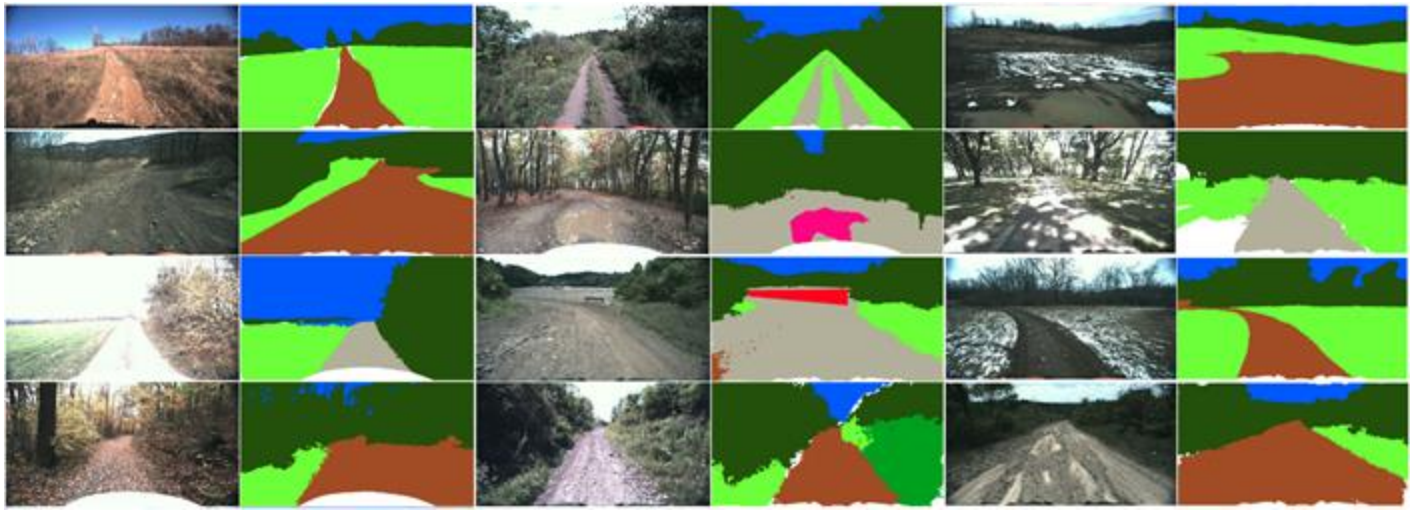
Machine Learning Project work: “Autoencoder based onboard image segmentation”

Mario Vento and Diego Gragnaniello

The task to address

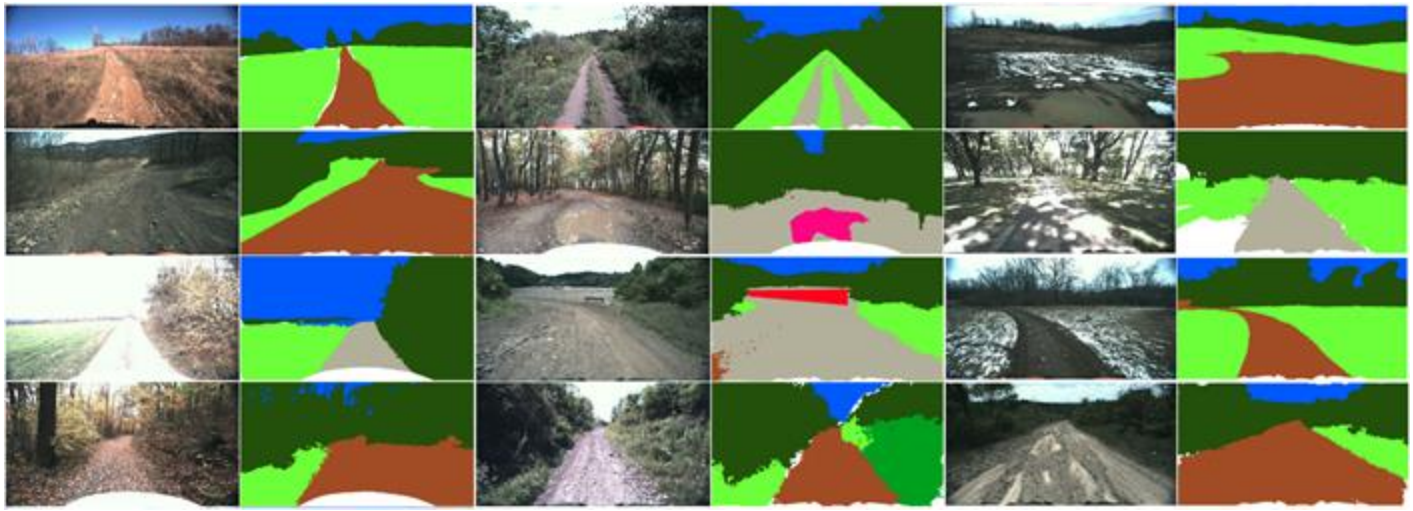
- The aim of this project is to segment images acquired onboard of cars in a rural environment
  - Moving camera
  - Still frame have been extracted
  - Classify each image pixel as one of 8 classes:
    sky, rough trail, smooth trail, traversable grass, high vegetation, non-traversable low vegetation, puddle, and obstacle



