ANNDL Homework 2 Report

Introduction

In the following document we are going to describe how we approached the segmentation problem given as an assignment. Our goal was to implement a model able to detect the class and location of objects in a set of given images. In particular, there were three classes: crop, weeds and background.

We decided to work on the entire dataset, instead of focusing on a specific one.
In the first section of the notebook we organized and loaded the dataset from disk. All the names of the files are merged together.
Then we apply data augmentation through the method getItem of the object CustomDataset.

We started from the notebooks shown in the lab sessions, and then rewrote some parts and tweaked most of the hyperparameters and variables. We eventually reached an average meanIoU of 0.62 on the whole dataset.

Computational Limits

To facilitate collaboration we used Google Colab, though it imposed some hard limitations on the computational capabilities:

- we were not able to set the batch size to a value higher than 1;
- we did not perform fine tuning on transfer-learned models;
- we couldn't set an image size greater than 2048x2048;
- we couldn't use too many filters on convolutional layers;

All of these are factors which might have affected the performance of our model.

Image processing

A major change that significantly improved our performance was to switch the library we used to open and process images from PIL to CV2. As a matter of fact, it seemed that due to the dimension of the images of the dataset, too much information was lost when processing the images with the first library.

Since images had very large dimensions, using a resize function was another of the reasons of the significant loss of information. Through trial and error, we found that 2048 x 2048 was the best image shape for the model's input, with the application of CV2.INTER_AREA as an interpolation method for downscaling and CV2.BICUBIC for upscaling.

Another important result in performances has been achieved by applying the same random transformation to image and mask target in custom dataset objects.

Building the model

To perform image segmentation, we tried different approaches to build our encoder and decoder. In the next sections, the most significant attempts are described.

U-Net

The first approach we tried was a traditional U-Net which initially consisted of a stack of Convolutional and Max Pooling layers for the encoder part and Upsampling and Convolutional layers for the decoder. To get better precise locations, at every step of the decoder we used skip connections by concatenating the output of the upsampling layers with the feature maps from the encoder at the same level (*u6=u6+c4; u7=u7+c3; u8 =u8+c2; u9=u9+c1*) and after every concatenation we again applied two consecutive regular convolutions so that the model can learn to assemble a more precise output.

We then tried changing the structure of the model several times, by adding batch normalization, and changing layers (for instance from Upsampling to Transpose Convolution in the decoder), and some regularization techniques such as Dropout and using the Leaky ReLU as activation function and noticed some improvements on the performance, reaching a meanIoU of 0.22 on the test dataset. In parallel, we decided to try an implementation with transfer learning.

Transfer Learning on Encoder

By leveraging previously trained models and including them in the structure of our new model we were able to obtain noticeably better results. In particular, as far as the encoder is concerned we tried transfer learning both from VGG16 and ResNet152, and eventually chose the latter as it performed consistently better.

We then proceeded to tune the hyperparameters and the structure of the model. Following are some of the parameters that influenced it:

- the learning rate was changed from 1e-5 to 5e-5 and eventually to 1e-4 to reduce oscillations

- Data Augmentation parameters: a parameter which seemed to influence the accuracy was the preprocessing function designed for ResNet

- adding a Learning Rate Plateau, that allowed us to dynamically change the model's learning rate whenever the val_loss parameter didn't change

Authors

Arianna Galzerano
Francesco Fulco Gonzales
Alberto Latino