IA323 - Computer Vision

Wildfire Detection Project

Etudiants :

CESAR Pierre
FIDELIN–DURAND Julian
HAMDAOUI Ismail
LOUVARD Enzo

Encadrant

FRANCHI Gianni

Table des matières

1 Introduction

This project focuses on fire detection using a provided dataset [3]. The dataset Wildfire Prediction is divided into three folders : a training set, a validation set, and a test set. The goal of this project was to be able to build an efficient classifier without using the labels of the training set. To do so, several methods can be used, such as transfer learning, self-supervised learning with different approaches,... This report will explore all our experiences and compare them.

2 Dataset

The dataset consists of satellite images and divided in a total of 6 folders ordered as : (type of use for the data) {"train", "valid", "test"} → (label of the data) {"wildfire", "nowildfire"}. Each image is named after its longitude and latitude coordinates and depending on if it presents a former wildfire, it will be sorted in one or the other folder. Many different areas can be found in the data such as forests, waters, cities, roads, sand...

(a) Image of city and water

(b) Image of forest

FIGURE 1 – Examples of images from the dataset

As shown in table 1, there is a huge unbalance in the data repartition which can lead to poor performances if a model is train only using a fraction of the valid folder.

Folder	Label	number of images
Train	Wildfire	15 750
Train	NoWildfire	14 500
Valid	Wildfire	3 480
Valid	NoWildfire	2 820
Test	Wildfire	3 480
Test	NoWildfire	2 820

Tableau 1 – Number of images in each folder

3 Baseline

We begin to train a CNN baseline only on the new train test to see which performances we could/should reasonably beat with the techniques used here.

Below is the global architecture of this Baseline :

Layer

`Conv2d(3, 32, 3, padding=1)`
`BatchNorm2d(32)`
`ReLU`
`Dropout(0.3)`
`MaxPool2d(2)`

`Conv2d(32, 64, 3, padding=1)`
`BatchNorm2d(64)`
`ReLU`
`Dropout(0.3)`
`MaxPool2d(2)`

`Conv2d(64, 128, 3, padding=1)`
`BatchNorm2d(128)`
`ReLU`
`Dropout(0.3)`
`AdaptiveAvgPool2d(1)`
`Flatten`

`LazyLinear(128)`
`ReLU`
`Linear(128, 2)`

Total

Training only on 80% of the val set, we obtain a final test accuracy of **95.27%** indicating that the challenge will be to gather the missing percentages.

4 Self-Supervised Learning

4.1 Self-training (pseudo-labeling)

Self-training (often called pseudo-labeling) is another technique to tackle the problem of unlabeled data, where the model uses its own predictions on unlabeled data as additional training labels.

The core idea is simple :

— After an initial supervised training phase on a labeled dataset, the model predicts labels for unlabeled examples. When these predictions exceed a confidence threshold (i.e., the maximum predicted probability is above a set value τ), they are treated as "pseudo-labels."

— Self-reinforcement Loop : The model is then retrained using both the original labeled data and the confidently pseudo-labeled examples. This iterative process can progressively refine the decision boundary by incorporating more data without manual annotations.

For our experiment, we started with **baseline** model that scored : 0.94 to predict the pseudo-labels at first. under a given confiedence threshold τ high enough to keep only most sure labels. and we ran through 3 iterations of Pseudo-labeling, here are the following results :

Accuracies	$\tau = 0.95$	$\tau = 0.98$
Iteration 1	93.3%	**93.7%**
Iteration 2	88.8%	91.2 %
Iteration 3	78.1%	–

Tableau 2 – Pseudo-labeling

We observed that the highest accuracy achieved was 93.7%, which is below the baseline. Consequently, we quickly concluded that this approach may not be suitable. One possible reason is that the pseudo-labels are noisy, which could explain the drop in performance after each iteration.

4.2 DINO

Here we used Dino to pre-train a VIT-small encoder on the training dataset (without labels) and then finetune this model on the val dataset. The code to train it was mainly based on the facebook repo [1] adapted to our dataset and on a non-distributed architecture. The main idea behind DINO is to use a teacher-student framework : both the teacher and the student process large (global) crops of the image but the student also receives several smaller (local) crops. The teacher whose weights are updated via an exponential moving average of the student's parameters for stability and to provide a target representation that

is sharpened and centered. The student updated by gradient descent is trained to match the teacher's outputs across these different views using a cross-entropy loss. This encourages students to learn consistent and robust representations without relying on explicit negative pairs.

Here is the configuration of the model used :

TABLEAU 3 – Hyperparameters for ViT-Small Architecture

Hyperparameter	**Value**
Input Image Size	$350 \times 350 \times 3$
Patch Size	16×16
Embedding Dimension	384
Number of Transformer Blocks (Depth)	12
Number of Attention Heads	6
Normalization Layer	LayerNorm with $\epsilon = 1 \times 10^{-6}$
Number of Classes	2

Below are the parameters of the training (see Tab 4), which are mainly the default parameters of the script found on GitHub.

