ROB313 Deep Learning for Computer Vision Practical Work - Automatic Wildfire Detection

Malécot Jeanne, Moro Maëlys, Amar Noé, El Kalache Georges

Contents

List of Figures

1 Introduction

The aim of this project is to propose a model able to classify satellite images in two categories: the images where there is a wildfire (class 1) and the images where there is no wildfire (class 0). The main challenge in this project is to do image classification with few labeled images (approximately 6300). Training a model from scratch requires a much bigger dataset for interesting performance. In this report we propose 4 different methods in an attempt to handle this problem of lack of training data. The most intuitive approach, when we have data availability problems, is to use foundation models. Conducting fine tuning and transfer learning to adapt the model to our task requires little data and minimal resources. We use multiple models and add pre-processing steps. For example we tested data augmentation methods with different CNN-based foundation models. We finally proposed a - from scratch - approach inspired from vision transformers and based on the use of an encoder trained to reconstruct images.

2 Semi-supervised learning

Pretrained models such as ResNet50 can be adapted for binary classification tasks. However, the small amount of labeled data prevents from reaching a satisfying accuracy. On the other hand, the provided training set contains many images (30,000), but the labels for these images are not available. To utilize the images, we generated labels and then used this pseudo-labeled data to fine-tune the pre-trained model. [1]. The semi-supervised learning is divided in three steps :

- **Step 1 :** Fine-tune a classification layer using ResNet50 and the labeled data. It is the teacher model.
- **Step 2 :** Predict labels for the non labeled data using the fine-tuned model
- **Step 3 :** Re-fine-tune the model using the auto-labeled data. Then, we can use this model, called the student model.

2.1 Fine-tuning a Binary Classification Layer on ResNet50

Using a pretrained CNN such as ResNet50 significantly reduces the necessary resources in terms of data and computation time. Since we do not have access to powerful clusters and have a small dataset, training a CNN from scratch is not feasible. ResNet50 is already pretrained, so we only need to fine-tune a classification layer.

A classification layer is added to the pretrained ResNet50 model:

- A linear layer that transforms the representations created by ResNet50 into a 256-dimensional space to capture complex relations and improve learning capabilities,
- A non linear activation function ReLu,
- A dropout regularization to avoid overfitting,
- A linear layer that outputs a unique value representing the probability of being in a class,
- A sigmoid function to transforms the output into the class number (0 or 1).

The weights of this classification head are trained on the labeled data. The validation set is divided into a new training set and a new validation set, with the repartition 80%-20%. The batch size is set to 32 and the number of epochs to 5. At the end of the training, the accuracy on the new validation set reaches 94.92 % which is a good performance for a first fine tuning.

2.2 Predict labels for non labeled data

In order to use the additional data in the 'train' repository, we thought about predict labels using the fine tuned model (ResNet + fine tuned classification layer). 4000 images are selected randomly and passed through the model. Then, the image's paths and the associated predictions are saved in a csv file.

2.3 Second fine tuning with the autolabeled data

The weights of all the layers are updated during this second fine tuning. The training data is the autolabeled data (csv file) and the validation set used is the same as in section 2.1. The results are shown is the table below (over 3 epochs with a learning rate of 10^{-5}) :

	Train accuracy	**Validation accuracy**
Fine tuning with the labeled data	93.69%	94.92%
Fine tuning with the autolabeled data	99.67%	96.27%

Table 1: Results of the ResNet50 fine tuning

The use of auto-labeled data to fine-tune the model is increasing the validation accuracy by almost 2%. But there is overfitting as the train accuracy is significantly bigger than the validation accuracy. This could be explained by the hyperparameters, but also by the possible inconsistency between the training data and the validation data. Indeed, the labels predicted at step 2 are not always true. In fact, there is around 5% of error (as the accuracy of the first model is close to 95% on the validation set). So, at step 3, the model learns some false data. On the other hand, the validation set contains only true labels. This could explain the difference between the train and validation accuracies. It could have been useful to correct this problem of inaccurate labeled data (as in [2]) but we chose to explore other methods of data augmentation which seemed more accurate.

