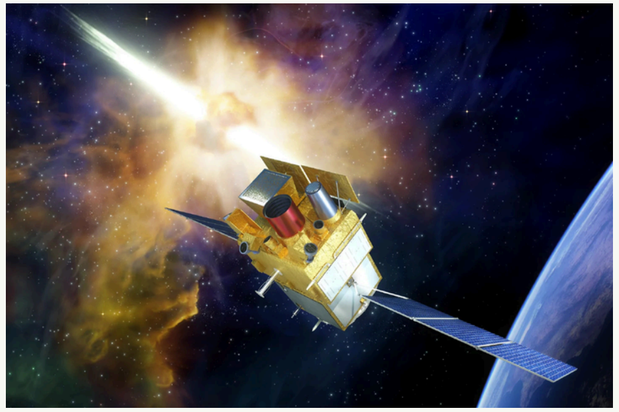
Machine Learning for gamma-ray burst images classification

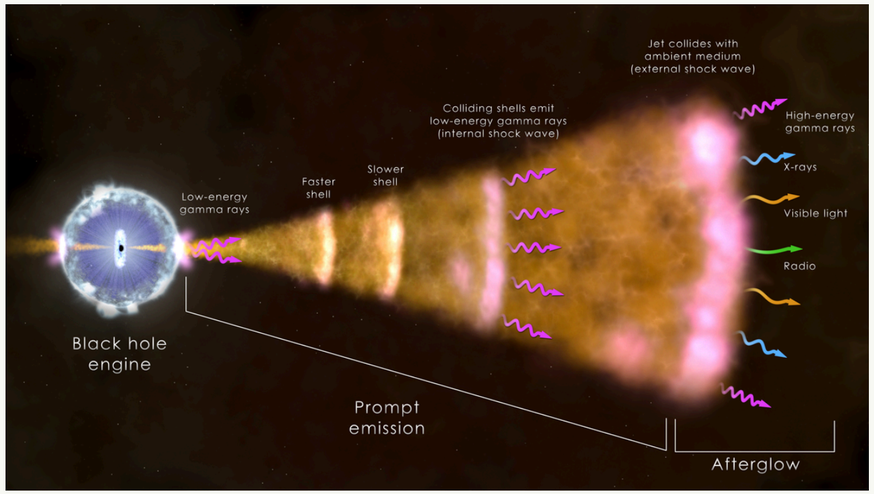


B. Hubert, CEA (Irfu/DAp)
S. Schanne, CEA (Irfu/DAp)

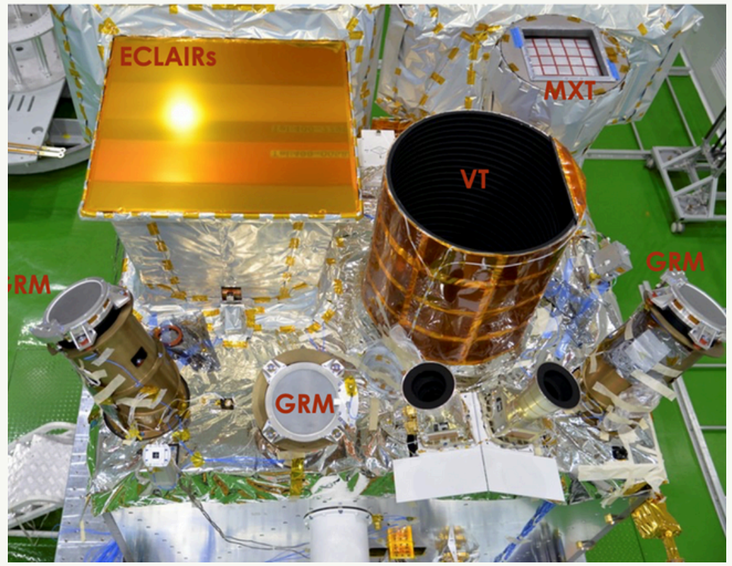
09/02/2026, Saclay, In The Art

SVOM Mission / ECLAIRs

- Satellite launched in June 2024, French-Chinese collaboration between CNSA+CNES (+CEA+CNRS)
- **Dedicated to Gamma-Ray Burst (GRB) studies** (formation of black holes in distant Universe)



Representation of a gamma ray burst



Different instruments on SVOM Satellite

- **Multiple instrumets Onboard :** ECLAIRs, VT, MXT, GRM… To analyze GRBs in different parts of the spectrum
- **ECLAIRs + Onboard trigger :** detection of new transient gamma-ray source, localization in the field of view (FoV 2 sr), repointing spacecraft in 2 min for follow-up (Visible & X-rays)

SVOM Mission / ECLAIRs trigger

Some **statistics** about ECLAIRs trigger up **Nov 2025** :

- **72 GRBs** detected
- **379 Known Gamma-ray sources** (238 in onboard CAT, 141 not in CAT)
- **380 false alerts**
- **848 triggers** (682 VHF SubImages from **IMT**, 166 VHF Shadowgrams from **CRT**)

There are 2 independent trigger algorithms running onboard ECLAIRs:

- **CRT**: searches for count-rate increase on timewindows from 10 ms to 20.48 s duration, then imaging best timewindow found (max 1 image every 2.56 s), search excess

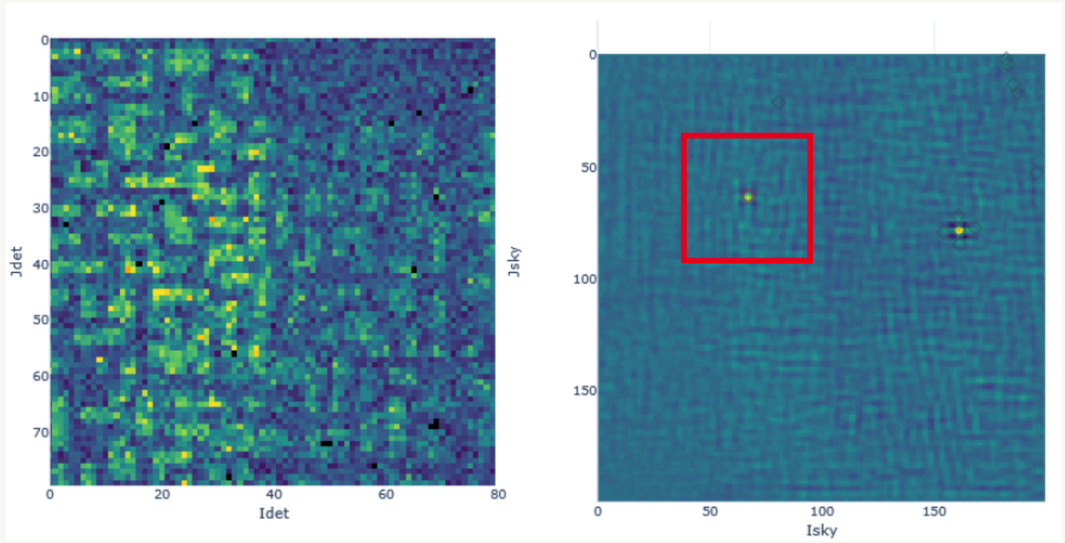
  => GRB Alert sequence with **VHF Shadowgram => not used in this work**
- **IMT**: produces sky images every 20.48 s (on 4 energy bands), stacks the images to reach timewindows from 20.48s to 22 min, search best excess not in onboard source CAT

  => GRB Alert sequence with **VHF Sub-Image => studied in this work**

SVOM Mission / ECLAIRs

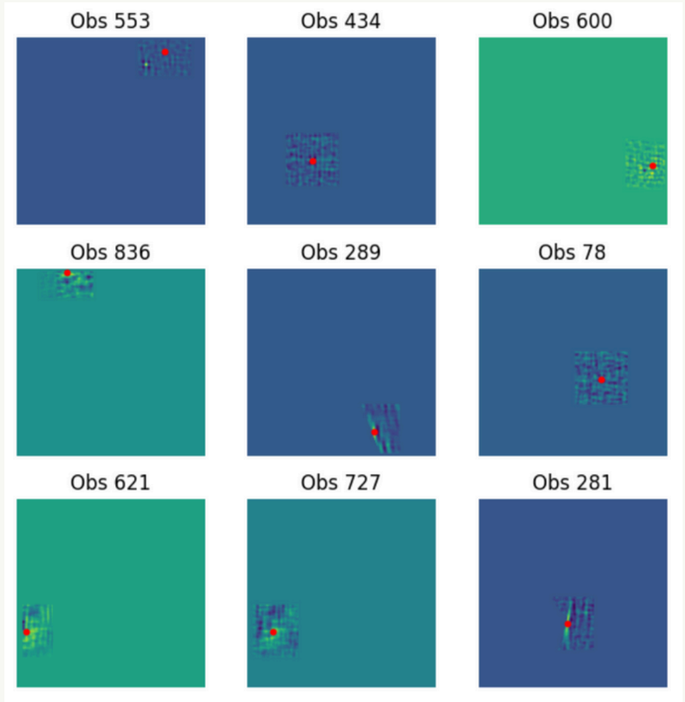
What happens for real onboard ?

The onboard trigger calculates SNR images of the sky (200x200 pixels) every few seconds.



On this image, differents values are calculated : maximum value, standard deviation, difference between maximum and second maximum...

Those values go through different filters/ thresholds, and triggers an **alert sequence** sent to the ground over the VHF network.
A sub image (56x56 pixels) is sent at the end of this alert sequence (for a trigger by IMT).



