Group Projects

May 2026

Sébastien Valade & Sophie Giffard-Roisin

VNIVERSIDAD NACIONAL
AVTÓNOMA DE
MEXICO

Objectives

- Apply a complete deep learning project workflow in practice:
  - *Build a training dataset (labeling, data augmentation, etc.)*
  - *Train, evaluate, and improve a model until reaching satisfactory results*
- Group project: 5-6 persons per project
- Evaluation: grading based on the participation during the project, and the quality of the final presentation

Projects

1. **Project 1**: semantic labeling of volcano monitoring cameras at Popocatépetl volcano
2. **Project 2**: detection of volcanic ash in satellite images

Objectives

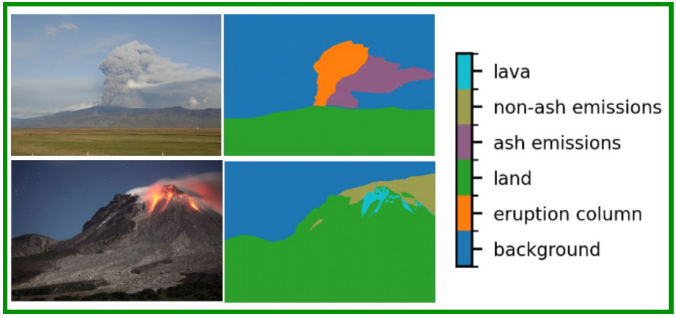
- Apply a complete deep learning project workflow in practice:
  - *Build a training dataset (labeling, data augmentation, etc.)*
  - *Train, evaluate, and improve a model until reaching satisfactory results*
- Group project: 5-6 persons per project
- Evaluation: grading based on the participation during the project, and the quality of the final presentation

Projects

1. **Project 1**: semantic labeling of volcano monitoring cameras at Popocatépetl volcano
2. **Project 2**: detection of volcanic ash in satellite images

Project 1: semantic labeling of volcano monitoring cameras at Popocatépetl volcano

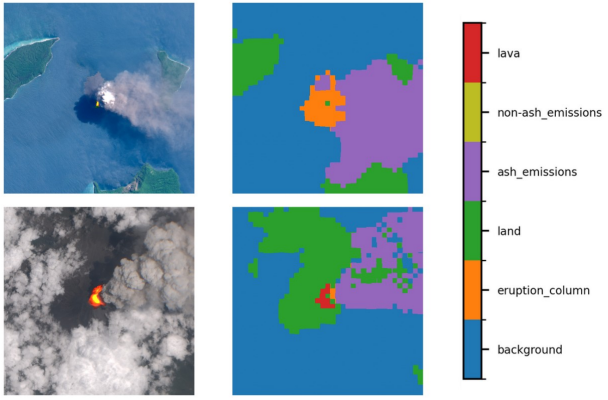
- Dataset: images from the monitoring cameras around Popocatépetl
- Objective: train a model to perform semantic labeling of images
  $\Rightarrow$ pixel-wise classes (ash plume, gas plume, lava, etc.) including day/night images



Giffard-Roisin et al. 2026 *"VIGIA-PlumeNet and VIGIA-PlumeData: Open-Source AI Segmenting Model and Database For Volcanic Plumes and Emissions From Optical Cameras"*

Project 2: detection of volcanic ash in satellite images

- Dataset: Sentinel-2 satellite images of volcanoes worldwide
- Objective: train a model to perform semantic labeling of images
  $\Rightarrow$ pixel-wise classes: ash plume, background (land+clouds)



Session 2

1. **Environment setup**: setup conda environment with trained "VigiaPlumeNet"
   - TASK 1: Create *create conda environment and test model with your image*
     1. Install *Anaconda*
     2. Pull repository vigia-plumenet (or download ZIP), and follow instructions
     3. Create conda environment for CPU (run commands below in terminal from repository folder)