3 Foundation model with fine tuning and data augmentation.

3.1 Context and Objective

In this part we explored another form of data augmentation combined with a foundation model : ResNet50. The purpose was to see the difference between training with a limited amount of data and a more complete dataset.

3.2 Methodology

3.2.1 Dataset Creation and Data Augmentation

In this part, the dataset was created by combining the original images from the validation dataset with augmented images. The original dataset (organized by classes) was first used to generate additional images through data augmentation techniques. We applied various random transformations (such as rotation, shifts, shearing, zooming, and horizontal flipping) to simulate different viewing angles and conditions. These augmented images were then saved to a separate directory. Both the original and augmented images were combined into a single dataset. Additionally, image statistics (average width and height) were computed to adjust the

target input size dynamically. This comprehensive dataset helps improve model robustness by increasing data variability.

3.2.2 Model Architecture and Training

We employed a ResNet50 model pre-trained on ImageNet. The final fully connected layer was replaced with a Global Average Pooling layer, followed by a Dropout layer (for regularization) and a Dense layer with a Sigmoid activation function to perform binary classification. The training was conducted in two phases:

1. **Phase 1:** The base layers of ResNet50 were frozen, and only the newly added classification layers were trained. This allowed the model to adapt the high-level features learned from ImageNet to the wildfire detection task.

2. **Phase 2:** Fine-tuning was performed by unfreezing the last 10 layers of the base model, allowing further adjustment of the network to better capture domain-specific features.

For training, the images were rescaled and split into 80% for training and 20% for validation. The binary cross-entropy loss was used along with the Adam optimizer. Early stopping and a learning rate scheduler were employed to prevent overfitting and optimize the training process.

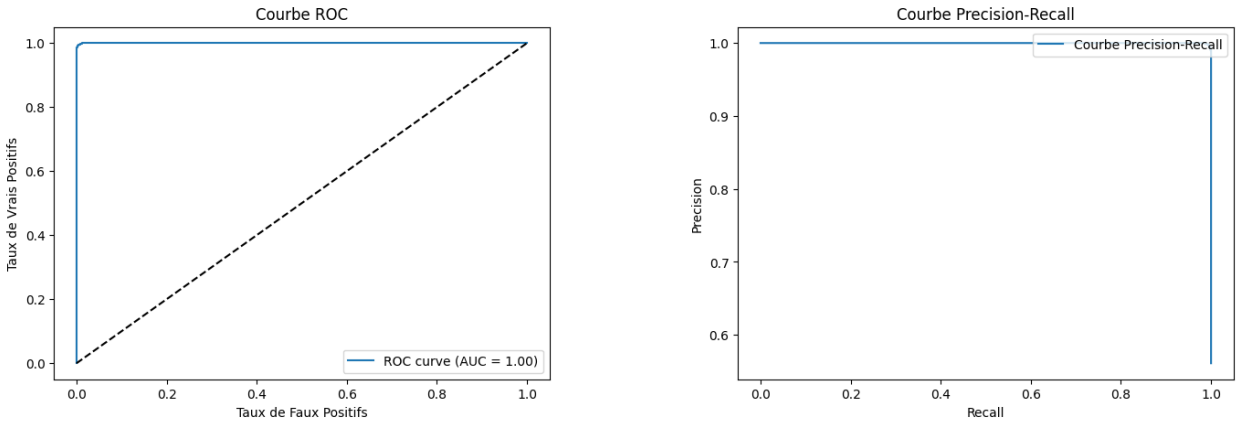
3.3 Results and Metrics Analysis

The model was evaluated on the validation set using several metrics:

3.3.1 ROC and Precision-Recall Curves

- **ROC Curve:** The ROC (Receiver Operating Characteristic) curve plots the true positive rate against the false positive rate at various threshold settings. An AUC (Area Under the Curve) of 1 indicates perfect separation between images with and without wildfires.

- **Precision-Recall Curve:** The Precision-Recall curve highlights the trade-off between precision (the percentage of correctly predicted wildfire images among all images predicted as wildfire) and recall (the ability to detect all wildfire images). An AUC of 1 for this curve confirms that the model achieves ideal detection performance.