SVOM Mission / ECLAIRs

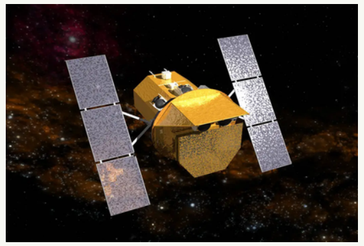
When an alert sequence is sent:

- Subimage to the ground
- Automatic request for a slew (repointing of the satellite)

Einstein probe Telescope

On ground a person, the "Burst Advocate" on shift, has to decide:

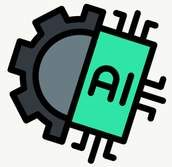
- was it a True GRB ? => organize ground follow-up with visible telescopes, space follow-up with other satellites (Swift and Einstein Probe)
- was it a False Alert ? => cancel the SVOM repointing, invalidate follow-up started



Swift Telescope

After ~1.5 year of operations: about 75 True GRBs + hundreds known sources + hundreds False Alerts => **huge work for Burst Advocates**

➡ **Automatic classification ?** Machine Learning algorithms applied to sub-images to help decision making



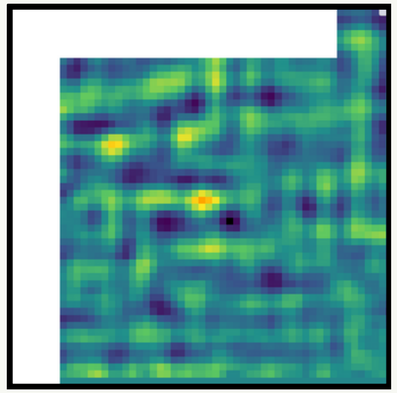
What is the goal ? What are the main difficulties ?

To summarize the goal : **Help the decision making for a GRB alert**

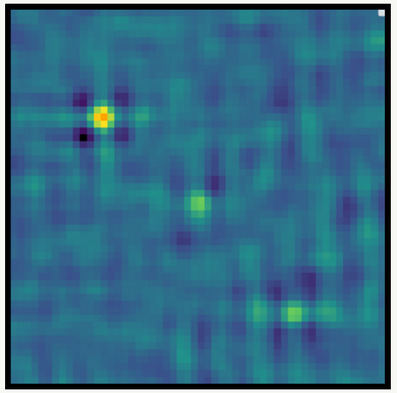
Basically : **Computer Vision** → a subfield of AI that equips machines with the ability to process, analyze and interpret images (or videos).

Not so basic in our case, why ? Our database :

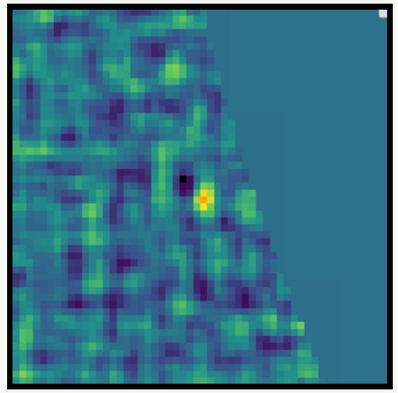
- 540 images (true and false triggers combined)
- incomplete images (not 56x56)
- unclear cases (where even the human can't really decide wether it is a true GRB or not)



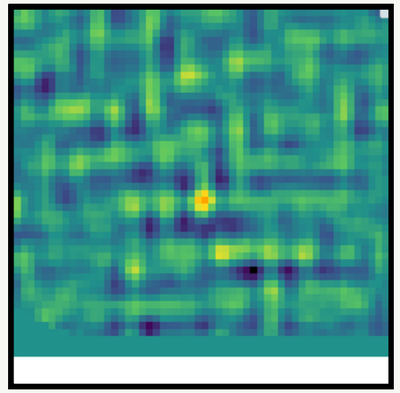
Incomplete images



Unclear decision



Earth appearing



Incomplete + unclear

What is the goal ? What are the main difficulties ?

Main problem with our small database :

- **overfitting** : when the model is very efficient on the training set but not on the test set. The model learns by heart images and can't adapt.

Main problem with incomplete images :

- Convolutional Neural Networks (CNN) require **as inputs a fixed size.** How can we fill the images without disturbing the model ?

Main problem with unclear cases :

- If even humans hesitate on the decision for a subimage, with lots of additional data, how can a CNN predict easily on those unclear cases ? (spoiler, it can't)

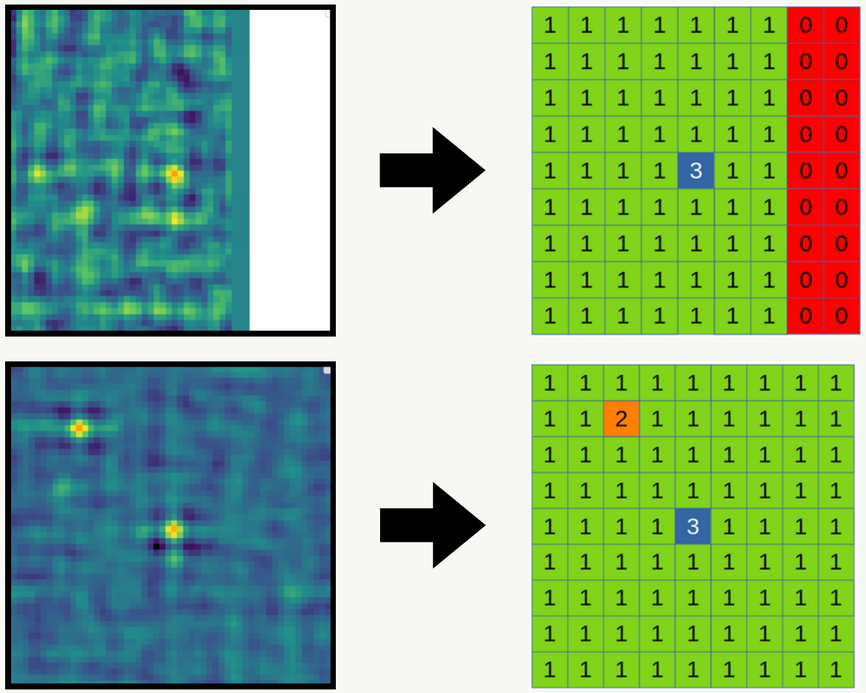
Multiple solutions :

- **Transfer learning** → use a pre-trained model to **prevent overfitting**
- **Data augmentation** → increase artificially the dataset to **prevent overfitting**
- Mask associated with a subimage → **fill the incomplete images** without disturbing the model

What solutions do we consider ?

We considered **multiple solutions** :
1) Hand made model, trained from scratch
2) Hand made model, trained from scratch, with a mask associated with each subimage
3) Transfer learning
[4) Autoencoders (did not have time to implement it yet)]
[5) IsolationForest (new idea)]

Mask concept :



- “0” for what the model should not take into account (missing part of a sub-image, FOV edges) → **red**
- “1” for the background (useful data) → **green**
- “2” for known sources → **orange**
- “3” for the pixel that triggered the alert → **blue**

What solutions do we consider ?

Here are the outcomes of basic tests/studies/researchs to identify the best solution :

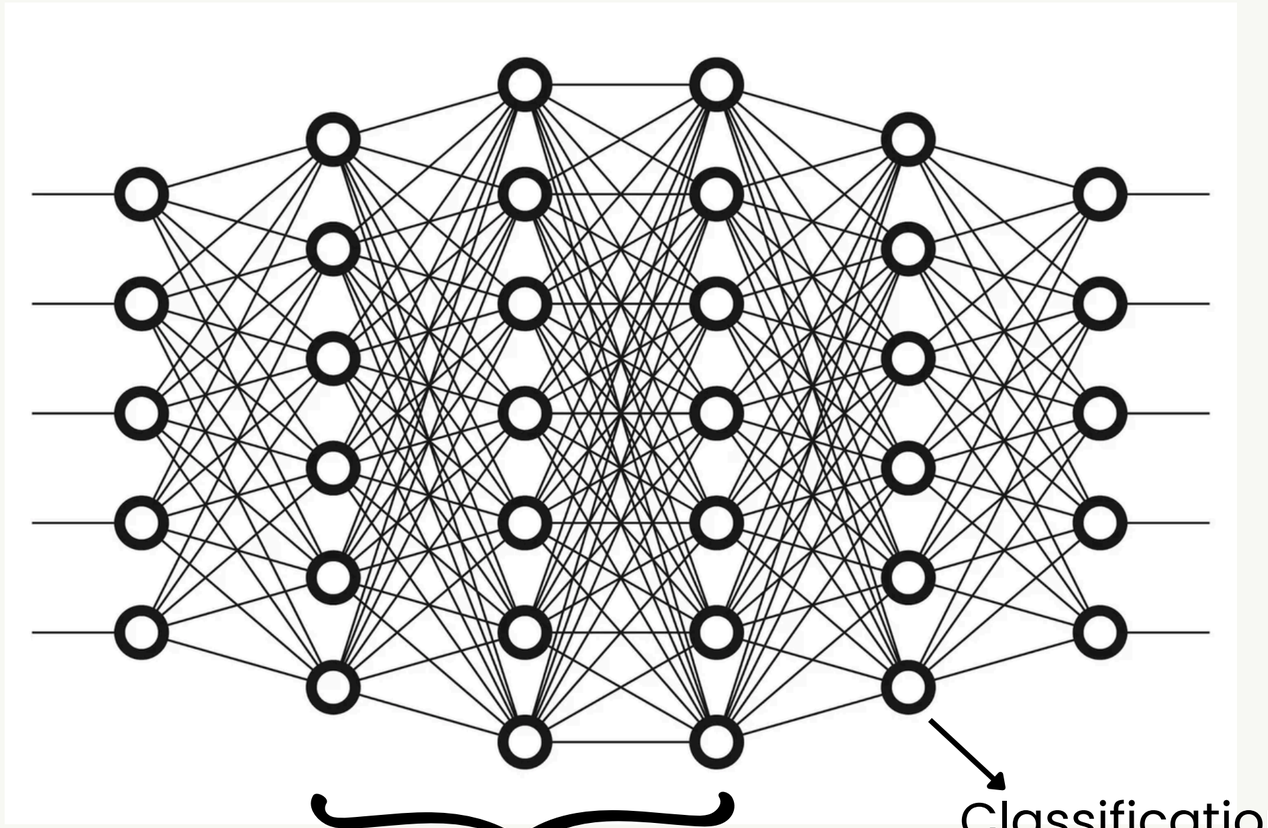
- **Filling the images with “0” is enough**, the additional mask does not help a lot, since the model identify on his own which data is useful
- **Indicating the triggered pixel (value “3”) is not useful,** since the sub-image is always centered on the triggered pixel
- Training a model **from scratch has lots of defaults** :
    - A lot of parameters possible (number of layers, number of parameters, what type of layers…) → I do not have the expertise to do so
    - The performance is poor compared to transfer learning
- **Transfer Learning is the most promising** (and environment friendly)

Combined with other methods (Data Augmentation…) → we can avoid overfitting and have good performance

Transfer Learning

1) What is transfer learning ?