During the dino pretraining, we see the teacher and student rapidly getting better and stabilizing (see Fig 2).

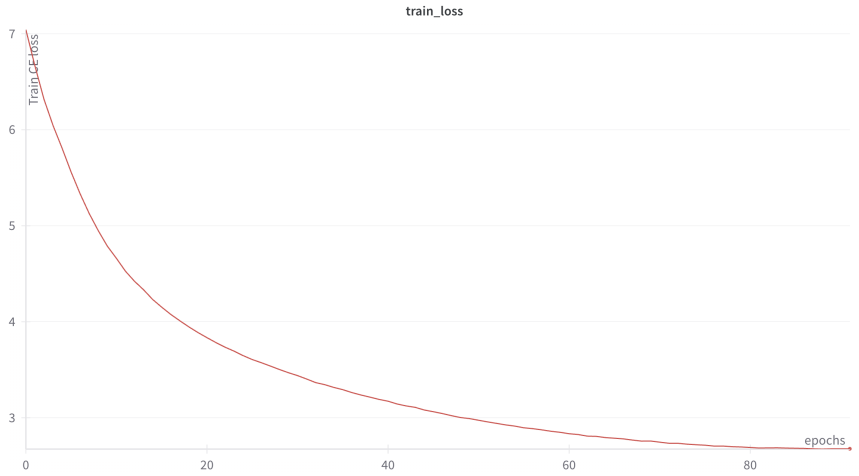

FIGURE 2 – Dino pretraining

Tableau 4 – Hyperparameters for Training ViT-Small

Parameter	Value
Architecture	vit_small
Patch Size	16
Output Dimension	16384
Normalize Last Layer	True
Momentum Teacher	0.996
Use BatchNorm in Head	False
Temperature Parameters	
Warmup Teacher Temp	0.04
Teacher Temp	0.04
Optimization	
Use FP16	True
Weight Decay	0.04
Weight Decay End	0.4
Clip Gradient	3
Batch Size	64
Epochs	100
Learning Rate	0.0005
Warmup Epochs	10
Minimum Learning Rate	1e-6
Optimizer	adamw
Drop Path Rate	0.1
Multi-Crop	
Global Crops Scale	[0.4, 1]
Local Crops Number	8
Local Crops Scale	[0.05, 0.4]

Based on that we change the head for either one or two MLP layers and try to :

— finetune only the last layer and freeze the backbone

— finetune the two last layers and freeze the backbone

— finetune the whole architecture with a big of small head

Below is how it performs in val/test when retraining on the 80% of the val dataset (see Fig 3).

The final results on the test set are available in Tab 5

What we see here is that the ViT model pretrained itself is not expressive enough for a finetuned head to be effective enough (even if it still performs very well but not better than the baseline). Indeed when freezing the backbone, the head finetuning highly depend on the power of the representation learned before and here we see a performance cap. However, when

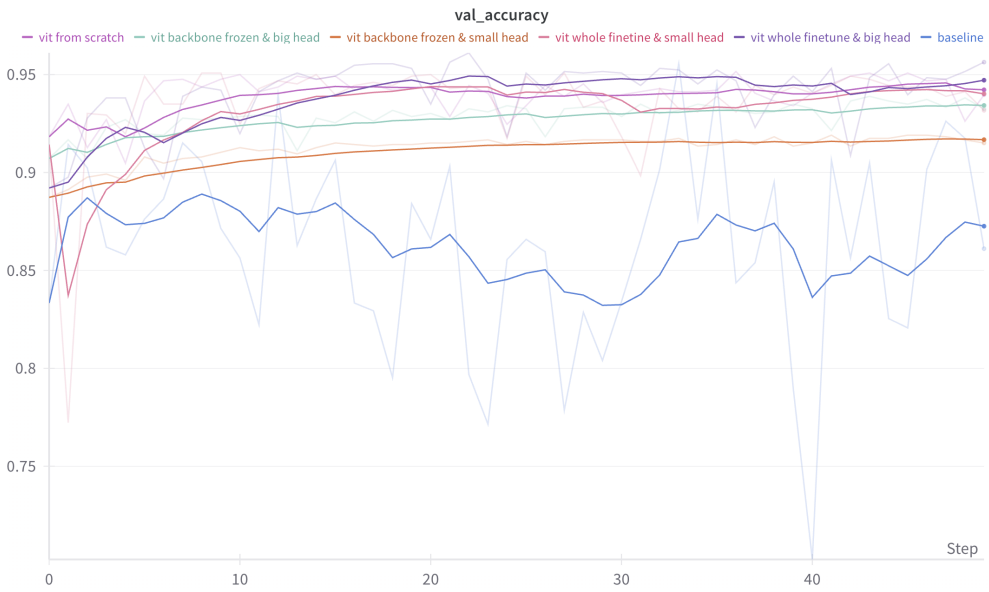

FIGURE 3 – Dino val accuracy curves

Model	**Accuracy**
Baseline	0.9527
ViT Whole Finetune Big Head	0.9519
ViT Whole Finetune Small Head	**0.9589**
ViT Backbone Frozen Big Head	0.9323
ViT Backbone Frozen Small Head	0.9093

TABLEAU 5 – Test Accuracy Results for Different Model Configurations

finetuning the whole model pretrained, we achieved better performances than the baseline very huge.

