HOMEWORK 2, ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING
INNOCENTE – MASCIULLI

We began our work from the notebooks seen during the laboratory session, and used it as a base to develop our own network. To avoid the bias on the test images set, we tried to submit only the solutions that were significatively improving the results or that required some checks.
The main issues were caused by the high and different resolution of each team dataset, along with the unbalancing of the classes: in each sample the background, crop and weed classes were distributed in a non uniform way.
We started our work with a simple encoder-decoder network, as seen during the laboratory. It was composed by VGG16 as a feature extractor and, for the decoder, six upsampling layers with Upsampling2D, Conv2D (256 starting filters, halved each level) and ReLU activation function. The last layer was a Conv2D layer with a number of filters equal to the different classes (three, in our case). This net was trained on the BipBip Haricot dataset, resizing the input images and masks to 256x256 and then feeding it for training. Also prediction was executed in this way. Starting from this point, we have modified some aspects: we changed our model, trying both a custom UNet model with skip connection and another one with a VGG encoder, adding concatenate layers in the decoder levels. In parallel, we have experimented with some custom loss functions in order to weighting the class influence during training. We have implemented a loss function that computes the IoU over each class, weighting these values and multiplying it with Keras SparseCategoricalCrossentropy.
As long as we improved our results, we enlarged the sets of images we used for training, moving from Bipbip-Haricot to the whole Haricot and then to the whole dataset.
To face the issue of having few training images, we tried to implement k-fold in our model training, but the high computation time was hardly compatible with colab and the deadline of the homework, so we discarded this option after a couple of days.
In order to preserve the image resolution, and all the information contained, we have applied a tiling over each data sample, capturing patches of the images with lower resolution (256x256 first, 512x512 then) and feeding each patch as a new data sample. With this purpose we've used the tf.image.extract_patches function. Each patch was not overlapping with the others, the padding was setted as "SAME" in order to cover the entire original data sample. The same procedure was used to make the predictions on the test set, feeding each patch and reconstructing the original image, deleting the padding borders. Since this method required a lot of computation power due to the size of the images we have never been able to train this method on the entire dataset, but only on some subsets.
In the last two days, we worked in parallel also with a library for the Image Segmentation found on GitHub[1], to verify the efficiency of our model and to test how far we could go with the test set.
The main noteworthy results we have been able to achieve are: 0.68 using UNet and vgg16 encoder, image resized to 512x512, training on all the teams and plants with augmentation and a weighted loss function; 0.61 with UNet and a vgg16 encoder, and 512x512 tiling on a augmented subset of the images and a weighted loss function; 0.67 with the segmentation_models library. The first two are the ones presented in the two allegated notebooks.

Federico Innocente
Francesco Masciulli

[1] segmentation_models: https://github.com/qubvel/segmentation_models