The concept of Transfer Learning is to use a **model trained on millions** of generic images (typically ImageNet set of images) and to **adapt it to our dataset**
Here is the general idea for a CNN :

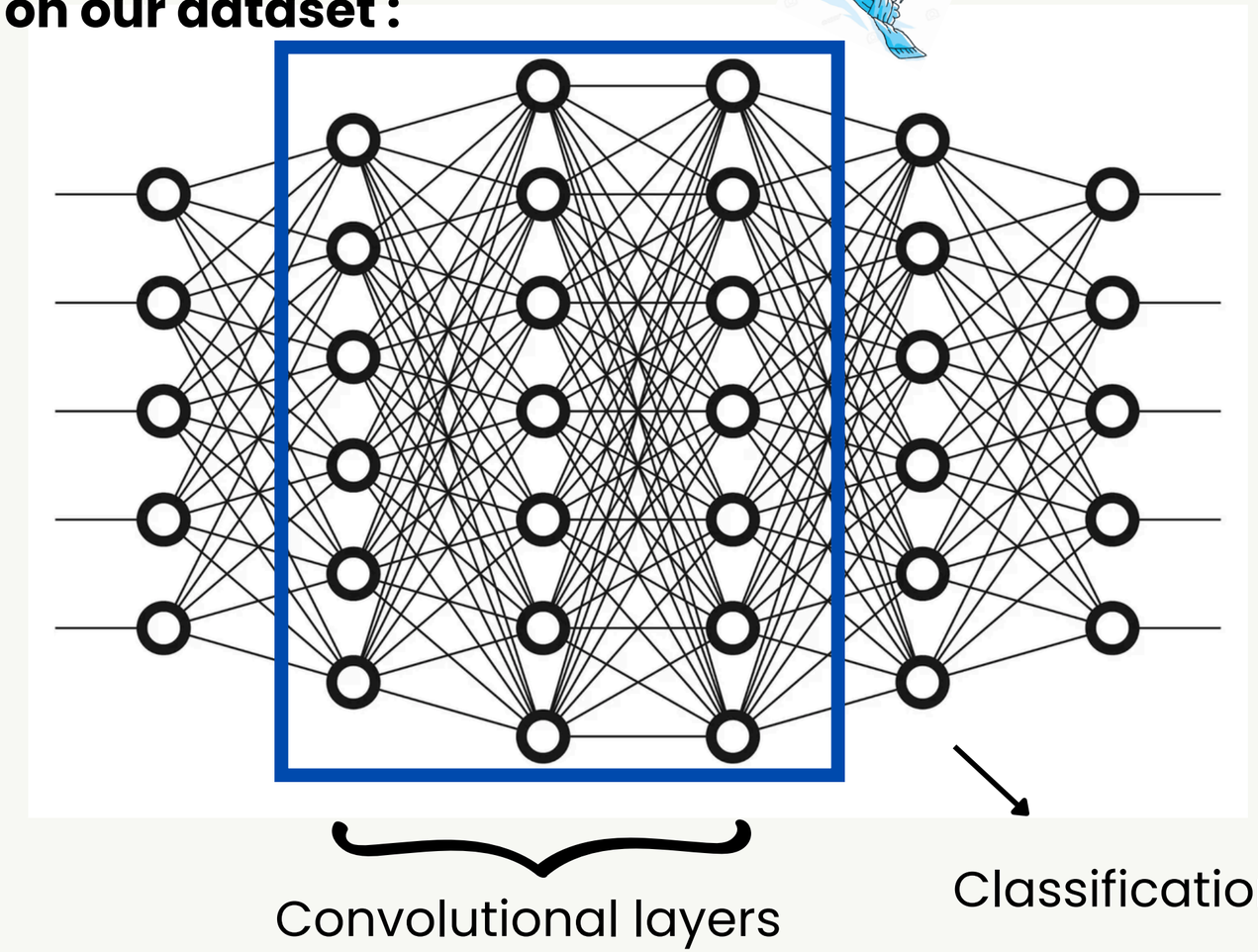


Convolutional layers are responsible for the patterns and strcutures recognition

Transfer Learning

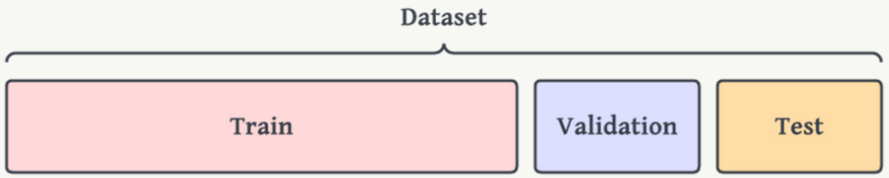
1) What is transfer learning ?

During training on our dataset :



Only the wieghts and biases of the classification layer will update during training

Transfer Learning

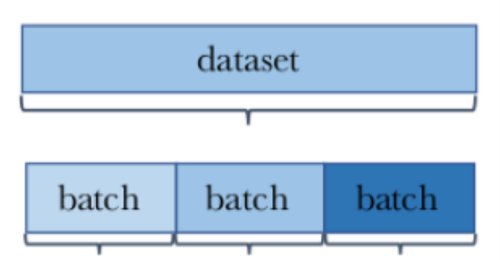


2) How to implement Transfer Learning ?

- **Split train/validation/test** : (not specific to transfer learning) we split our full dataset into 3

The choice I made is a 50/20/30 repartition :
We need a big amount of data to train but we are going to do data augmentation on it so 50% is ok
Validation set is just to follow if our model is really learning something
Test set is here to evaluate our model at the end (can't be too small otherwise it's not representative)

- We **prepare our data** to the pre-trained model :
  - Resize from 56x56 to 224x224
  - Normalisation
  - Data augmentation (not necessary generally speaking)
  - Transformation into tensors (typical inputs for a model)



- We divide our **train dataset** into **batches** of a certain size (eg batch size of 50 for 200 images is 4 batches)

- We load the pre-trained model, and we start the **training part** on our dataset

Transfer Learning

3) How do we optimize model's training

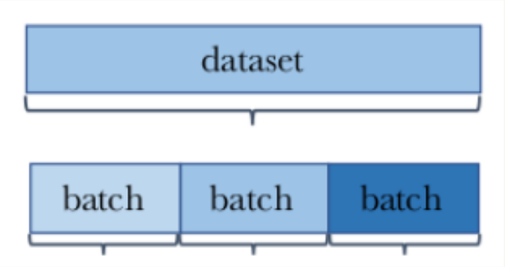
What parameters can we play on ?

- split (50/20/30 ou 70/15/15 ou ...)
- data augmentation
- normalisation : global ? local ? others ?
- batch size
- number of epochs
- loss calculation choice
- scheduler choice
- optimizer choice
- learning rate
- which data (with each subimage)
- fine tuning on or off

Our goal : have the **best** model
"best" = a model that predicts the right decision every time

=> LOTS OF PARAMETERS : we need to find which value/choice is the best for each parameter (we understand now why a hand made model is even more complex, because even more parameters to adjust)

Transfer Learning



4) What happens during training ?

<u>We consider 1 batch of images :</u> (reminder, we divided our train dataset into batches)

- We give this batch to the model
- It tries to predict the good output
- We calculate the loss (CrossEntropy, MSE…)

} The model answers with what it knows
=
Forward pass

- We calculate the loss gradient
- It gives, layer by layer, how each weight influences the decision

} The model understands where it went wrong
=
Backward pass

- We update the parameters (weights and biases) of every neurons → depending of the learning rate

} The model updates its weights to reduce the error
(choice of the **optimizer**)

Transfer Learning

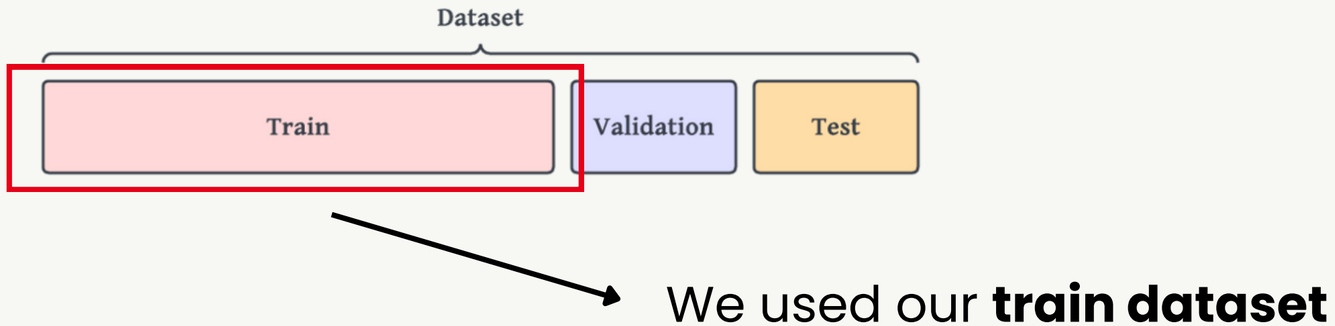
4) What happens during training ?

To summarize : for 1 batch → forward pass / backward pass / update of weights

Exemple : we have 500 images, and batches of 50 images, then we have 10 batches.
That means that we will have to do 10 times the forward/backward/update process

When the dataset have seen the 10 batches, we completed **an epoch**
To make it simple : an epoch is done **when the model has seen all the images from the dataset**

Now, we can repeat this multiple times (choice of the number of epochs)



We used our **train dataset**

Transfer Learning

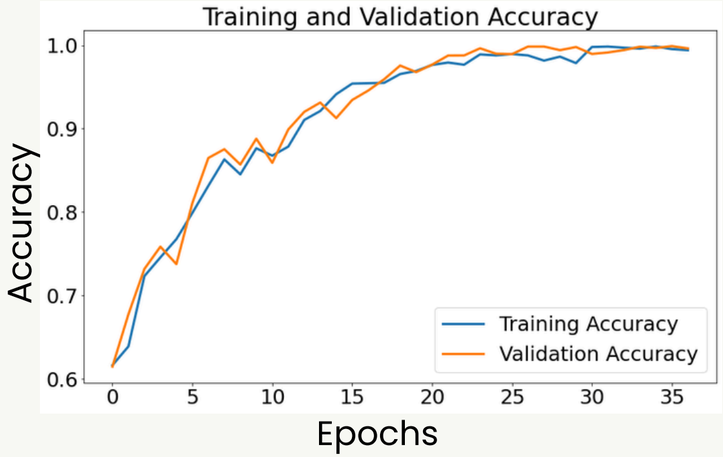
4) What happens during training ?