4.3 Pretraining the model on a close task

One of the approaches we tested is to pretrain the baseline model to do a task rather close to the wildfire detection in order to get it used to handle the images. The one we chose is a coloration detection one.

4.3.1 Coloration detection

The coloration detection task was done on the train dataset, the whole images were used and labeled accordingly to the randomly added color :

— No color added : label 0 → [0, 0, 0]

— Green color added : label 1 → [0, 100, 0]
— Blue color added : label 2 → [0, 0, 100]
— Red color added : label 3 → [100, 0, 0]

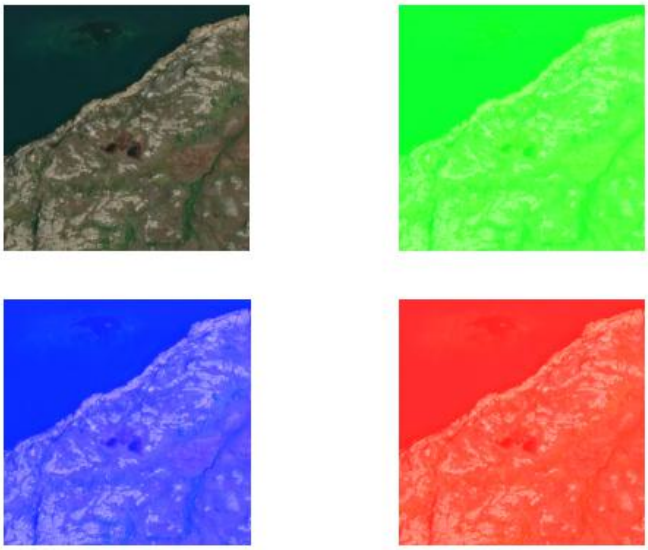

FIGURE 4 – Example of colored images

The initial model had 4 outputs (one for each color) and randomly initialized. Once a training over 10 epochs with a learning rate of 10^{-4} was completed we obtained the following results :

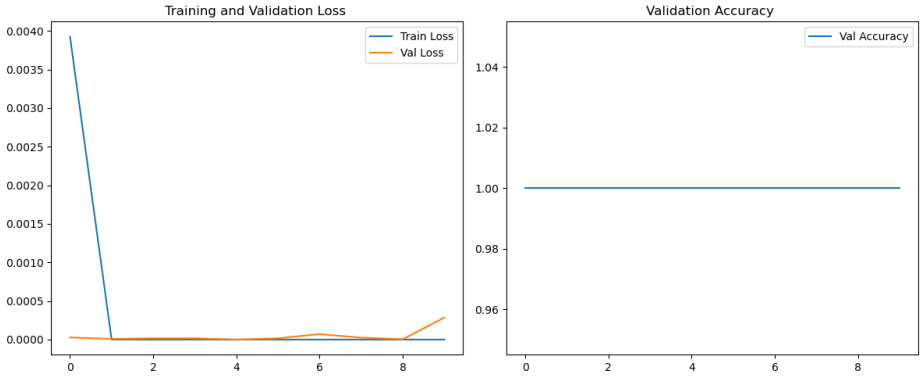

FIGURE 5 – Losses for the coloring training

It appears here that it was an easy task for the baseline model, in future works it would be interesting to modify the number of added colors and their values (maybe it was to obvious for now).

4.3.2 Training for the wildfire classification task

Then the classification head of the model was changed to a 2 outputs one (wildfire and nowildfire) and the other layers were frozen in order to train only the dense one on the "validation" dataset which was splitted in a new train and validation repartition for 7 epochs with a learning rate of 10^{-3}. The goal was to let this layer catch up to the rest of the model.