The task to address

- Images are acquired
  - by the very same camera
  - during daylight with different illumination conditions

- You **cannot** extend the training set
- You ***can*** decide the split

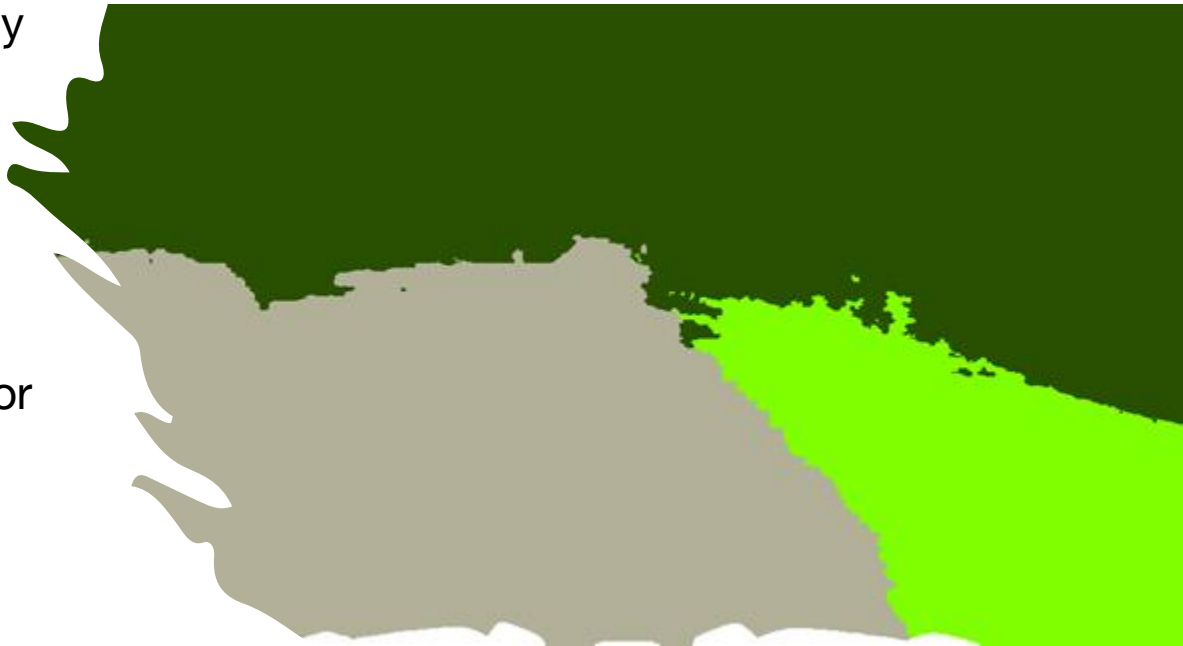


The dataset

- Log with the UNISA account and download the **training dataset**:
  - https://drive.google.com/file/d/1PabQabNg1WT8WrDHWyesEetxXZq3Umb0/view?usp=sharing