        ```
        $ conda create --name VIGIAPlumeNetCPU python=3.11
        $ conda activate VIGIAPlumeNetCPU
        $ pip install -r requirements_quick_eval_cpu.txt
        ```

     4. Download the trained model (.pth & .pth) saved on HuggingFace, and save it in `trained_models/`
     5. Evaluate on evaluation images, then on your own image

        ```
        python VIGIAPlumeNet_quick_eval.py
        --root_test data_examples/
        --traced_model_path trained_models/VIGIAPlumeNet_traced_Dinov2_s14reg_6classes.pt
        --saving_path data_examples/results/
        ```

Session 2

2. **Data labeling**: label images for your project using "Label Studio"
   - TASK 2: Create *Label Studio* account and join project: link (link shared in class only)
   - TASK 3: Each person should label images from his project with the following classes:

Project 1

1. eruption-cloud (eruptive ash column)
2. ash emissions (dispersed ash plume)
3. non-ash emissions (gas plume)
4. land
5. background (sky+clouds)
6. lava (incandescent flow/blocs)

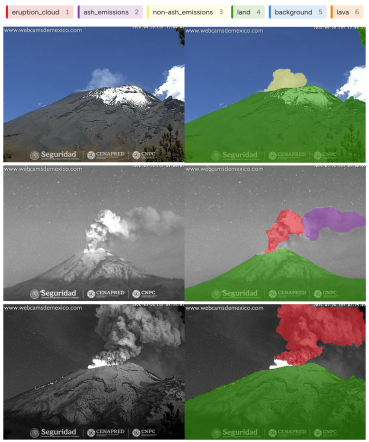
Project 2

1. ash
2. background (land+clouds)

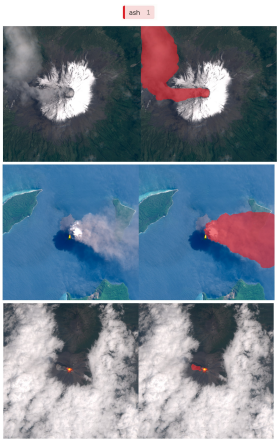
Session 2

2. **Data labeling**: label images for your project using "Label Studio"

Project 1



Project 2



Session 2

2. **Data labeling**: label images for your project using "Label Studio"

Important guidelines:

1. Can pixels be left without labeling?

   ⇒ *yes, leaving unlabeled pixels is okay, ONLY IF they always belong to the same class, i.e.* ***background*** *class*
   - → *project 1 ⇒ don't label the* ***background*** *class (sky+clouds)*
   - → *project 2 ⇒ don't label the* ***background*** *class (land+clouds)*

2. Can classes overlap?

   ⇒ *yes, adjacent classes can overlap, as long as it is kept to a minimum. It is always better to have overlap rather than leaving gaps. (Note: class "priorities" will be defined programmatically to assign a single class to each pixel.)*
   - → *project 1 ⇒ ballistic blocs (which belong to the class "lava") can be small, so it is difficult not to leave gaps with the background land. In this case, label all the volcano slopes as "land", and add a label to the bloc pixels ("lava" class), even if it overlaps with the "land" class.*

3. Can labels be copied from one image to another in *Label Studio*?

   ⇒ *yes, you can copy labels from one image to another:*
   1. *use the "****Move tool (V)****", select the class label you wish to copy (click on the class in the menu "Regions", then on the region class on the image), and hit* ***ctrl+C***
   2. *go to the new image, select the same class label with the "****Move tool****", and hit* ***ctrl+V*** *to paste the labels.*

   - → *project 1 ⇒ land can be copied from one image to another for images from the same webcam*

Session 2

3. **Data search**: find additional images for your project
   - TASK 4: each person should find additional images for his project (5 minimum)
     1. search images, put emphasis on variety (different volcanoes, different conditions, etc.)
     2. download images and place them in folder Drive (link shared in class only)/PROJECTS/Project_X/images/
     3. fill-in the GoogleSheets (link shared in class only), tab corresponding to your project