(a) ROC Curve

(b) Precision-Recall Curve

Figure 1: ROC and Precision-Recall curves for wildfire detection.

3.3.2 Confusion Matrix

The confusion matrix obtained is:

$$\begin{pmatrix} 551 & 2 \\ 4 & 703 \end{pmatrix}$$

- For the **no wildfire** class (class 0), 551 images were correctly classified, with 2 images mistakenly identified as wildfire.
- For the **wildfire** class (class 1), 703 images were correctly detected, with only 4 misclassifications.

3.3.3 Classification Report

The detailed classification report is as follows:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	553
1	1.00	0.99	1.00	707
accuracy			1.00	1260
macro avg	0.99	1.00	1.00	1260
weighted avg	1.00	1.00	1.00	1260

- **Precision:** For the no wildfire class, 99% of the predictions are correct; for the wildfire class, the precision is 100%.
- **Recall:** The model detects 100% of no wildfire images and 99% of wildfire images, ensuring minimal false negatives.
- **F1-score:** The harmonic mean of precision and recall is 0.99 for class 0 and 1.00 for class 1, indicating excellent overall performance.
- **Overall Accuracy:** With an overall accuracy of 100% on the validation set (1260 images), the model demonstrates nearly perfect classification.

3.4 Discussion

The results—an AUC of 1 for both ROC and Precision-Recall curves, along with outstanding performance in the confusion matrix and classification report—demonstrate that the model reliably detects wildfires. We can try and test it on other datasets to be sure that the model doesn't overfit, but the results are very interesting.
In the following part we will see another model that is supposedly more adapted for this kind of task, EfficientNet.

4 Fine-tuning the pre-trained EfficientNet-B0 model

EfficientNet is a family of convolutional neural networks (CNNs) designed using a compound scaling method that balances depth, width, and resolution for optimal performance. EfficientNet-B0, the smallest variant, was chosen for this project due to its efficiency in terms of computational cost and accuracy trade-offs.

4.1 Architecture of EfficientNet-B0

EfficientNet-B0 is based on the **Mobile Inverted Bottleneck Convolution (MBConv)**, originally introduced in MobileNetV2, and employs **squeeze-and-excitation (SE) blocks** to improve channel-wise feature recalibration. The key architectural features include:

- **Depthwise Separable Convolutions:** Reduce the number of parameters and computations while maintaining spatial feature extraction.
- **MBConv Blocks:** Use inverted residual connections that improve gradient propagation.
- **Squeeze-and-Excitation Mechanism:** Reweights channel importance dynamically to enhance important features and suppress redundant ones.
- **Compound Scaling:** Instead of arbitrarily increasing model size, EfficientNet-B0 applies a principled approach to scale network depth (D), width (W), and resolution (R) efficiently.

Given that fire detection in satellite images requires fine-grained feature extraction while maintaining efficiency, EfficientNet-B0 is well-suited due to:

1. **Lightweight yet Powerful Architecture:** Balances efficiency and accuracy, making it ideal for deployment in real-time or computationally constrained environments.
2. **SE Blocks Enhance Feature Discrimination:** Fire often has unique spectral and texture characteristics; SE blocks help emphasize critical fire-related features.
3. **Efficient Use of Parameters:** With only ~5.3M parameters, EfficientNet-B0 provides high accuracy with fewer computations compared to traditional CNNs like ResNet or VGG.

4.2 Fine-tuning

The initial weights of the pre-trained model on the ImageNet dataset are first loaded. Then the the last fully connected layer is replaced by the classification head with a binary classification head. This layer is then trained on 80% of the validation dataset before test. This allows transfer learning and adaptation to the fire detection task. Finally inference is conducted on the updated model and results are discussed below.

4.3 Results

The results of our image classification model after 15 training epochs demonstrate a strong performance across training, validation, and test datasets. The metrics indicate effective learning, generalization capabilities, and insights into potential overfitting.