- **931** samples:
  - Each image may or may not contain all classes
  - You can choose your train/val split

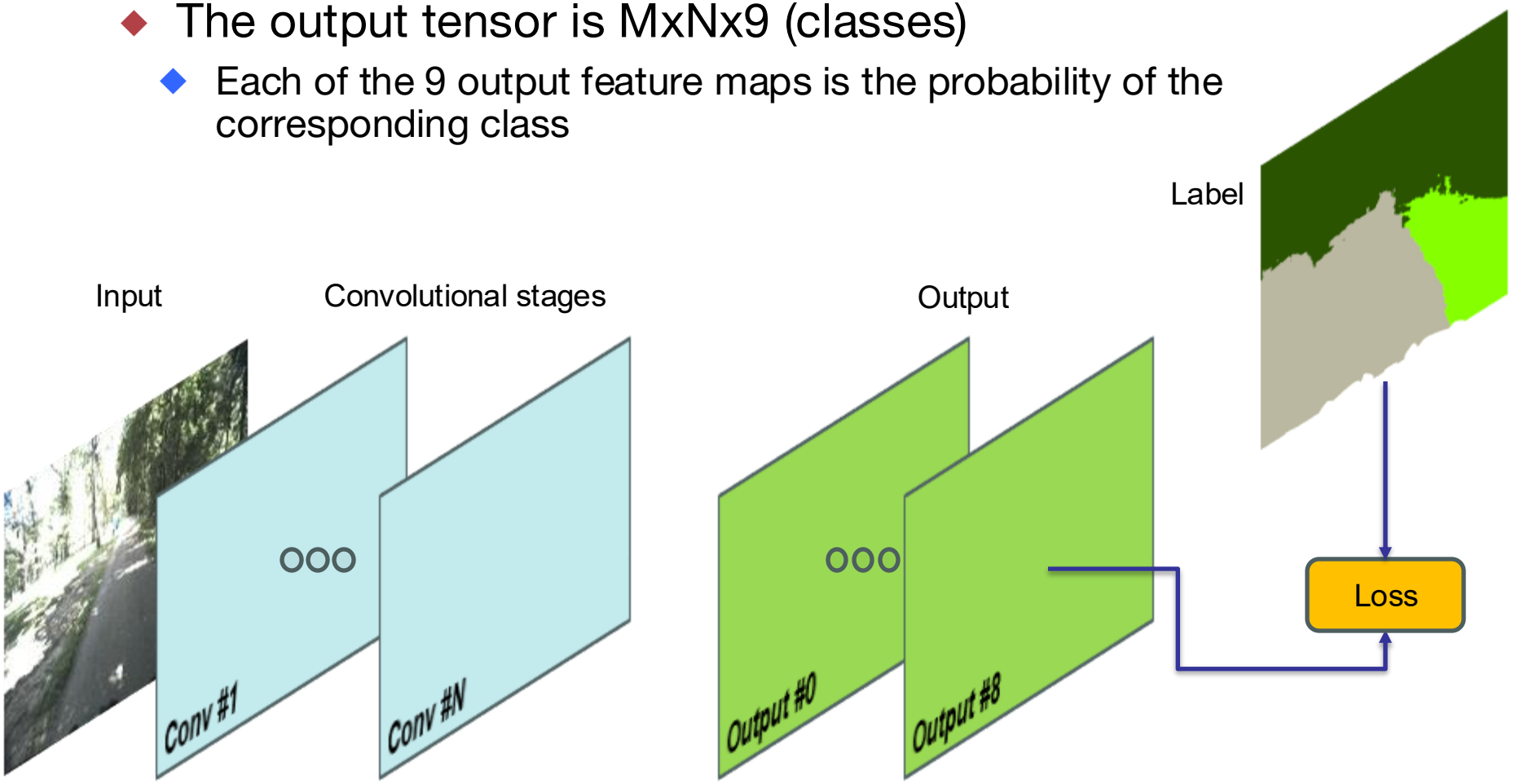
- Labels range from 0 (background) to 8
  - You can open the file for a color coded visualization