Project 1
⇒ search images from Popocatépetl (e.g. webcamsdemexico.com/webcams/volcanes/)
⇒ image filename template: <webcamname>_<yyyymmdd>T<hhmmss>.<format>
(e.g. "webcamsofmexico_20220928T115037.jpeg")

Project 2
⇒ search images from platform MOUNTS (Sentinel-2)

Session 3

1. **Review labeled images**: good and bad practice
2. **Export labeled data** for training
   - In "Label Studio" → go to "Project" → click "Export"
   - Select "JSON"
   - Transform the exported JSON file to PNG files
     ⇒ download script "labelstudio_json_to_masks.py" from shared Drive/scripts
     ⇒ run in terminal with the following command:

```
$ python labelstudio_json_to_masks.py
--json_path path/to/exported_json.json
```

3. **Train model**: train model with the exported data

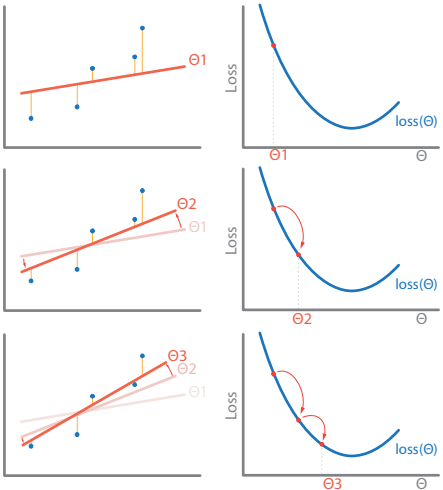
Session 4

Reminders

- Optimization
  $\Rightarrow$ how to find the best parameters Θ for our model?
- Gradient descent
  $\Rightarrow$ how to iteratively update the parameters Θ to minimize the loss function?
- Confusion matrix
  $\Rightarrow$ how to evaluate the performance of a classifier at a fixed threshold?

Optimization

We start with a random guess for our parameters Θ



We will iteratively look for the best position of our line, by varying its parameters (Θ).

But how can we efficiently vary our parameters (Θ)?

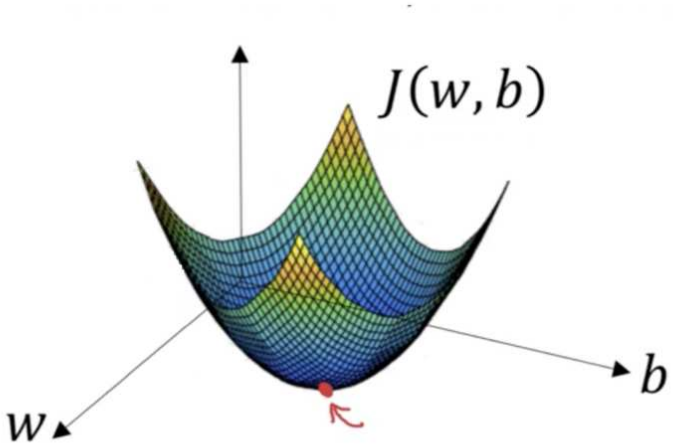
Note : Loss functions could be :

$$\text{RMSE}(X, h_\theta) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left[h_\theta(X^{(i)}) - Y^{(i)}\right]^2}$$

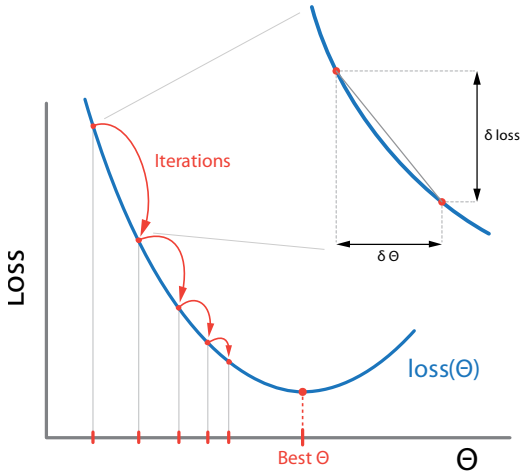
$$\text{MSE}(X, h_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[h_\theta(X^{(i)}) - Y^{(i)}\right]^2$$

Optimization

Of course, in practice, Θ represents more than one parameter, so for example with 2 parameters you should imagine a surface...