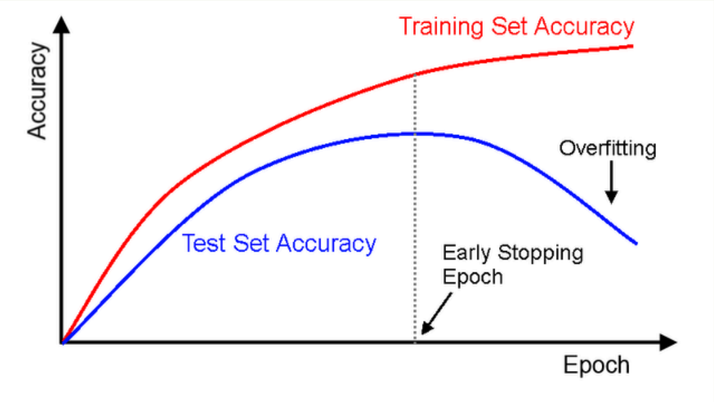
Reminder, an epoch is done when the model has seen all the images from the train dataset

But what the validation set is for ?
→ At the end of each epoch, we “validate” the model

The model **tries to generalize** what he learnt from the training. We calculate different metrics, to follow if the model is still learning or if it’s stuck somewhere.

⚠ It doesn’t adjust its wheigts during validation ⚠



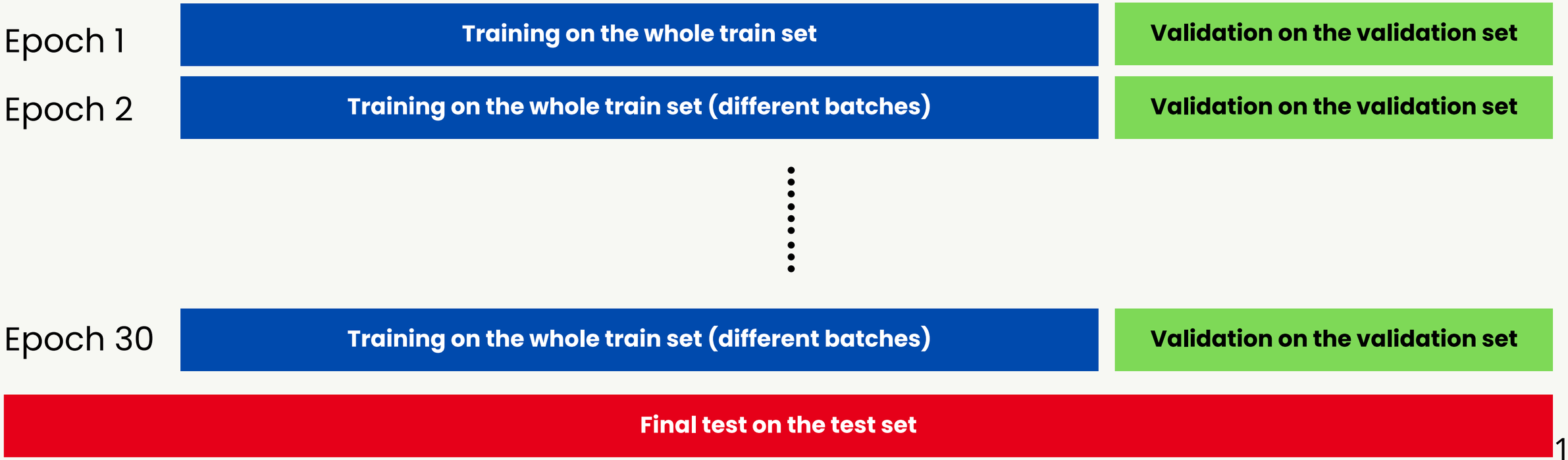


Transfer Learning

5) What happens after training ?

When the training is complete, **the model is fixed** and won't move anymore (as long as we don't train it again).
We can now **test** it on the test set, a set of images he has never seen before.

<u>To summarize :</u>



Transfer Learning

5) What happens after training ?

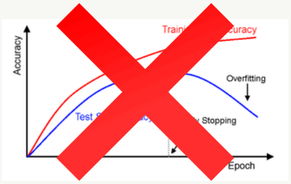
With that in mind, we now want to **adjust every parameter** possible to have the best model.
In order to do that, we need some indicators/values that help us make a decision :

A) Performance :

- Accuracy (% of good predictions on the test set)
- Precision (ratio between true positive and all positives)

B) Least overfit :

- Train accuracy VS validation accuracy curve



C) Good stability :

- Validation accuracy stable at the end ?

D) Probabilites repartition :

- At what point the incorrect prediction are separated from the good ones

Transfer Learning

5) Analysis

A) Performance : the simplest analysis (compare different values)

Exemple :

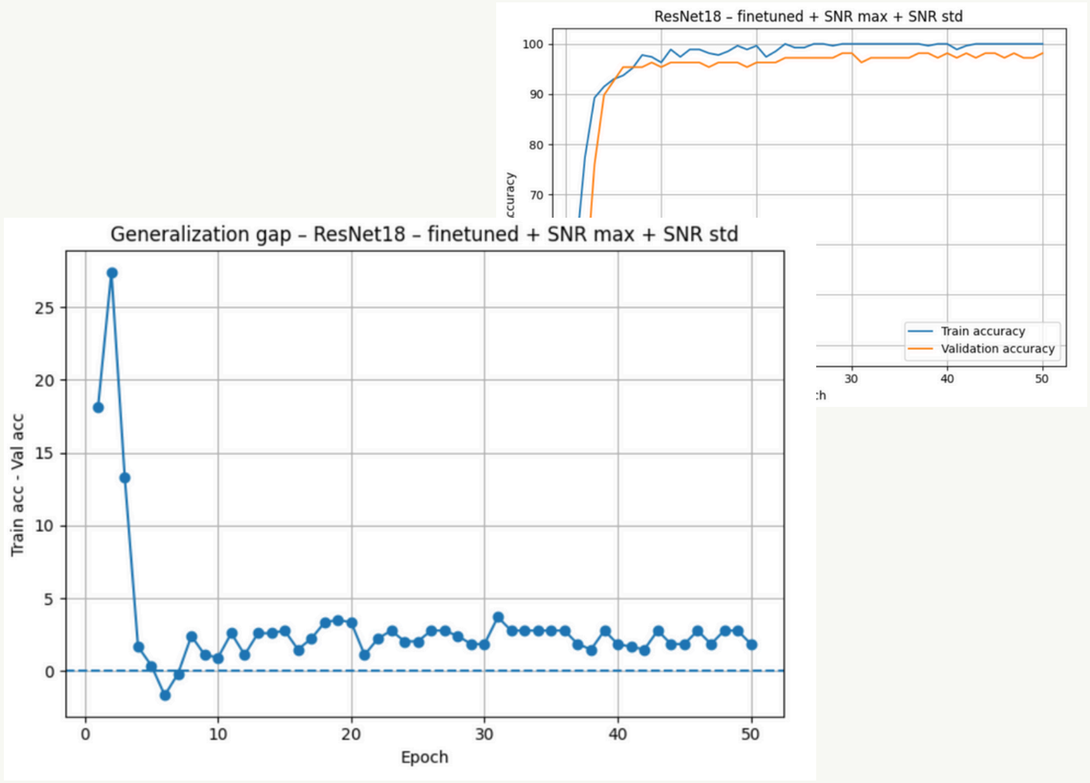
	experiment	model_name	accuracy	precision	recall	f1	train_time_sec	batch_size	lr	epochs	r
0	resnet18_with_SNRmax_finetune_normvraimentMAX	ResNet18_with_SNRmax	97.530864	0.961538	1.0	0.980392	232.911536	16	0.0001	30	
1	resnet18_with_SNRmax_finetune	ResNet18_with_SNRmax	96.296296	0.943396	1.0	0.970874	272.757874	16	0.0001	40	

It gives a **good idea** if a model performs well, but it is absolutely **not enough** to make a decision

Transfer Learning

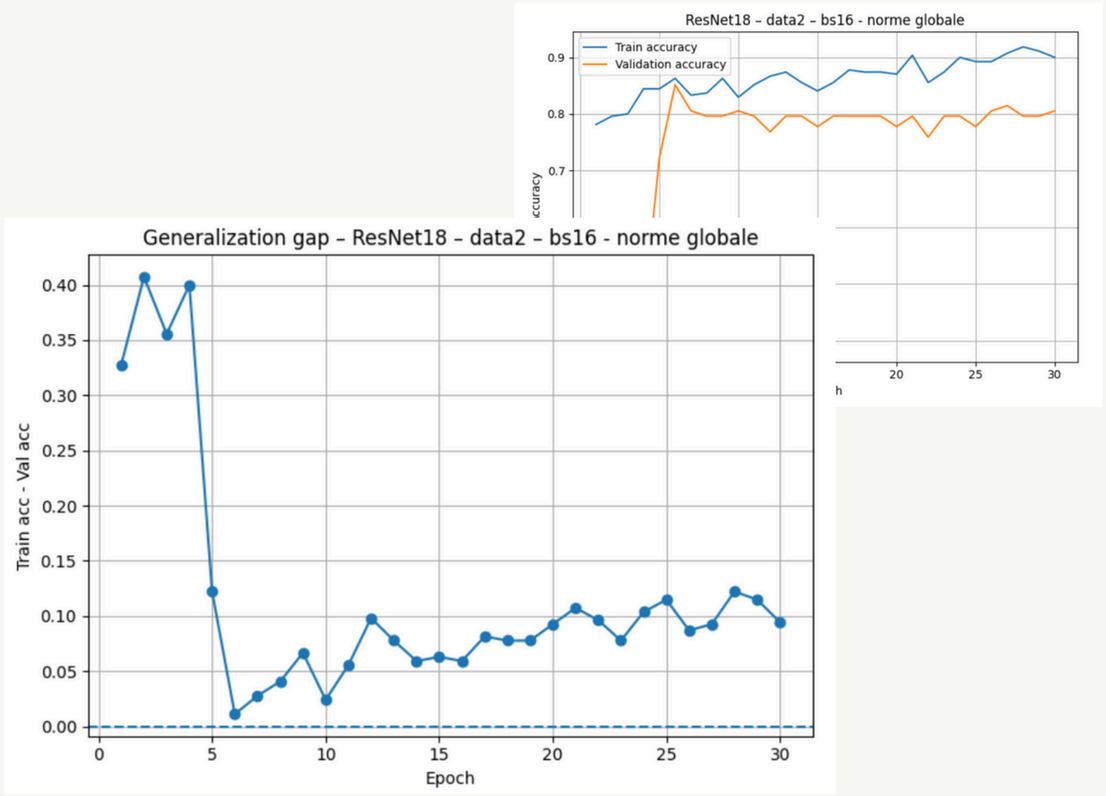
5) Analysis

B) Least overfit possible : we compare performance on the train set and on the validation set