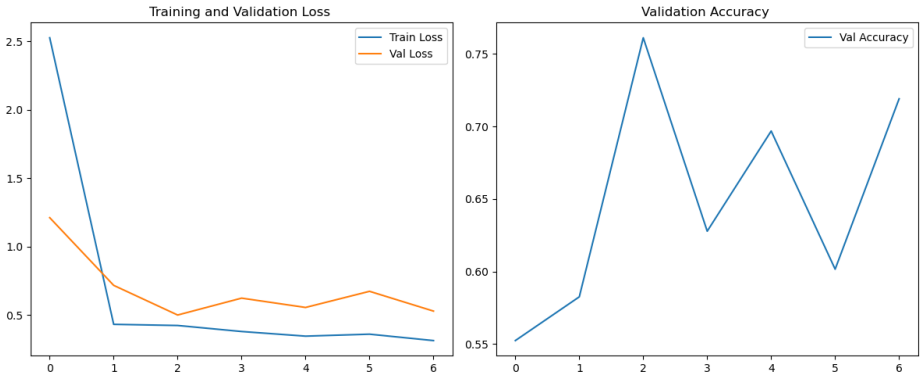

FIGURE 6 – Losses for the partially frozen model

We then trained the whole model for 10 epochs with a learning rate of 10^{-4} to make as robust as possible.

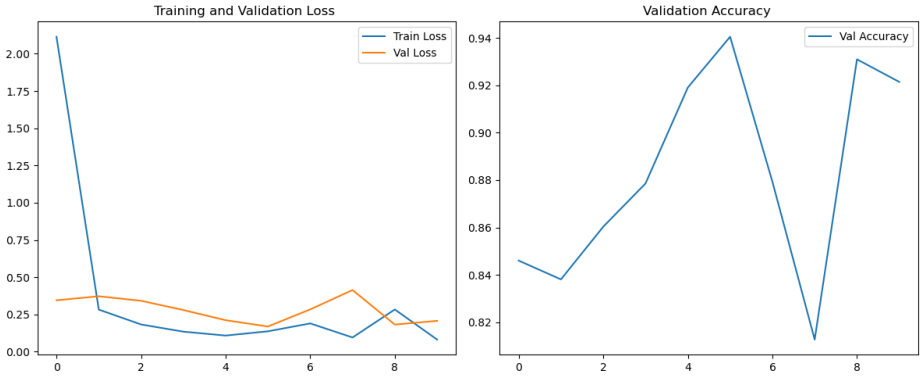

FIGURE 7 – Losses for the completly trained model

4.3.3 Results analysis

The summary of the accuracy at each step is written in Table 6 :

The top accuracy obtained for the final task on the test dataset is 95% which is good but not really better than the inital baseline model. We can see that the pretraining task was not difficult enough to create a model that outperforms the baseline, a pursuit of study would be to determine how many colors are needed and to what extent should they affect

Model	Validation
Coloration	1.0
Partially frozen	0.77
Completly Trained	0.95

TABLEAU 6 – Accuration of the model at each step

the input images. Also, a fine-tuning for the hyper-parameters might enable better results. However, this idea seems a good way to take avantage of unlabeled data.

4.4 Context Encoder : Learning by Impainting

Context Encoders are a self-supervised learning technique, where the pretext task is to learn "what goes in the middle". They are trained to predict and generate missing parts of an image based on the surrounding context. This architecture was exposed in [4]. We use this technique to pretrain an encoder on the training dataset without labels. This encoder will be used later on as a backbone for the classifying task.

4.4.1 Architecture and training

Our implementation is based on the architecture proposed in the original paper :

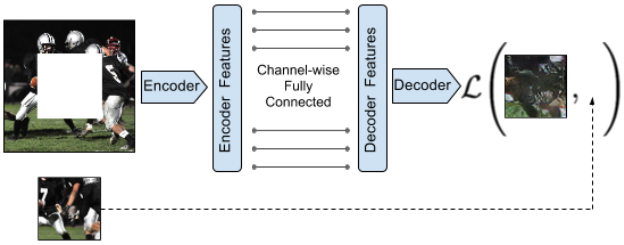

FIGURE 8 – Context Encoder. The context image is passed through the encoder to obtain features that are connected to the decoder using a channel-wise fully connected layer

Table 7 gives a more detailed description of the Encoder and Decoder :

As in the paper, we choose to use GAN for the masked zone reconstruction. To have an easier training process, we also used Wasserstein GAN loss with gradient penalty. For the training we use parameters of Table 8.

The training was done using one L40s, with 40G memory, for 24h. Figs 9, 10. show that the training was complete and that there is no mode collapse on the training of the GAN.