Gradient descent



By changing Θ from δΘ
We improve loss(Θ) of δloss

The gradient is the slope we will follow to minimize our loss function.

$$\text{gradient} = \frac{\delta \text{loss}}{\delta \theta}$$

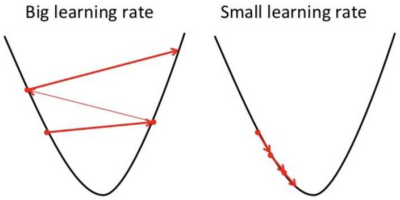
One iterative solution is : $\theta \leftarrow \theta - \eta \cdot \frac{\delta \text{loss}}{\delta \theta}$

where η is the learning rate

This process is called gradient descent and the function used to optimize the descent, optimization function

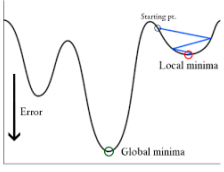
Gradient descent

Importance of the learning rate:



https://builtin.com

- A too large learning rate will never reach the minimum
- A too small learning rate can require too much steps (computation time), and can also fall in a local minima



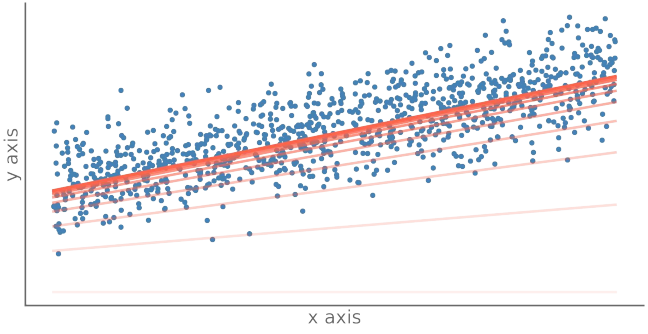
Gradient descent

$$MSE(X, h_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[h_\theta(X^{(i)}) - Y^{(i)}\right]^2$$

$$\nabla_\theta MSE(\Theta) = \begin{bmatrix} \frac{\partial}{\partial\theta_0} MSE(\Theta) \\ \frac{\partial}{\partial\theta_1} MSE(\Theta) \\ \vdots \\ \frac{\partial}{\partial\theta_m} MSE(\Theta) \end{bmatrix} = \frac{2}{n} X^T \cdot (X \cdot \Theta - Y)$$

Iterative solution is : $\Theta \leftarrow \Theta - \eta \cdot \nabla_\theta MSE(\Theta)$
where η is the learning rate

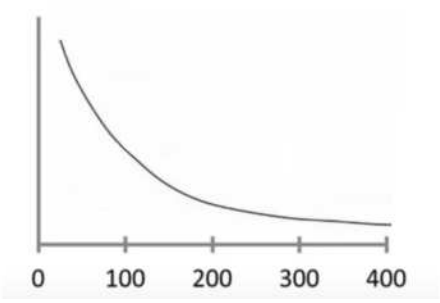
n : number of observations
m : number of characteristics