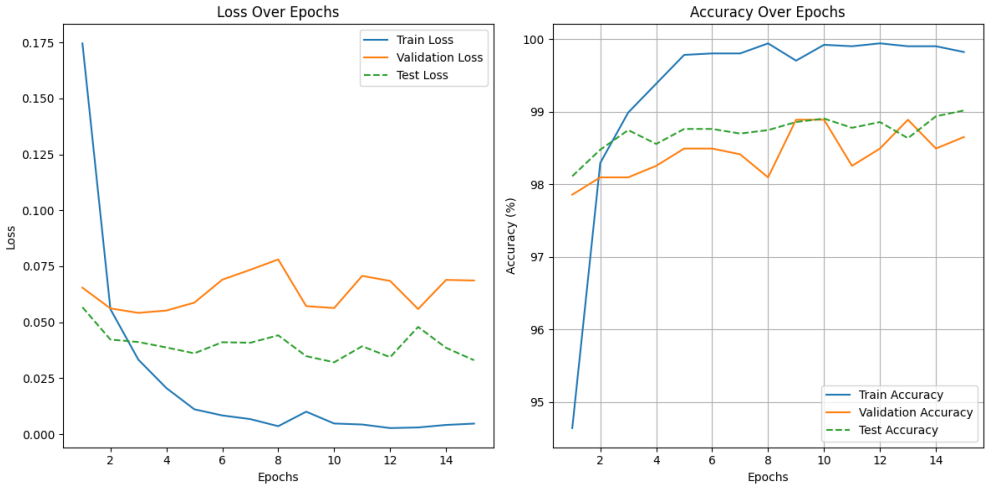


Figure 2: EfficientNet-b0 Training and Validation results

The left subplot illustrates the loss values for training, validation, and testing datasets across the epochs, highlighting the model's performance during training. The right subplot presents the corresponding accuracy metrics, demonstrating the model's classification performance for each dataset throughout the training process. The dashed lines represent test metrics, emphasizing the model's generalization capabilities.

4.3.1 Comparison of Metrics

Table 2 summarizes the training, validation, and test loss and accuracy for the first and last epochs. The training loss decreases significantly from 0.1746 to 0.0047, while the validation and test metrics also reflect high accuracy and low loss values.

Epoch	Train Loss	Train Acc (%)	Valid Loss	Valid Acc (%)	Test Loss	Test Acc (%)
1	0.1746	94.64	0.0654	97.86	0.0567	98.11
15	0.0047	99.82	0.0686	98.65	0.0330	99.02

Table 2: Comparison of training, validation, and test metrics: First vs. Last Epoch.

4.3.2 Overall Performance Assessment

The overall performance of the model is impressive, with both training and testing accuracies surpassed 99%. The low training and test losses indicate effective fitting of the training data while maintaining strong performance on unseen test data.

It is important to note that initially this approach was done for a baseline perspective, given that it introduces minimal fine-tuning, no data augmentation and label generation steps as done previously with ResNet models. This highlights the importance of an adapted initial architecture and choice by task like EfficientNet. So far we've seen attempts of transfer learning using vision foundation models based on CNNs, in the last section we attempt a new architecture involving transformers and sequential processing, introducing encoders and masks.

5 Masked AutoEncoding

We explored another approach of self-supervised learning based on a pretrained encoder coupled with a classification layer. However, we pretrained the encoder ourselves on the training set to ensure the model was trained only on pertinent data for the fire detection task.

The main idea is to train a model to *reconstruct* these images, so it does not require labels. We then use only the encoder part for the classification task, taking advantage of the features learned during the reconstruction phase. Furthermore, to improve the encoder's ability to understand the structure of the images and to learn contexts, we trained the model to reconstruct **partially masked** images.

5.1 Images Patching

We chose to represent the images using **patches**, which are easy to partially mask according to a predefined ratio and are pertinent for use within a transformer architecture.

Figure 3: Patched image

Thus, instead of working with one RGB image, we have to process a set of *n_patches* images.
In this part, all the input data considered will be split into patches as part of the pre-processing.