Component	Configuration
Input	$350 \times 350 \times 3$ RGB image
Encoder	4× Conv2d(k=4, s=2, p=1) + BN + LeakyReLU : $3 \to h/8 \to h/4 \to h/2 \to h$ channels Output : $175^2 \to 87^2 \to 43^2 \to 21^2$
Channel-Wise MLP	Per-channel FC : $21^2 \to 2(21^2) \to 21^2$ + ReLU + Dropout Cross-channel : Conv1x1($h \to h$) + BN + ReLU
Decoder	4× ConvTranspose2d(k=4, s=2, p=1) + BN + ReLU : $h \to h/2 \to h/4 \to h/8 \to 3$ channels Output : $43^2 \to 87^2 \to 175^2 \to 350^2$ Skip connections from encoder
Discriminator	6× Conv2d + LeakyReLU + Dropout : $3 \to n \to 2n \to 4n \to 8n \to 16n \to 1$ Output : $175^2 \to 87^2 \to 43^2 \to 21^2 \to 10^2 \to 1$
Loss	Reconstruction : L2 (masked region) Adversarial : WGAN-GP (masked region) Total : $\lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{rec}}\mathcal{L}_{\text{adv}}$

Note : h = hidden_dim, n = ndf (discriminator features)

Tableau 7 – Context Encoder Architecture

Component	Configuration
Architecture	hidden_dim : 64 mask_size : 50 ndf : 64 Discriminator dropout : 0.5
Optimizers	Generator : Adam(lr = 10^{-4}) Discriminator : Adam(lr = 10^{-5}) Batch size : 64 λ_{rec} : 0.999 λ_{rec} : 0.001

Tableau 8 – Context Encoder Training Configuration

4.4.2 Context Encoder reconstruction capacities

We will briefly go over the reconstruction capabilities of the model.

Based on the images, note that Fig 11 shows a very poor reconstruction of the masked area ; the model struggles to recover shapes, colors, or any discernible details. In contrast, the

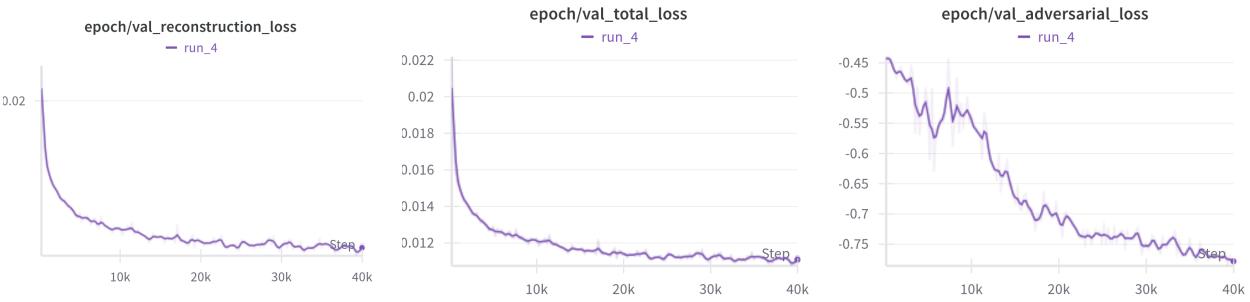

FIGURE 9 – Validation losses

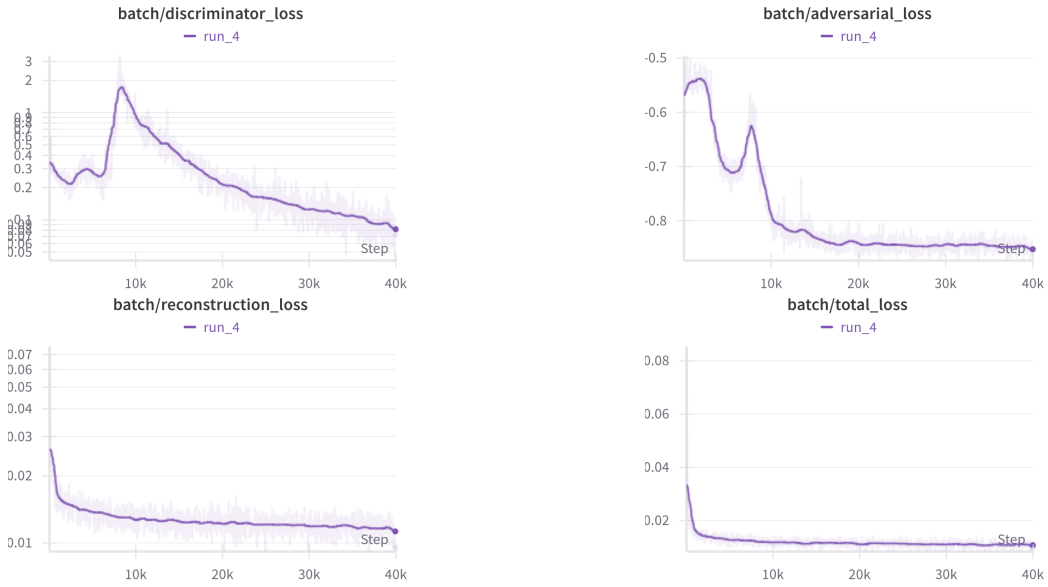

FIGURE 10 – Training losses

full-image reconstruction is much more accurate, indicating that the encoder is successfully capturing key features. This is promising for its role in the downstream classification task.