Example of neural network architecture

CNN for pixel classification:

- The input image is MxNx3 (color channels)
- The output tensor is MxNx9 (classes)
  - Each of the 9 output feature maps is the probability of the corresponding class



IMPORTANT: Model training and validation is not a feed-forward process

- Alternate trainings and validations:
  - After each validation, try to understand which samples are misclassified and why
  - Change your model and/or training set to improve the performance

IMPORTANT: Model training and validation is not a feed-forward process

- Alternate trainings and validations:
  - If the performance seems very good, be sure that the validation set is challenging enough
  - Keep track of your countermeasures/improvements for the final project presentation

What you can use

- You **cannot** extend the provided training set

- You ***can*** decide the train/val split
- You ***can*** use data augmentation techniques

- Any algorithm (whatever kind of classifier, including non-neural ones, preprocessing, training strategy, validation), **BUT**:
  - You must be able to explain what you have used

  - It must be **runnable**
    - inside **Google Colab;**
    - in **test** using **less than 4 GB GPU RAM;**
    - in **training** using **less than 5 GB GPU RAM;**

Model evaluation (1 of 2)

Performance will be evaluated on a **private test set** measuring the per-class mean Intersection-over-Union (IoU):

- For each class we compute the binary masks of the ground-truth labels and the network prediction
- The metric is the ratio between:
  - The intersection
    (logical *and* of the two binary masks)
  - The union
    (logical *or* of the two masks)

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- We average IoU for all the images

Model evaluation (2 of 2)

In the evaluation we will consider the resources needed by the model and its computational complexity

You have to select the best trade-off among:

- IoU performance, larger is better
- GPU memory required (during test), less is better
- Processing frame rate (during test), i.e., how fast the method can process the input sample using a certain GPU, larger is better

Share the load

- Each member of the team will be requested to submit an estimate of the individual effort contributed by all members
  - To prevent "free riders"
  - Submissions will be "blind" (each member will not see the submissions of other members)

What you must submit

1. A link to a Google Drive folder containing:
   - The code used for training your system, in the form of a Google Colab Notebook
   - The additional training/validation data used (including labels) generated with off-line augmentation
   - The train/validation split protocol
   - The files containing the saved models/weights after the training
   - The code needed to test your system (test script; see next slide)

What you must submit

The test script must be a Google Colab Notebook containing:

- The code to load your trained model
- A "**predict**" function with the following specification:
  - Prototype: *predict(X)*
  - where: *X* is the tensor containing a batch of test samples; the shape of *X* is*:* (batch_size, rows, cols, 3); the type is uint8
  - return value: the tensor with the prediction on the batch; the shape of the return value is: (batch_size, rows, cols, 1); the type is uint8, just like the label files format
  - [*batch_size* is the number of samples in the batch, and is not a fixed value; the function must work independently of this value]
- The *predict* function will not see the whole test set at once
- The *predict* function must perform all the required preprocessing on the batch of test data, and the postprocessing on the results before returning them

What you must submit

2. You will also need to make a 8-minutes presentation and a report (both in English) containing:
    1. The rationale to collect the dataset.
    2. How the dataset split into training and validation sets was done.
    3. The pre-processing pipeline adopted.
    4. The network architecture selected.
    5. The training hyperparameter setting, including the loss function.

IMPORTANT: Don’t forget to

- Write the **names of all the team members** in the Google Drive folder (in a text file)
- Ensure that the **link** you submit is **readable to anyone** (no authorization must be requested)
- Make sure that the **test script** is **compliant with the specification** (if you have doubts about the specification, **ask**)