Well fitted model : small difference between validation and training cruve

VS

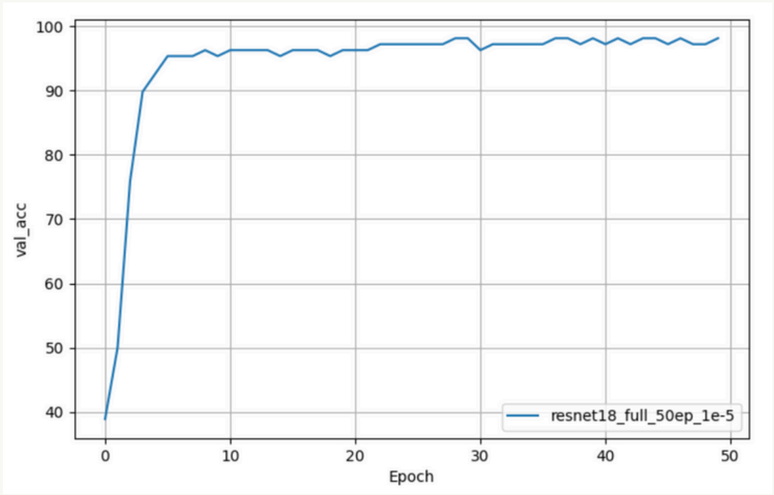


Overfitting model : train accuracy above validation accuracy

Transfer Learning

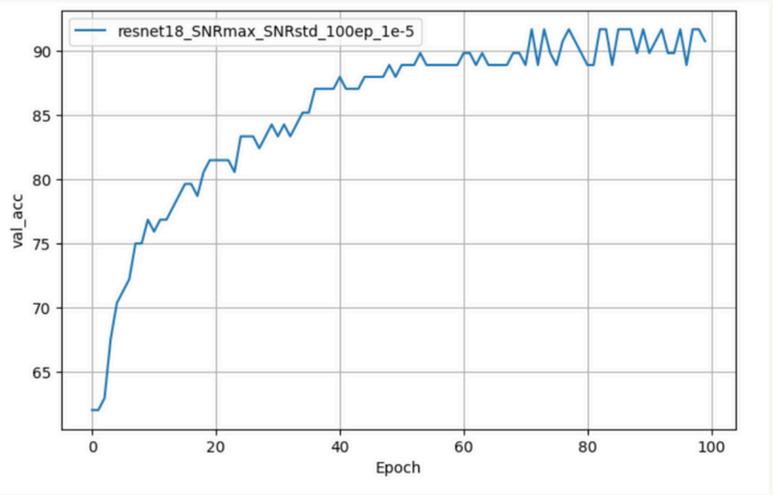
5) Analysis

C) Good stability : we want the validation accuracy curve to be stable at the end



Good stability

VS



Bad stability

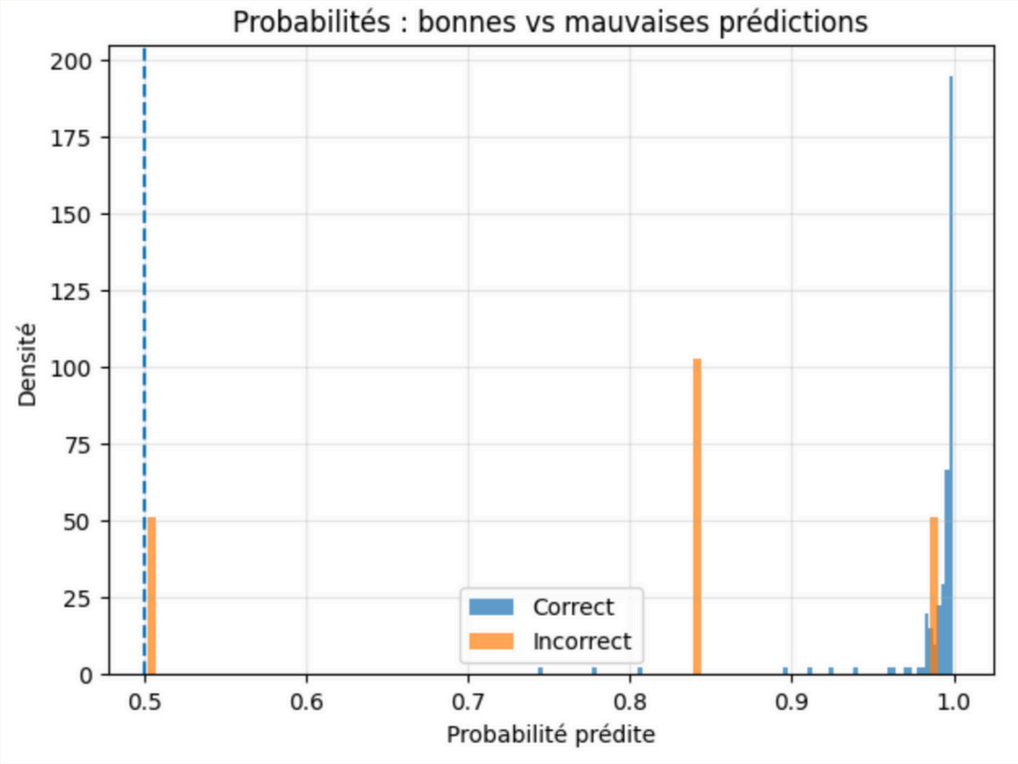
This stability depends a lot on the value of the **learning rate** and the presence of **fine tuning**

Transfer Learning

5) Analysis

D) Probabilites repartition :

At the end, we want a model that can predict with a lot of certainty if an alert is a true GRB or a false alert.
BUT in the worst case, we prefer it says “I don’t know” than saying a verdict it’s not sure of.



Here are the probabilities the model calculated before giving an answer (0 or 1, true or false)

- In **blue**, the good prediction
- In **orange**, the bad prediction

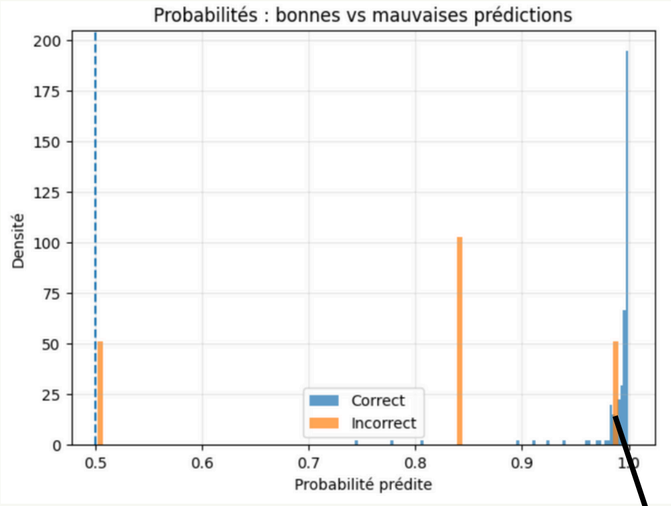
If the bad prediction are far from the good ones, we can set a threshold to signal : “The model is not sure of its prediction”

Transfer Learning

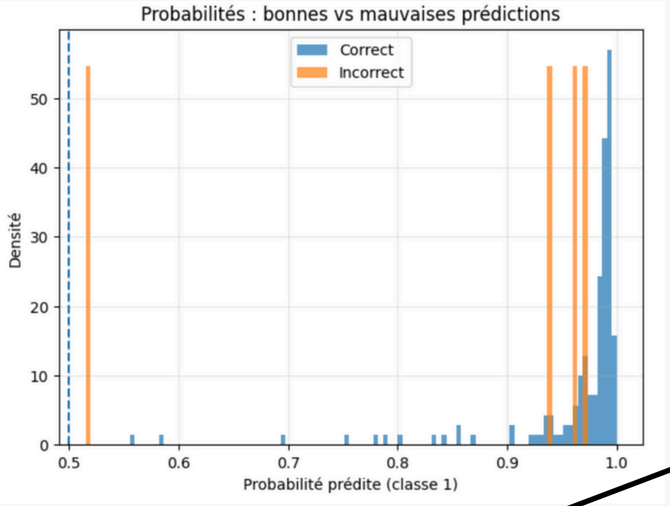
5) Analysis

D) Probabilites repartition :

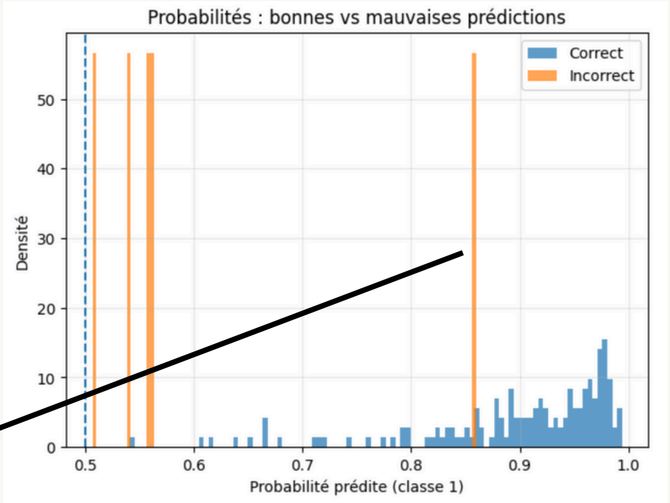
The probabilites repartition aren't always the same for each model, depending on the number of epochs, the learning rate...