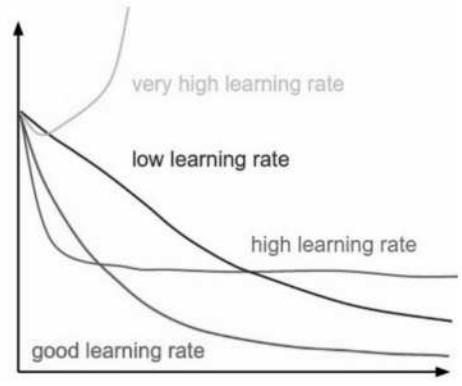


#i	Loss	Gradient		Theta	
0	+12.481	-6.777	-1.732	-3.388	+0.000
20	+4.653	-4.066	-1.039	-2.033	+0.346
40	+1.835	-2.440	-0.624	-1.220	+0.554
60	+0.821	-1.464	-0.374	-0.732	+0.679
80	+0.455	-0.878	-0.224	-0.439	+0.754
100	+0.324	-0.527	-0.135	-0.263	+0.799
120	+0.277	-0.316	-0.081	-0.158	+0.826
140	+0.260	-0.190	-0.048	-0.095	+0.842
160	+0.253	-0.114	-0.029	-0.057	+0.851
180	+0.251	-0.068	-0.017	-0.034	+0.857
200	+0.250	-0.041	-0.010	-0.020	+0.861

Gradient descent

We can plot the value of the loss with respect to the iterations

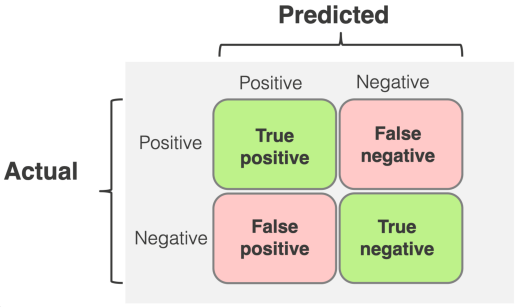


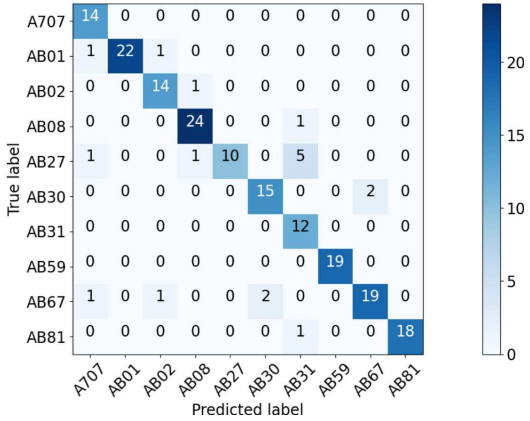


https://builtin.com

Confusion matrix

- Summary of the performance of a classifier *at a fixed threshold*
- Allows to see where are the mis-classifications
- Is useful for binary and multi-label classification problems





The classifier is perfect if everything outside of the diagonal is 0

Session 4

- **Review trained models**
  1. Download data set (images+labels) from shared repository
     ⇒ *download CV4GS_2026/PROJECTS/Proyecto_<X>/data/database_<yyyymmdd>*
  2. Download new model from shared repository
     ⇒ *download CV4GS_2026/PROJECTS/Proyecto_<X>/results/database_<yyyymmdd>*
     ⇒ *includes:*
     `.pth file`: model weights
     `.png files`: model training curves:
     → **loss**: should decrease
     → **IoU**: should increase
     → **accuracy**: should increase
     `.txt files`: model training curves:
     → **epoch metrics**: used to create .png files
     → **hyper-parameters**: store learning rate, batch size, etc.
  3. Evaluate model on validation dataset (example for project 2)
     ⇒ *segment images from validation dataset, and save results as .png + .npy*

```
$ conda activate VIGIAPlumeNetCPU
$ python VIGIA-PlumeNet/VIGIAPlumeNet_quick_eval.py
--traced_model_path Proyecto2/results/database_20260512/finetuning1/VIGIAPlumeNet_best_model_traced.pth
--root_test Proyecto2/data/database_20260512/valid_set
--saving_path Proyecto2/results/database_20260512/finetuning1/results_valid_set/
```

Session 4

- **Review trained models** (continued)
  4. Compute metrics on validation dataset (confusion matrix, etc.)

