Artificial Neural Networks and Deep Learning 2020

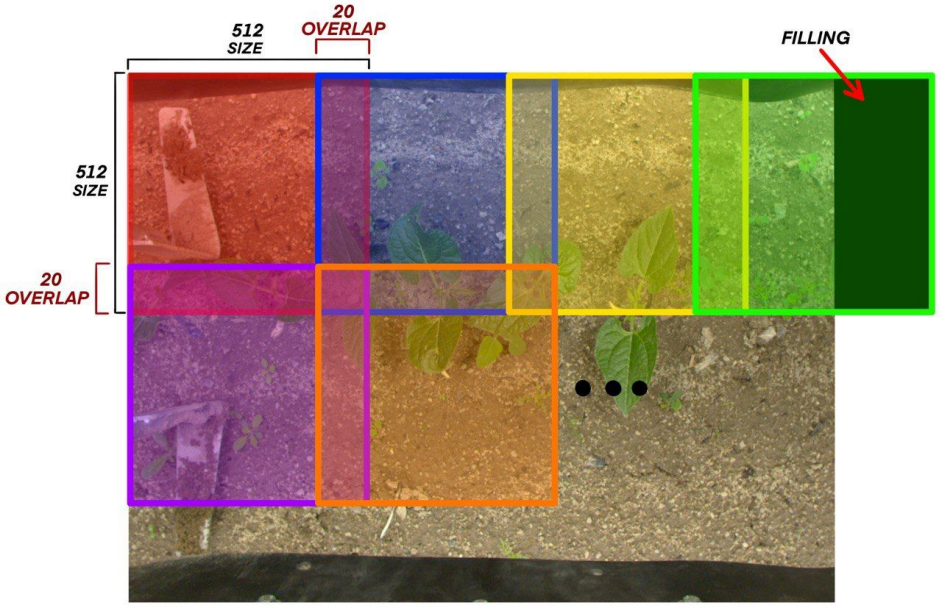
Homework 2 - Image Segmentation

Team: StavoCorLibbanese

Amorosini Francesco - Awad Yasmin - Fioravanti Tommaso

For this challenge we decided to use the BipBip Haricot dataset exclusively. As Neural Networks suitable for solving the proposed challenge, two structures have been created.

The first approach we present is an encoder-decoder handmade net with *depth = 5*. The number of filters per convolutional layer doubles as we get deeper in the network, starting at a value of 8. Given the considerable size of the images in the dataset, to solve the problem of loss of resolution within the network, we decided to divide the images into tiles. In order to crop images, we applied a handmade algorithm *save_crops* that takes as input the path of the image to be tiled, the directory in which to save it, a string that tells if the image is from the Images or from the Masks part of the dataset, the size of the square crop and the overlap between crops. Through this algorithm, the image is divided into sub-images given by a division similar to the one in the following figure.



In order to fulfill the size of the crops decided, in each crop of the image we filled the missing parts with a filling color. Both for images of plantation and their masks we applied black as a filling color in order to let the net know that that part is part of the background. We also tried to use, for the first ones, as a filling the RGB color (148, 124, 92), which was determined to be one of the dominant brown colors of the image, but the results were worse. As size dimension for the crop we opted for 512 pixel width and height, as overlap dimension we opted for 20 pixels. We introduced the overlap in order to ease the process of classifying leaves that happened to be placed across two tiles.

Once the network is trained, in order to analyze the images in the test set, we apply the same size tiling to them, with overlap equal to zero. Each crop of the image is fed to the network which returns the corresponding predicted mask. Then we apply to them the *crops_reconstruct algorithm* to obtain the final mask for the entire image. Finally we used the *make_equal* algorithm in order to ensure that each pixel has one of the colors of the 3 classes considered.

During our experiments we discovered that increasing the depth of the model would worsen the performances, as each additional MaxPooling layer would lose too much information.