Easy to set a thershold



Hard to set a threshold



Easy to set a thershold, different repartition

(We ignore this error)

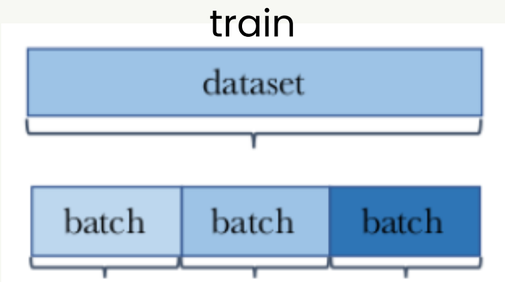
Transfer Learning

6) Final models

After all those experiences, I kept 3 models that combined good score on all the indicators.
Here are the parameters :

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

explained earlier



Model 1, 2, 3

Transfer Learning

6) Final models

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Data Augmentation

- Goal : prevent overfitting by artificially increasing the amount of data
- The choice of which operation we do to our images is crucial. We must keep the physics sense behind the operation.
- Here, the augmentation is made in pre-processing. But it is possible to make it randomly during training, and that considerably increase the total number of different images (a bit harder to do)

Transfer Learning

6) Final models

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Normalisation

- Goal : help the model to learn easier
- The principle of “learning” is based on gradient. If the data is too “wide” (high values in an image and low values on another), the gradients will be unstable, it becomes hard to optimize.
- The choice of normalisation depends on the data :
  - No normalisation ?
  - Global normalisation ? (max of the dataset)
  - **Local normalisation ? (max of the image)**

Additional data

- To compensate the fact that we used local normalisation, we give to the model the max snr and the standard deviation of the snr

Transfer Learning

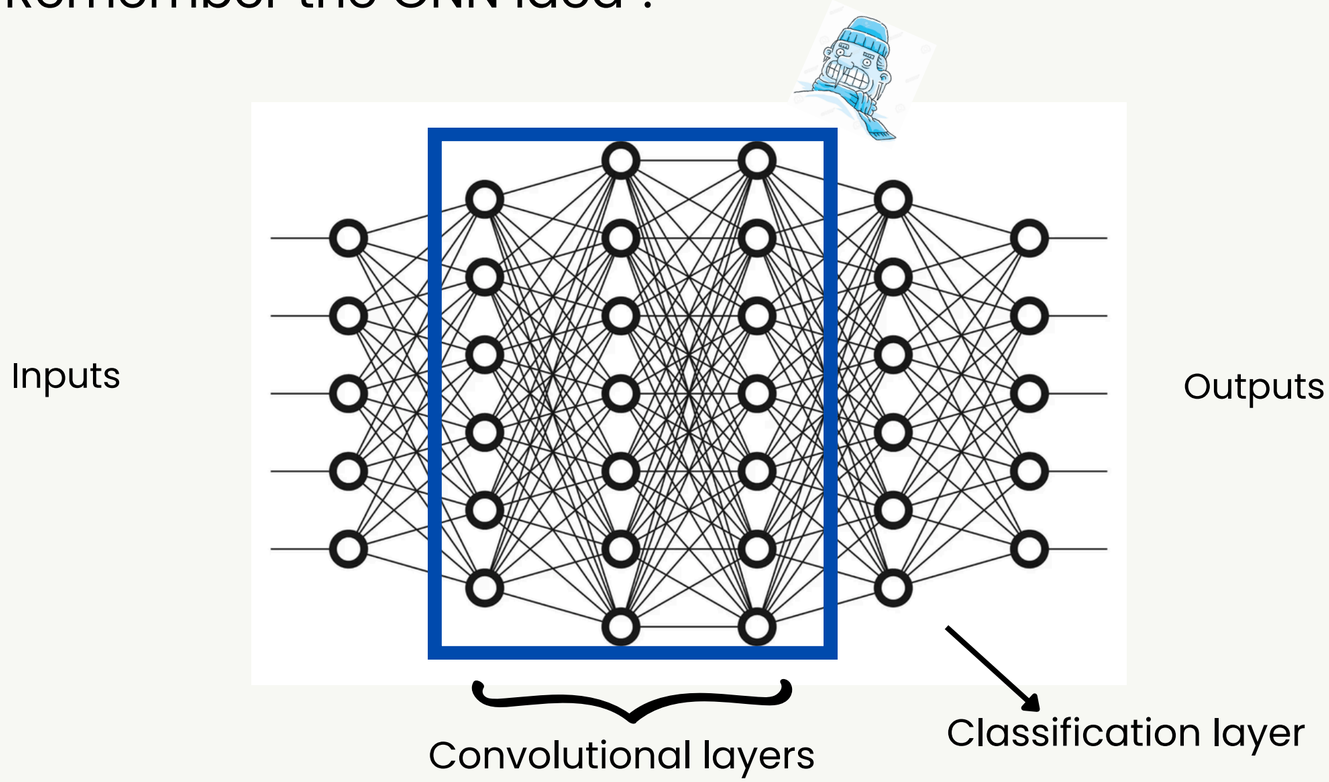
6) Final models

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Fine tuning

- Goal : help the model to adapt even better to our dataset

Remember the CNN idea ?



Transfer Learning

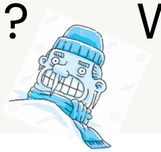
6) Final models

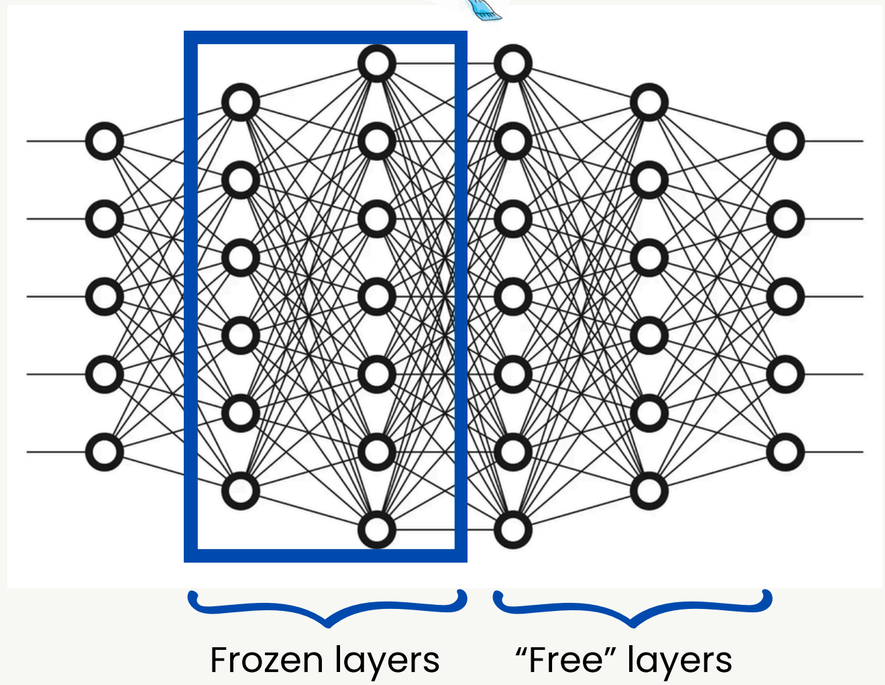
- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Fine tuning

- Goal : help the model to adapt even better to our dataset

Remember the CNN idea ? We unfreeze a layer





Transfer Learning

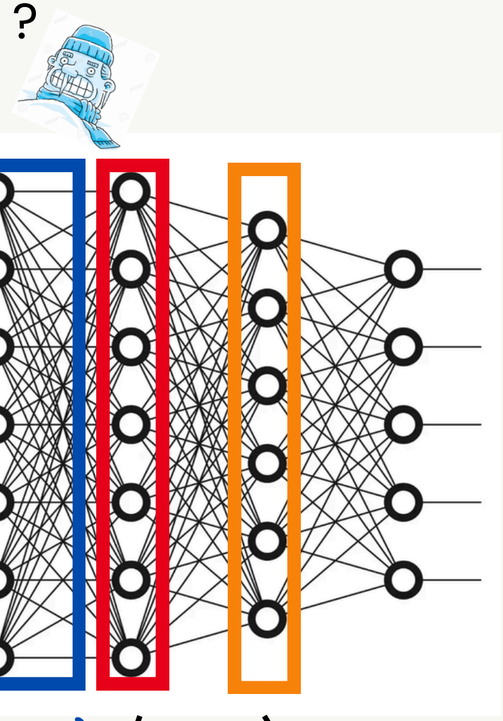
6) Final models

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Fine tuning

- Goal : help the model to adapt even better to our dataset

Remember the CNN idea ?



