world deployment scenarios.