We can draw the first conclusion from this approach. First, the architecture's size are primarily influenced by the MLP between the encoder and decoder, which significantly increases training costs. Therefore, the training process is slow. Secondly, input reconstruction is a very challenging task and may be unnecessary for the application we are targeting. Many training resources will be dedicated to reconstruction, with considerable effort spent on "useless" details such as exact color and precise boundaries. Finally, there is a gap between the pretext and downstream tasks, as the encoder will be provided with the full image.

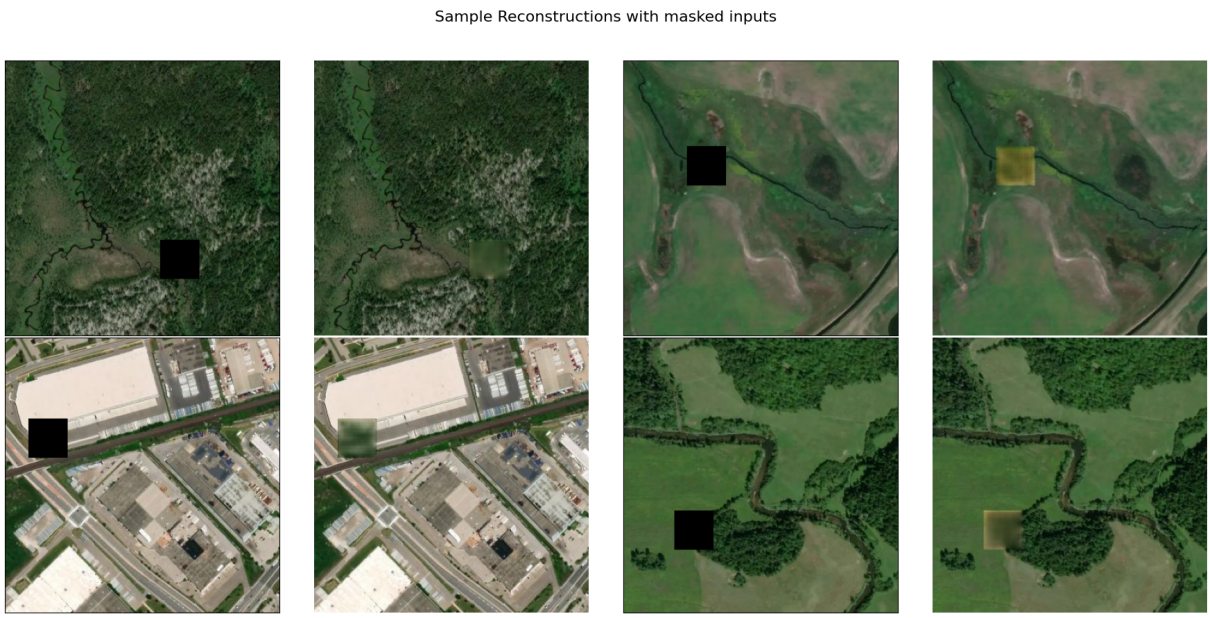

FIGURE 11 – Reconstruction of the masked zone with masked inputs

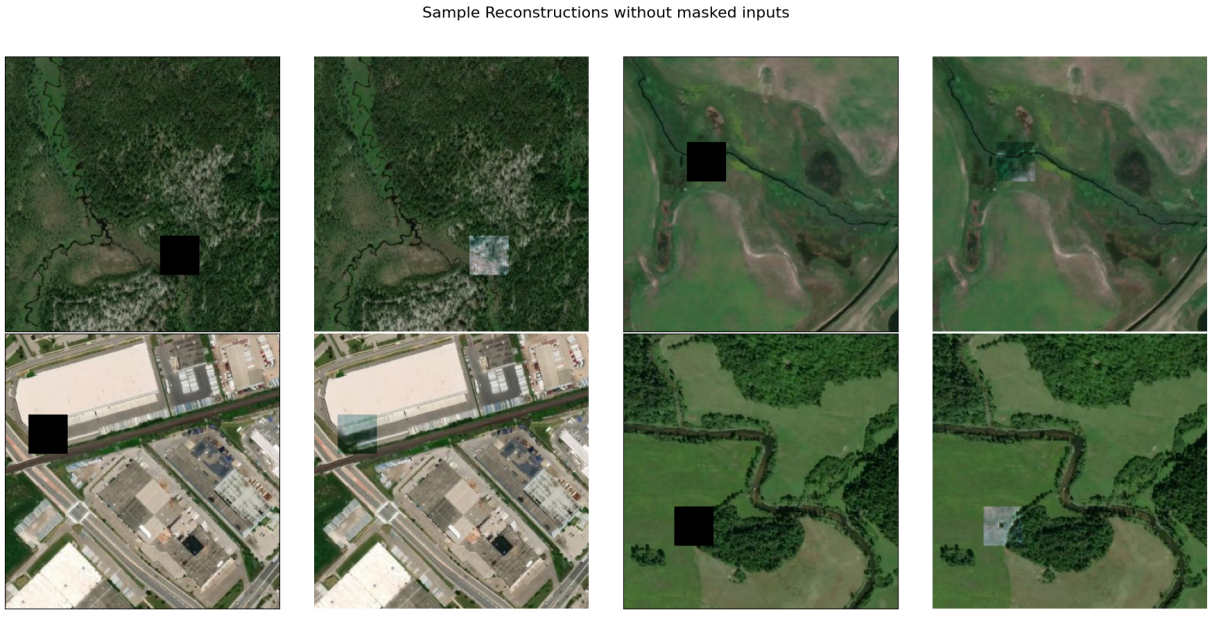

FIGURE 12 – Reconstruction of the same zone without masked inputs

4.4.3 Downstream task : classifier

With the pretrained context encoder, we will train an additional MLP classifier at the output of the encoder. We also add a global average pooling layer between the encoder and the classifier :

Component	Configuration
Input	$350 \times 350 \times 3$ RGB image
Backbone	Pretrained Context Encoder : $4\times$ Conv2d(k=4, s=2, p=1) + BN + LeakyReLU $3 \to h/8 \to h/4 \to h/2 \to h$ channels Output : $175^2 \to 87^2 \to 43^2 \to 21^2$
Classifier Head	Global Average Pooling : $21^2 \to 1^2$ Flatten : $h \times 1 \times 1 \to h$ FC1 : $h \to 256$ + ReLU + Dropout(0.2) FC2 : $256 \to 128$ + ReLU + Dropout(0.2) FC3 : $128 \to 2$ Output : Softmax
Training Options	Backbone freezing : Optional Loss : Cross-entropy

Note : h = hidden_dim (default : 64)

Tableau 9 – Wildfire Classifier Architecture