Transfer Learning

6) Final models

- jeu de données : dataset_2.pkl
- split : 50/20/30
- augmentation : horizontal flip + rotation 90°
- normalisation : locale (img / snr_max(img))
- batch size : 16
- Utilisation du snr max en entrée : oui
- Utilisation du snr std en entrée : oui
- fine tuning : oui
- epochs : 30 / 200 / 200
- loss : CrossEntropyLoss
- scheduler : oui -> lr_scheduler.ReduceLROnPlateau
- optimiseur : Adam
- learning rate fc : 1e-4 / 1e-6 / 1e-4
- learning rate layer4 : 1e-5 / 1e-6 / 1e-4

Loss

- Goal : calculate the error on predictions
- CrossEntropy for classification, MSE for regression

Scheduler

- Goal : adapt the learning rate to help the learning
- Several methods :
  - StepScheduler → lower the learning rate every X epochs
  - ReduceLROnPlateau → lower the learning rate when the validation loss doesn't lower

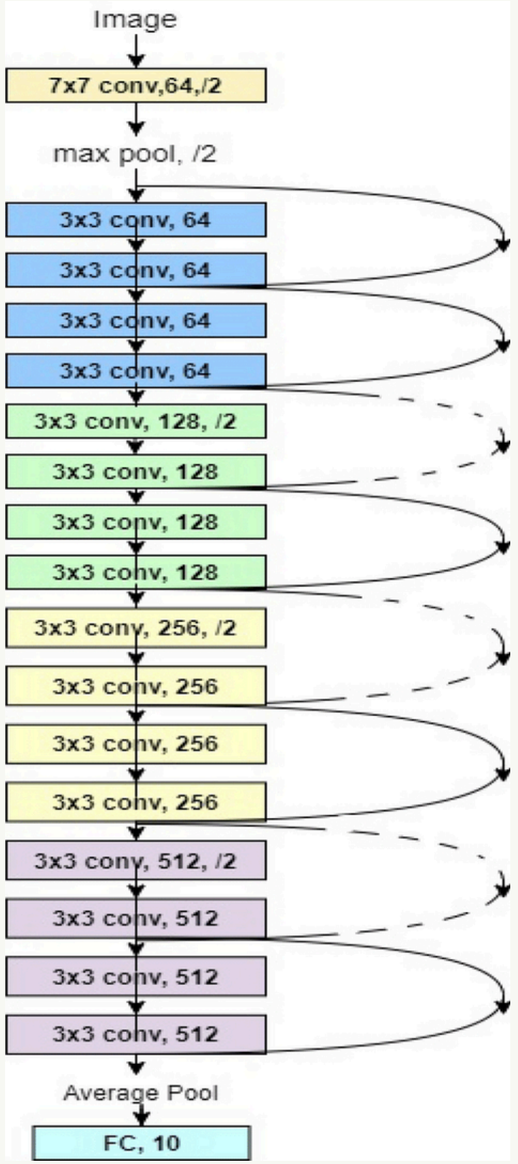
Optimizer

- Goal : it's what truly make the weights update
- Several possibilities :
  - SDG (Stochastic Gradient Descent)
  - Adam
  - AdamW (Adam + Weight Decay)

Transfer Learning

6) Final results : ResNet 18

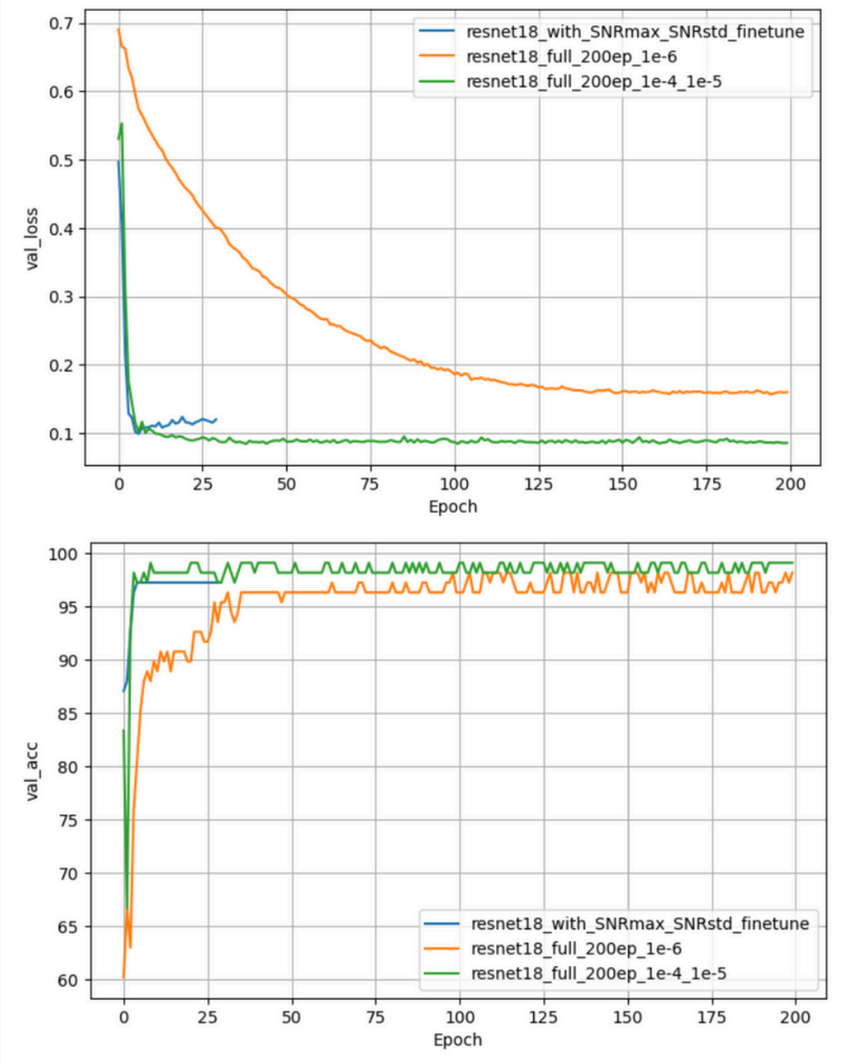
- 18 layers
- 11 millions of parameters
- 4 “residual blocks” :
    - 2 convolutional layers
    - 1 batch normalization layer
    - 1 ReLU
- Revolutionary in Deep Learning :
    - allows to increase dramatically the number of layers without losing accuracy (“vanishing gradient problem”)
    - use of “skip connections” that allows the gradient to take a short cut and not vanishing (that prevents wieght adjustment)



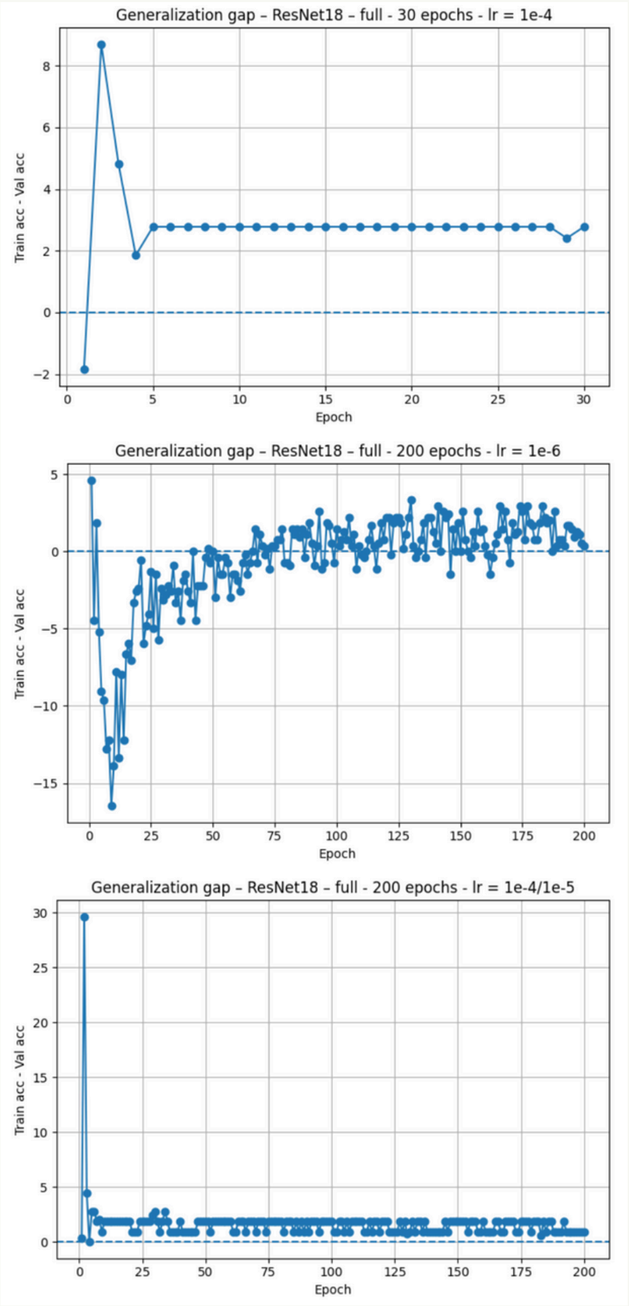
Transfer Learning

6) Final results : ResNet 18

Validation loss / accuracy



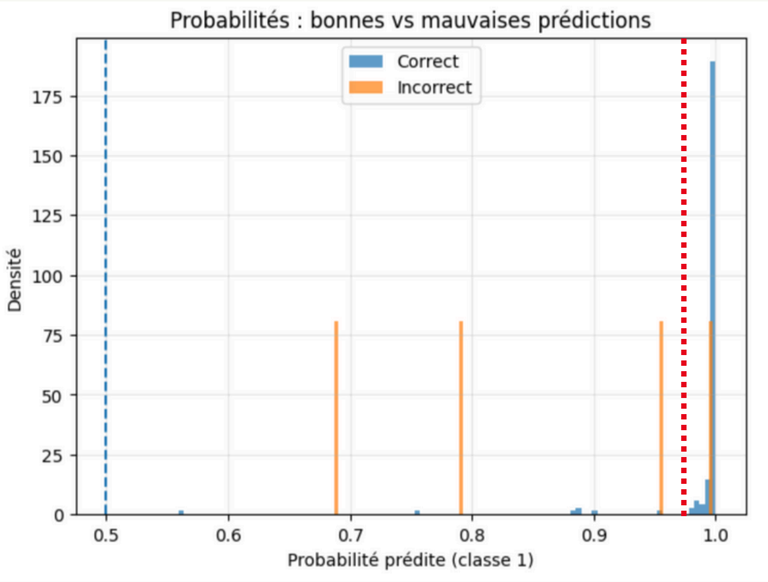
Gap between train and validation accuracy



