Artificial Neural Networks and Deep Leaning

Homework 2: Image Segmentation

Giovanni Dispoto, Matteo Sacco

December 27, 2020

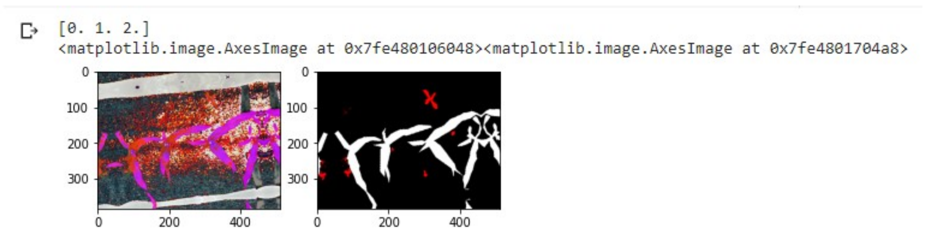
We implemented the model in the *Multiclass Segmentation Notebook*. We start with Bipbip Haricot Dataset. Data was not preprocessed and every image was rescaled at 256x256 dimension. The result was mediocre with a val_meanIoU ≈ 0.5, but with very low score on test set. When generating prediction on the test set, we saw that the prediction were very strange due to the downscaling of the image from 2048x1536 to 256x256 and then upscaling to the original size.

U-Net

Then, we tried U-Net (initially without skip connection) at this point we reached obtained val_meanIoU ≈ 0.6. The model was composed of 4 convolutional block and each block was composed by 2 Convolutional layers with ReLU activation function and a MaxPool layer. The number of filter starts from 32 in the first layer to 256. We have also applied input preprocessing function of VGG16.

The training was made using batch size = 8 for both training set and validation set, and learning rate = 1e-4. This values were obtained after a little of fine tuning.

Furthermore, we added skip connection which did improve our results.



```
[0. 1. 2.]
<matplotlib.image.AxesImage at 0x7fe480106048><matplotlib.image.AxesImage at 0x7fe4801704a8>
```

Figure 1: *Preprocessed image with input mask*

We then included also BipBip/Mais and obtain a result of val_meanIoU ≈ 0.7.

After this, we implemented **Image Tiling**. We preprocessed the dataset creating tiles of dimension 224x224. These images are used in the training set and the validation set was made of whole images. This allows us to not train the model on the whole image.

After few epochs it reach a value of val_meanIoU of 0.4.At this point, we have switched the learning rate from 1e-4 to 1e-5 and the model restarted to learn. We obtained val_meanIoU = 0.74 and val_loss ≈ 0.08.

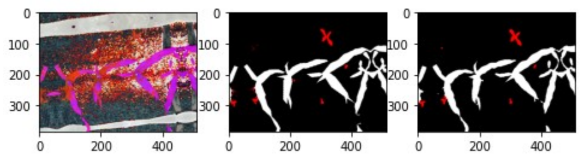
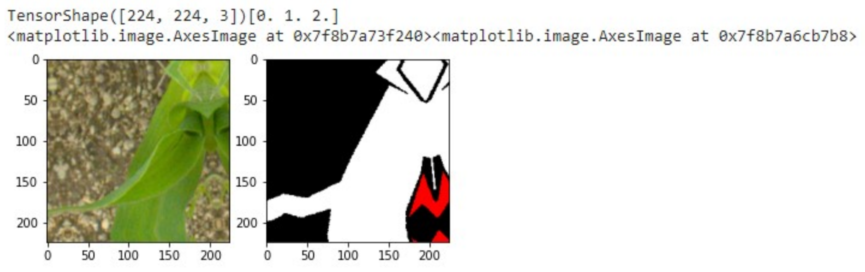


Figure 2: *Input image with mask and predicted mask from U-Net model with Skip Connections*



```
TensorShape([224, 224, 3])[0. 1. 2.]
<matplotlib.image.AxesImage at 0x7f8b7a73f240><matplotlib.image.AxesImage at 0x7f8b7a6cb7b8>
```

Figure 3: *Tiled image and mask*

FPN

Then we tried a different model called FPN (Feature Pyramid Network). Then reason why we thought this could perform better than U-Net is because FPN is particularly good at recognizing objects at different scales.

For the encoder parts we used VGG16, we also tried to use transfer learning on the encoder freezing the layers at different levels, but we never obtained better results compared to learning all weights.

The pyramid part works similarly to the U-Net decoder, although the layers from the encoder are added instead of concatenated.