5.2 Reconstruction Architecture

The architecture described below has been trained solely on the training set. It uses basic transformer encoders and decoders, so we will specifically describe the aspects unique to the masked architecture. [3]

5.2.1 Patch Encoding

After splitting the image into patches, each patch is embedded. A dedicated 'patch encoding' block is added to the global architecture, where all patches are embedded. The positional encoding of the patches is also defined within this block.
If a non-zero *masked_proportion* is specified, the block also performs random sampling of patches, whose embeddings are replaced by the same learnable token (they still retain positional encoding to enable the decoder block to reconstruct a coherent image).

5.2.2 Transformer architecture

The key of this encoder-decoder architecture is that the encoder is trained only on the unmasked patches. The masked patches - tokenized as explained below - are fed directly to the decoder,

which must still reconstruct the entire image.

By doing so, we expect the encoder to generalize better and learn global features, as the masked patches are sampled randomly and are not always the same across different training iterations.

Figure 4: Reconstruction Inference

We observed satisfactory reconstruction results, with a strong ability to reproduce the global structures of the images. The colors and shapes of buildings and roads are clearly identifiable, proving that the encoder has learned meaningful representations.

5.3 Fine tuning for classification

We used the weight of this two-step encoder (patch encoding and transformer encoding) to fine-tune a model for the classification task.
A basic Linear Layer convert the encoded patches into a single float, which is used to predict the class thanks to a sigmoid activation.

For the fine-tuning, we chose to fine tune the classification layer with a learning rate of 10^{-3} and the encoder with a learning rate of 10^{-5}.

5.4 Results

After 20 epochs of fine-tuning, we reached an accuracy of 97.78% on the test set, which was really satisfying since the model was trained from scratch, but we were not able to equalize the performances of the pre-trained foundation models like REsNet and EfficientNet.

The masked encoding was still a great method since it improved a lot the accuracy: we tried to train the classification layer alone, with a simple Linear Layer to embbed the images, and the accuracy was only 89.90%.

6 Conclusion

Our first intuition to use foundation models to deal with the lack of labels proved efficient, since we always reached accuracies greater than 95%. The results of some pre-trained models, even with a very light fine-tuning were absolutely stunning and achieved even an almost perfect accuracy.

Method	Accuracy
Semi-supervised learning	96.27%
Fine-tuning ResNet50 + data augmentation	**99.5**%
Fine-tuning EfficientNet-B0	99.02%
Masked AutoEncoding	97.78%

Table 3: Methods Comparison

The lack of labeled data was a significant challenge in this task, and we noticed a tendency to overfit during the different training and fine-tuning steps. The ResNet model, combined with data augmentation and Dropout layers, achieved the best results among all the methods and demonstrated a great ability to avoid overfitting to the validation data while still learning significant representations.

Our other approach to address the insufficient number of labels was to use the training set data for semi-supervised or self-supervised learning. Despite the high accuracy obtained, the fine-tuned ResNet and EfficientNet models still proved to be the best methods. Again, overfitting is likely the cause: in the semi-supervised learning approach, the model is trained on potentially false labels predicted by the model itself, so it might perform well on the validation data but less so on the unseen data of the testing set.
In the case of self-supervised learning, we trained our model from scratch, so the only data seen by the model came from the Wildfire Prediction Dataset. On one hand, we hoped this would make the model more accurate, as it was specifically trained for this task. On the other hand, we observed that the model overfitted significantly. Perhaps this issue could have been addressed with a deeper architecture, as our model was less complex than the pre-trained ones.

Finally, the most efficient way to achieve excellent accuracy was to use a pre-trained model, not specifically trained on a similar task, so it could generalize well across the limited labeled data. Adding a light fine-tuning step allowed the model to adapt to the specific task of detecting wildfires.

References

[1] Kan Chen Manohar Paluri Dhruv Mahajan Zeki Yalniz, Hervé Jégou. Billion-scale semi-supervised learning for image classification. 2019.

[2] Qizhe Xie Minh-Thang Luong Quoc V.Le Hieu Pham, Zihang Dai. Meta pseudo labels. 2021.

[3] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. 2021.