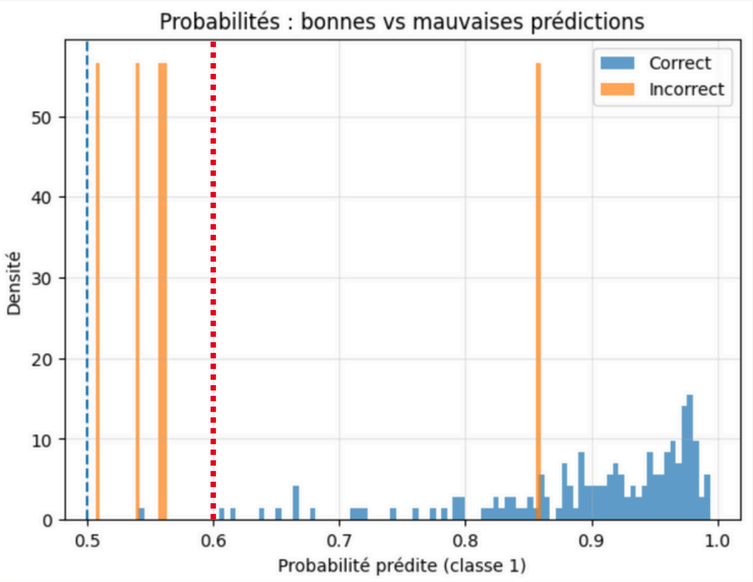
Transfer Learning

6) Final results : ResNet 18

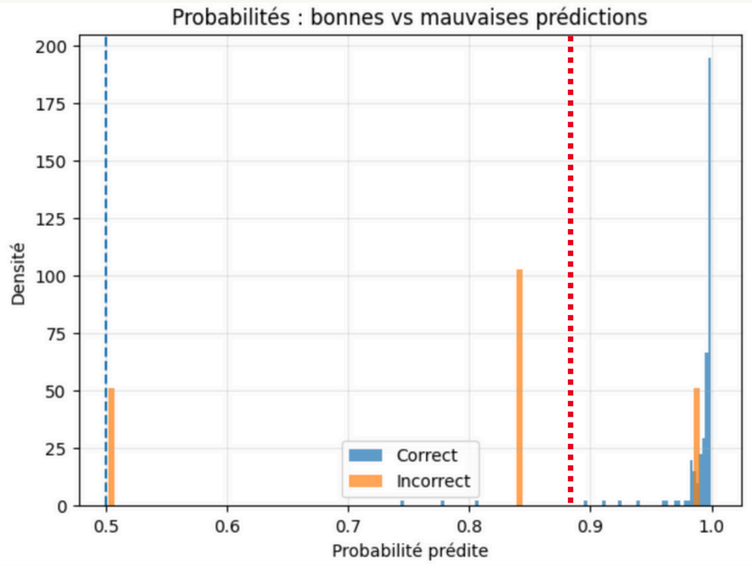
30 epochs - lr = 1e-4



200 epochs - lr = 1e-6



200 epochs - lr = 1e-4/1e-5



Probabilities repartition

What’s next ?

Real case use ?

A function that takes **VHF images as input** (and additional data such as maximum snr, standard deviation…) and that gives **as outputs** :

- a **prediction** (true or false alert)
- the **confidence** on its prediction (probability)
- if the probability is below a certain threshold, indicate that the decision is unclear

More in details, we could use **several models** that have different learning parameters, good performance but different probabilities repartition.
By crossing different sources, we can make sure that we don’t miss some particular cases.

One goal in the remaing weeks is to integrate the trained model at the FSC (French Science Center) in the iFSCtools. On reception of Sub-Image, give the confidence (0-100%) for True or False Alert (or Unsure: if below x% e.g. 80%).

SVOM FR Science Center

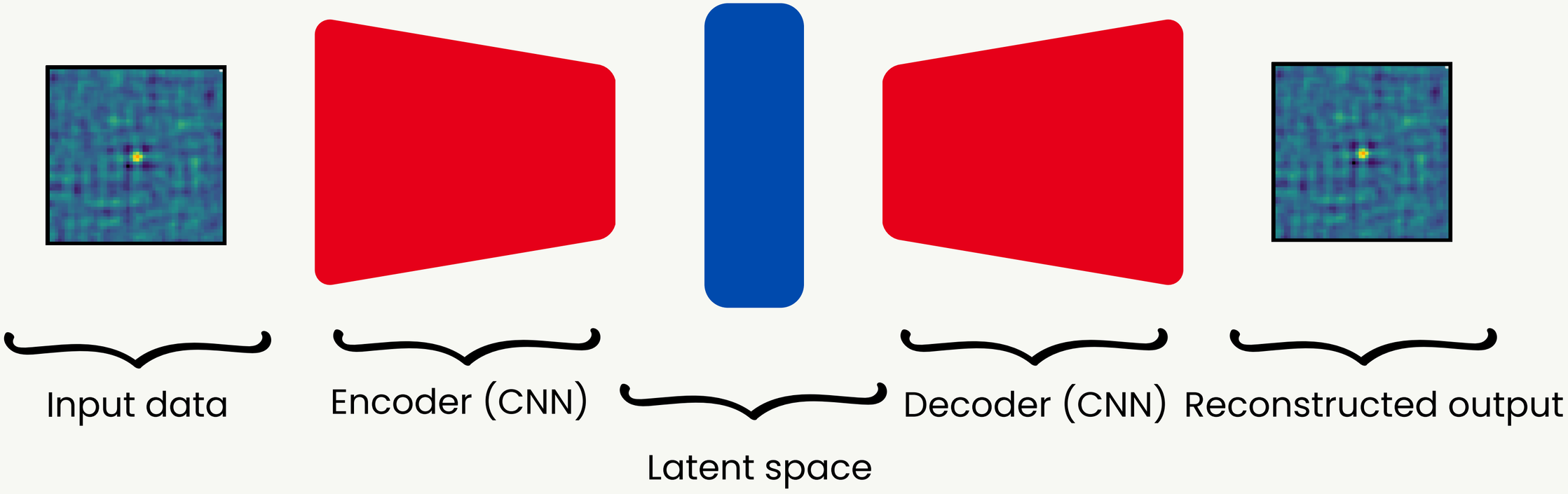
What’s next ?

Training a new model knowing when there is another source ?

Right now the model doesn’t know particularly when another source is on the subimage, it is something that could be intersting to implement, to **tell the model where the other source is located in the subimage.**

What's next ?

Autoencoders ?

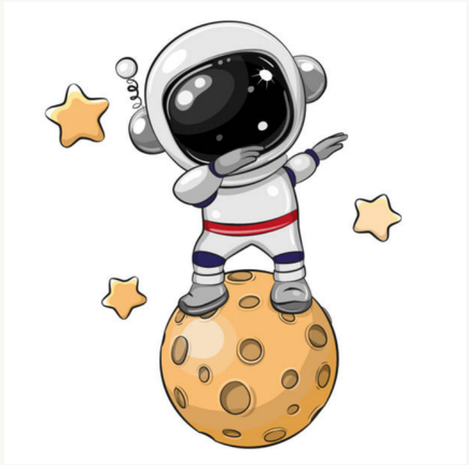


Particular type of neural network for unsupervised learning, very efficient for **anomaly detection**
2 parts :

- The encoder that compress the input data in a latent space
- The decoder that tries to reproduce the input using the data in the latent space

When trained, the decoder poorly reconstruct the input if the input is an anomaly

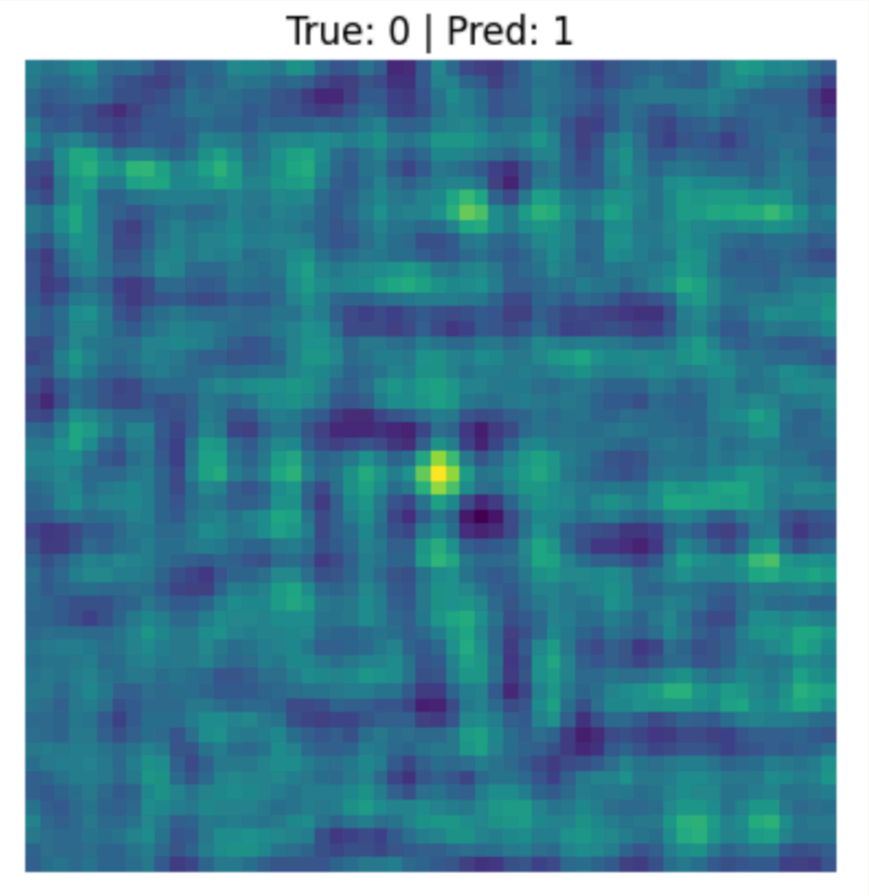
Thanks for listening !



Annexes

https://forge.in2p3.fr/projects/etog-validation/wiki/Basic_TriggerVerdict

Weird error



The verdict (in the Etog-Validation) is : **false alert**
(verification on Mattermost channels, same verdict)

But with the naked eye, it looks like a real trigger.
Considering that the model only have this image as information (with the maximum value of the image and the standard deviation), it can't predict a false trigger.

Annexes

CrossEntropy Loss

Pour une classe vraie $y \in \{0, 1\}$ et prédiction p :

$$\text{Loss} = -[y \log(p) + (1 - y) \log(1 - p)]$$

This loss calculation punishes the error, and takes into account the confidence that the model had in its prediction.

Annexes

Weight Decay

We punish the weights proportionnaly to the square of their value.
That prevents some weights to become too high for no reason, and at the same time prevent overfitting.

Annexes

ReLU activation

Rectified Linear Unit

$$ReLU(x) = \max(0, x)$$

Allows to introduce non-linearities to the model

Annexes

Batch Normalization layer

BatchNorm layer normalize slightly the intermediate activations by using maximum and standard deviation from the batch of data considered
This allows to :

- use bigger learning rates
- introduce a little bit of noise in the data → help to generalize