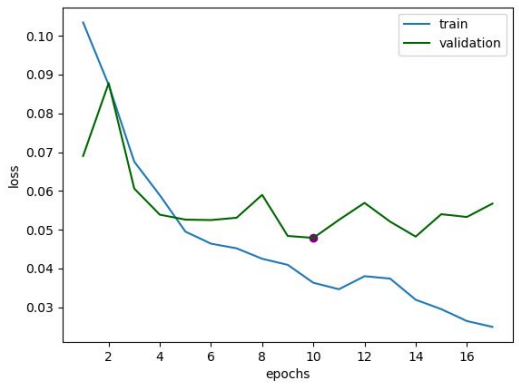
With this net we obtained, with an Early Stopping of 10, the best val_loss at the 30th epoch obtaining a test result of 59.14%, and the best val_meaniou at the 40th epoch obtaining a test result of 61.57%.

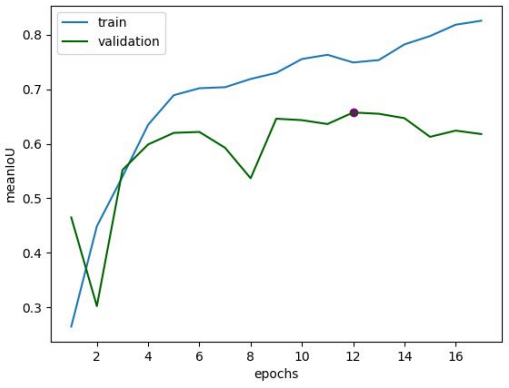
The second approach used is always an encoder-decoder network, but different from the first one, we have used a pretrained model (VGG-16) as encoder. We have tried to apply fine-tuning, changing the number of trainable layers of the VGG, but we have achieved the best results using a simple Transfer Learning (so without retrain the VGG's layers). For the decoder part, we have stacked UpSampling and Convolutional layers. As activation function we have used the *softmax* since we have to predict 3 classes. For what concerns the loss and the optimizer, we have used the *SparseCategoricalCrossentropy* since we have integers and not one-hot encoded labels and the Adam optimizer with a learning rate of `1e-4`.For what concern the evaluation metrics, we have used the accuracy but mostly the MeanIoU.

With this net we obtained, with an Early Stopping of 7, the best val_loss at the 10th epoch obtaining a test result of 66.30%, and the best val_meaniou at the 12nd epoch obtaining a test result of 67.36%.

We tried also to use a learning rate scheduler technique, using an *exponential decay* approach to see if we could gain in the performance, reducing exponentially the learning rate at different epochs, but we didn't find so much in this sense.

Below we reported the model summary of the VGG approach.





Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 16, 16, 512)	14714688
up_sampling2d_5 (UpSampling2	(None, 32, 32, 512)	0
conv2d_6 (Conv2D)	(None, 32, 32, 256)	1179904
re_lu_5 (ReLU)	(None, 32, 32, 256)	0
up_sampling2d_6 (UpSampling2	(None, 64, 64, 256)	0
conv2d_7 (Conv2D)	(None, 64, 64, 128)	295040
re_lu_6 (ReLU)	(None, 64, 64, 128)	0
up_sampling2d_7 (UpSampling2	(None, 128, 128, 128)	0
conv2d_8 (Conv2D)	(None, 128, 128, 64)	73792
re_lu_7 (ReLU)	(None, 128, 128, 64)	0
up_sampling2d_8 (UpSampling2	(None, 256, 256, 64)	0
conv2d_9 (Conv2D)	(None, 256, 256, 32)	18464
re_lu_8 (ReLU)	(None, 256, 256, 32)	0
up_sampling2d_9 (UpSampling2	(None, 512, 512, 32)	0
conv2d_10 (Conv2D)	(None, 512, 512, 16)	4624
re_lu_9 (ReLU)	(None, 512, 512, 16)	0
conv2d_11 (Conv2D)	(None, 512, 512, 3)	51

Total params: 16,286,563
Trainable params: 1,571,875
Non-trainable params: 14,714,688