     ⇒ *Download ‘‘evaluating_models’’ folder from shared Drive: CV4GS_2026/PROJECTS/scripts/evaluating_models/*

     ⇒ *Edit paths in the script ‘‘segmentation_multilabel_metrics_calculate.py’’:*
     1. *where the validation results .npy were saved*
     2. *where the ground truth labels .png are located*

     ⇒ *Run script (and install missing libraries):*

```
$ conda activate VIGIAPlumeNetCPU
$ pip install scikit-learn
$ pip install natsort
$ python segmentation_multilabel_metrics_calculate.py
```

     → *creates .png: confusion matrix, precision, etc.*

Session 4

- **Improve model**
  - hyper-parameters: change model hyper-parameters, and re-train
    - Example Project 1
      $\Rightarrow$ *"finetuning 1" used batch size = 6*
      $\Rightarrow$ *"finetuning 2" used batch size = 8 $\Rightarrow$ enough to improve by 10% ash class (see confusion matrix)*
  - training data: improve labeling, increase number of labeled images, and re-train

Session 5

$\Rightarrow$ labeled dataset has been improved: *more labeled images + labels reviewed/homogenized*

$\Rightarrow$ models have been re-trained: *new architectures and hyper-parameters have been tested*

$\Rightarrow$ results have been improved: *let's review each project!*

Session 5

Reminders

- Train vs. Test vs. Validation datasets
  - **training set**: *used to train the model (i.e. to optimize the parameters)*
  - **validation set**: *used to evaluate model performance during training*
  - **test set**: *used to assess the final model performance on unseen data*

  $\Rightarrow$ confusion matrix can be computed on either validation or test set

Project 1

Training 1

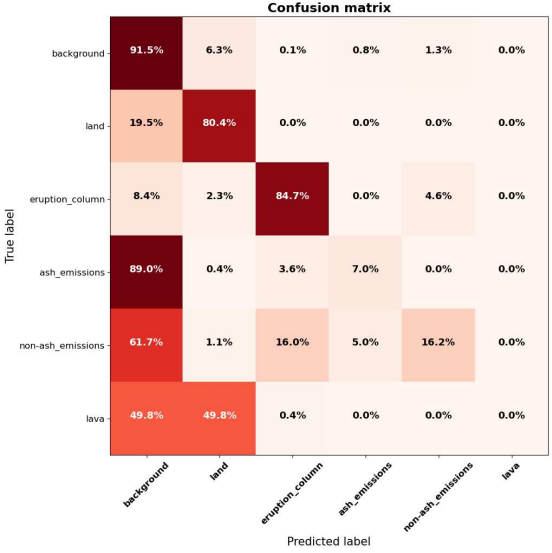
- Model architecture = VIGIA-PlumeNet
  - backbone = *Dino v2*
  - head = *1 Linear (FC) + 3 Upsamples + 1 Conv2d + output n=6*
- Hyper-parameters:
  - learning rate = *1e-05*
  - batch size = *6*
  - number of epochs = *10*
  - loss function = *weighted cross-entropy*
- Training dataset:
  - size = *83 (train 57 + validation 26)*
  - label quality = *poor*