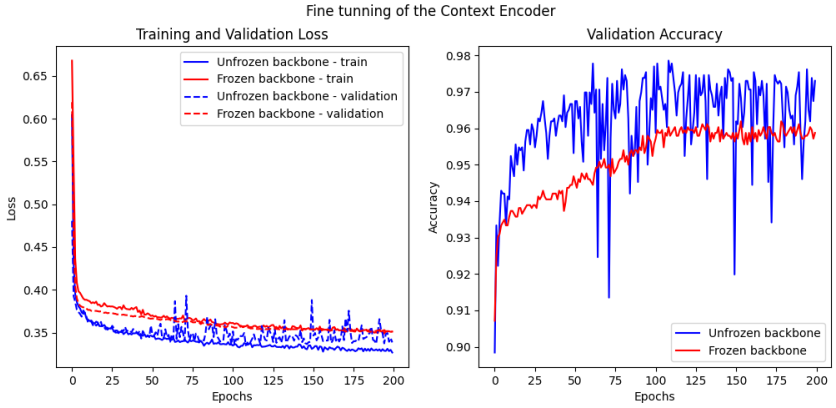

Figure 13

We will train this architecture for 200 epochs with and without freezing the backbone. Fig 13 displays the training and validation loss and validation accuracy across the training. The model is able to learn and the accuracy is reaching a plateau. Here are the final accuracy results for validation and test sets :

Model	Validation	Test
Frozen backbone	0.9659	0.9665
Unfrozen backbone	0.9786	**0.9740**

Tableau 10 – Fine tunning results

We can see that on both validation and test and with frozen and unfrozen backbone we outperform the baseline as well as the ViT approach. Logically, the unfrozen backbone approach is outperforming the frozen backbone approach. However, the gap is not huge, indicating that the representation learned by the pretraining is adequate. To improve those results, we could improve the architecture of the Context Encoder, or the learning procedure itself by tweaking the weights between reconstruction and adversarial loss.

5 Transfer learning & Finetuning

5.1 Using an existing pretrain model on a huge dataset

After trying to pretrain our model with dino, we found one based on swin v2 model that has been pretrained on the Sentinel-2 satellite imagery([2]). It is open imagery released by the European Space Agency which contains many satellite images. We thus tried to finetune this model on the val dataset to see what performances it could achieves. To do so, we replaced the head with either one final linear layer to finetune or a bigger one with 3 layers, we tried to train only these layers as well as the whole architecture and see what performances we could reach.