We apply two convolutional layers to all the pyramid layers and we upscale all the layers but the last one so we can sum or concatenate them (we chose to sum them). Then we can work on the result to make our predictions.

We found FPN already implemented in a public github project: `github.com/qubvel/segmentation_models`.

The project also offered some loss functions and the possibility to combine them, the ones we studied and tried were:

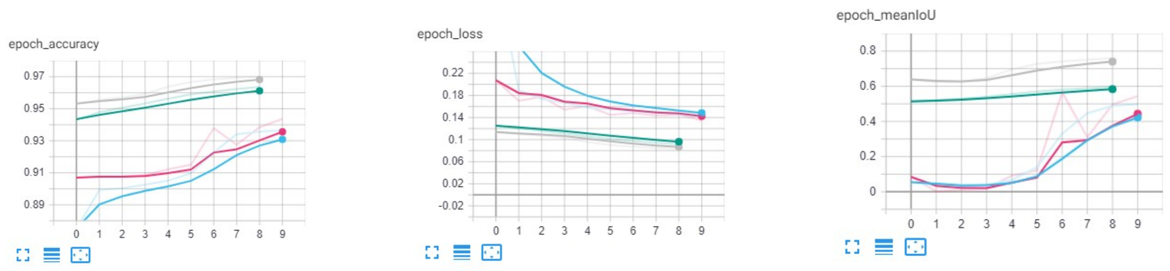
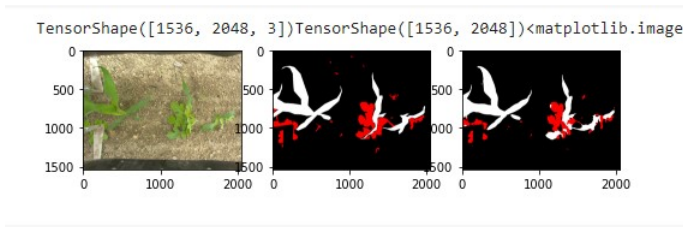
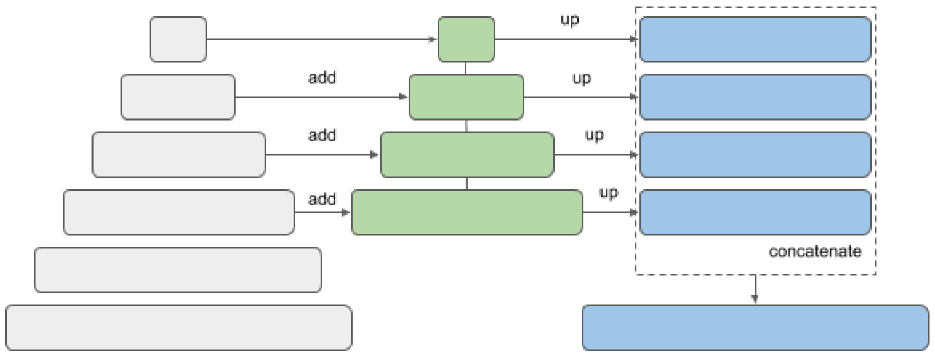


Figure 4: Training model on Bipbip dataset



```
TensorShape([1536, 2048, 3])TensorShape([1536, 2048])<matplotlib.image
```

Figure 5: *Input image with mask and predicted mask from U-Net model with Skip Connections trained on tiled images*



1. **JaccardLoss**: a different name of Intersection over Union
2. **DiceLoss**:
$$L(tp, fp, fn) = \frac{(1+\beta^2)\cdot tp}{(1+\beta^2)\cdot fp + \beta^2 \cdot fn + fp}$$
   Where β is the coefficient the class balance, since we can give difference balance weights to the classes
3. **FocalLoss**: down-weights easy examples and focus training on hard negatives

After several training attempts while fine tuning the model, we obtain the best results with the followings hyper parameters:

- batch size = 8
- learning rate = 10^{-4}
- loss = dice_loss + (0.5 * focal_loss)
  - Dice Loss class balances weights: background=0.5, white plants=1, red weed=1.5

Instead of using tiling this model was trained by using the zoom on ImageDataGenerator in this range [0.15,0.5], and using 448x448 images, the CustomDataset class has been tweaked to apply the zoom before we downscale the image to 448x448 so we don't lose any detail.

Unfortunately we could not obtain better results than the ones obtained on U-Net with skip connections and tiling.