Training 2

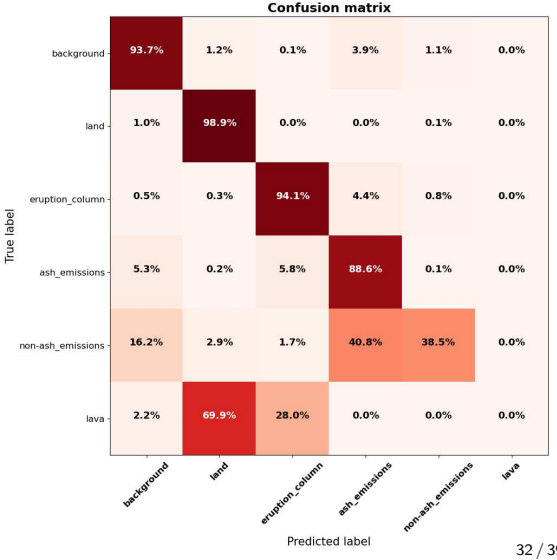
- Model architecture: unchanged
- Hyper-parameters: unchanged
- Training dataset:
  - size = 160 (train 84 + validation 37 + test 39)
  - label quality = satisfactory

Project 1

Training 1

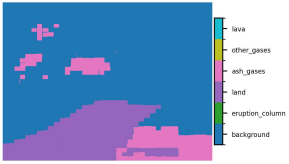


Training 2

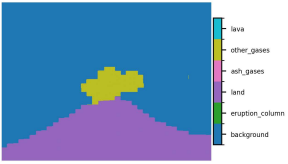


Project 1

Training 1



Training 2



Project 2

Training 1 (finetuning 2)

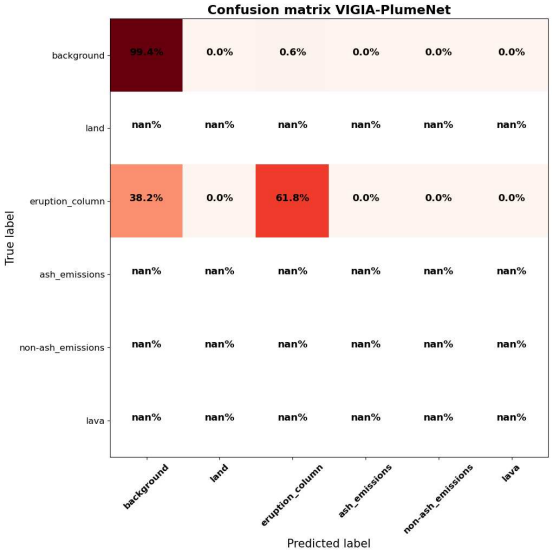
- Model architecture = VIGIA-PlumeNet
  - backbone = *Dino v2*
  - head = *1 FC + 1 upscale + 1 CONV + softmax n=6*
- Hyper-parameters:
  - learning rate = *1e-05*
  - batch size = *8*
  - number of epochs = *10*
  - loss function = *weighted cross-entropy*
- Training dataset:
  - size = *45 (train 31 + validation 14)*
  - label quality = poor

Training 2

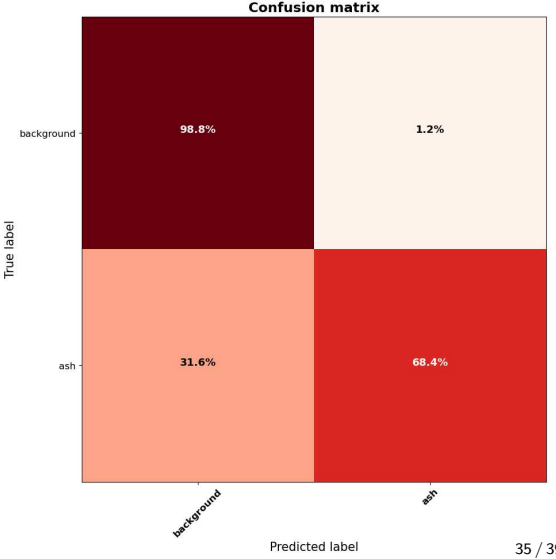
- Model architecture: unchanged
- Hyper-parameters: unchanged
- Training dataset:
  - size = 108 (train 80 + validation 28)
  - label quality = satisfactory