Below are the accuracy curves compared to the baseline :

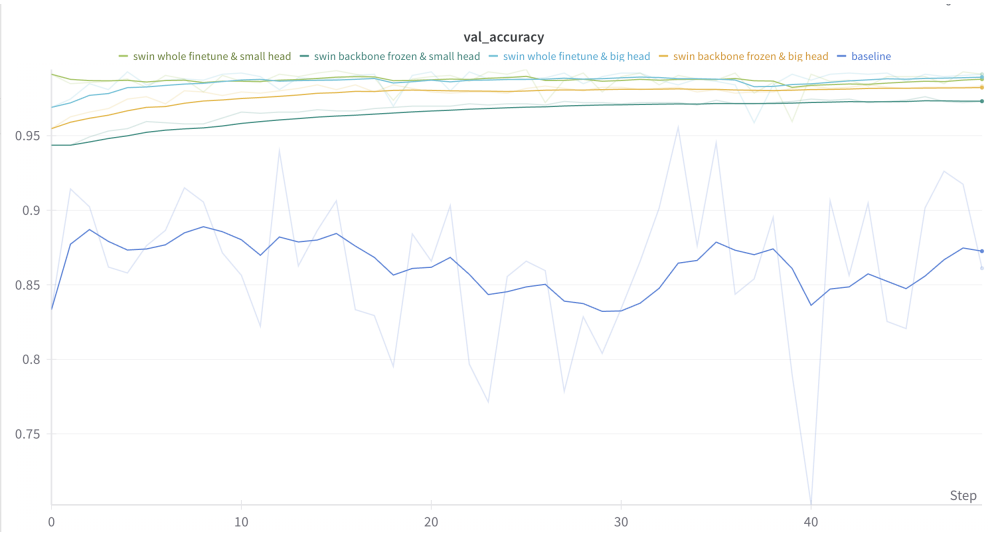

FIGURE 14 – Swin validation accuracy curves

We first see that the performances are more promising than the last one and are far more stable which might be due to the size of the model which is bigger than the ViT small. It is especially outstanding compared to the baseline

The final accuracy achieved on the test set are as follow :

Model	Accuracy
Baseline	0.9527
Swin Whole Finetune Big Head	0.9914
Swin Whole Finetune Small Head	**0.9922**
Swin Backbone Frozen Big Head	0.9852
Swin Backbone Frozen Small Head	0.9808

Tableau 11 – Test Accuracy Results for Swin with different finetuning configurations

Here we achieve almost perfect accuracy showing the usefulness of a bigger model pretrained on much more data. The experiences with the frozen backbone perform very well demonstrating that the representation learned is far better than the one we had with the previous ViT model. Overall when finetuning the whole model we get outstanding performances.

6 Discussion

Tab. 12 provides an overview of all the approaches employed to address the task in this project. Unsurprisingly, the large pretrained architecture [2] significantly outperforms other methods, as it is an optimized architecture trained on a substantial amount of data. The second most effective approach is the Context Encoder [4]. Although the model's reconstruction capabilities are limited, as discussed in the dedicated section, the encoder is sufficiently robust to achieve high accuracy. The Vision Transformer trained using DINO [1] matches the baseline accuracy. The other approaches do not perform as well as the baseline, which may be attributed to the architecture, training process, or methodological choices. Indeed, self-supervised learning methods are highly dependent on the type of images used.

7 Conclusion

This project explored various approaches to tackle the task of wildfire detection using a provided dataset. The results demonstrate that large pretrained architectures, such as the Swin Transformer, significantly outperform other methods due to their optimization and extensive training on large datasets. The Context Encoder also proved to be a robust approach, despite its limited reconstruction capabilities. Other self-supervised learning methods, including DINO and pseudo-labeling, showed varying degrees of success, highlighting the importance of careful methodological choices and the dependency on the type of images used. Overall, this project underscores the effectiveness of leveraging pretrained models and self-supervised learning techniques in enhancing the performance of wildfire detection systems.

Method	**Test Accuracy**
Baseline CNN	0.9527
Pseudo-labeling ($\tau = 0.95$)	0.9330
Pseudo-labeling ($\tau = 0.98$)	0.9370
ViT Whole Finetune Big Head	0.9519
ViT Whole Finetune Small Head	0.9589
ViT Backbone Frozen Big Head	0.9323
ViT Backbone Frozen Small Head	0.9093
Coloration Detection	0.9400
Context Encoder Frozen Backbone	0.9665
Context Encoder Unfrozen Backbone	0.9740
Swin Whole Finetune Big Head	0.9914
Swin Whole Finetune Small Head	**0.9922**
Swin Backbone Frozen Big Head	0.9852
Swin Backbone Frozen Small Head	0.9808

TABLEAU 12 – Summary of test accuracy results for different approaches.

Références

[1] Dino algorithm and vit implementation. https://github.com/facebookresearch/dino, 2021.

[2] Satlas : Open ai-generated geospatial data. https://github.com/allenai/satlas?tab=readme-ov-file, 2023.

[3] Wildfire prediction dataset (satellite images). https://www.kaggle.com/datasets/abdelghaniaaba/wildfire-prediction-dataset, 2023.

[4] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders : Feature learning by inpainting. *CoRR*, abs/1604.07379, 2016.