Project 2

Training 1

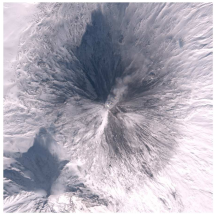


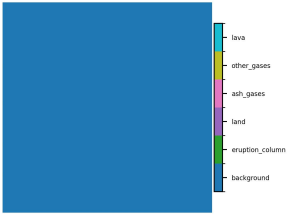
Training 2



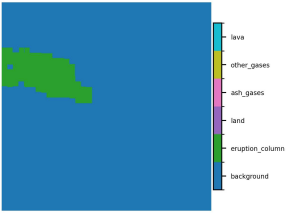
Project 2

Training 1





Training 2



Project 2 ⇒ test new architectures and train from scratch

Training 3

- Model architecture:
  - backbone = *Dino v2*
  - head = *1 FC + 1 upsample + 1 CONV + softmax n=2*
- Hyper-parameters:
  - learning rate = *1e-05*
  - batch size = *8*
  - number of epochs = *30*
  - loss function = *weighted cross-entropy*
- Training dataset:
  - size = *108 (train 80 + validation 28)*
  - label quality = *satisfactory*

Training 4

- Model architecture:
  - backbone = *Dino v2*
  - head = *1 FC + 3 ConvTranspose2d + 1 upsample + softmax n=2*
- Hyper-parameters:
  - learning rate = *1e-05*
  - batch size = *8*
  - number of epochs = *30*
  - loss function = *weighted cross-entropy*
- Training dataset:
  - size = *108 (train 80 + validation 28)*
  - label quality = *satisfactory*

```python
class DINO2SEG_small_reg(nn.Module):

        def __init__(self, num_cls):
                super(DINO2SEG_small_reg, self).__init__()
                self.backbone = torch.hub.load('facebookresearch/dinov2', 'dinov2_vits14_reg')
                self.num_class = num_cls

                switch = False
                for name, param in self.backbone.named_parameters():
                        if param.requires_grad:
                                if 'blocks.4.' in name:
                                        switch = True
                                if switch:
                                        param.requires_grad = True
                                else:
                                        param.requires_grad = False
                # token projection
                # linear
                self.l1 = nn.Linear(384, 512)
                self.relu = nn.ReLU()
                # upsample
                self.up1 = nn.Upsample(scale_factor=2)
                self.up2 = nn.Upsample(scale_factor=2)
                self.up3 = nn.Upsample((518, 518))
                self.out = nn.Conv2d(512, num_cls, 3, 1, padding='same', padding_mode='replicate')
```

```python
class DINO2SEG_small_reg_upconv(nn.Module):

        def __init__(self, num_cls):
                super(DINO2SEG_small_reg_upconv, self).__init__()
                self.backbone = torch.hub.load('facebookresearch/dinov2', 'dinov2_vits14_reg')
                self.num_class = num_cls

                switch = False
                for name, param in self.backbone.named_parameters():
                        if param.requires_grad:
                                if 'blocks.4.' in name:
                                        switch = True
                                if switch:
                                        param.requires_grad = True
                                else:
                                        param.requires_grad = False

                self.l1 = nn.Linear(384, 512)
                self.relu = nn.ReLU()
                self.up1 = nn.ConvTranspose2d(512, 256, 4, 2, 1)
                self.up2 = nn.ConvTranspose2d(256, 128, 4, 2, 1)
                self.up3 = nn.ConvTranspose2d(128, num_cls, 4, 2, 1)
                self.out = nn.Upsample((518,518))
```

Oral presentation

Present your project (20 minutes presentation max):

- Goal: what is the problem you are trying to solve?
- Data: what data are you using? How did you label it? What are the challenges?
- Model training: limits of the original Vigia-PlumeNet for this task? What did you change, why?
- Results: present the results you obtained (original model vs. best model): confusion matrices on database 2026-05-21, and example of segmented images
- Discussion: what can you suggest as next steps?