Alex Bergamasco 10521973
Virginia Negri 10504560

Image Segmentation - Homework 2

---

1. Dataset

1. Split

Dataset is splitted using ImageDataGenerator, with 0.8/0.2 ratio between training set an validation set. The seed is hardcoded.

2. Augmentation

At the beginning, we tried to perform the same data augmentation as in the first challenge. The results we obtained were worse than without performing data augmentation, probably because changes like rotations and shifts are not meaningful for segmentation tasks. Thus we focused on flips, zoom and brightness.

2. Losses

1. Sparse Categorical Crossentropy

In our first models we used as loss function the Sparse Categorical Crossentropy. To use this loss we had as final layer in the network a softmax layer with 2 outputs. This however wasn't giving us great results.

2. Binary Crossentropy

We then tried using Binary Crossentropy using a last layer of sigmoid with only one output. With this loss our segmentation accuracy grow.

3. Dice

Finally we used the Dice Loss, the most used in image segmentation tasks. While making training a bit more unstable, it lead to a significant increase in accuracy.

3. Models

1. Custom network

1. Lab example

The first model we tested was the one provided as example during the second lab session. This model is made of two symmetric parts, an encoder and decoder. The encoder was made of down-sampling blocks, all containing a layer of convolution, a layer of max pooling and an activation layer. The decoder instead was made of up-sampling blocks, all containing a layer of upsampling, a convolution and an activation.

The model alone gave as a pretty poor performance, reaching a score not higher than 0.27 on the leaderboard.

We experimented with this model prior to building our own one by adding a few layers of batch normalisation to avoid overfitting, playing with the filter size and number at each convolution layer and with the depth of the encoder/decoder.

These changes however didn't give us meaningful results, thus leading us to build our own network (UNet architecture).

2. Custom UNet

Our most successful architecture is UNet. We built this model from scratch by taking inspiration from the state-of-the-art UNet. As in the previous model, the system is made of two symmetric components: a decoder and an encoder. The new addition however is the use of skip connections between the two parts. Precisely, in Keras we adopted Concatenate layers to concatenate the intermediate outputs of the decoder at each convolution block with the last output of the decoder at each convolution block. The significantly increased the global MAP. The reason for this is that the model takes advantage of features at different level of abstractness and this results in a more accurate segmentation.

With this architecture, after having tuned the depth of the encoder/decoder and the number and size of filters (specifically depth of 5, 3x3 filters of size 3x3) we experimented by adding the following components, that gave us both positive and negative results:

- Batch Normalisation: Normalisation layers significantly increased our network's precision. These were added between convolutional and activation layers. Batch normalisation helped both in speeding up training and reducing overfitting.
- Dropout layers: To further avoid the problem of overfitting we tried adding Dropout layers following activation layers and tuned their probability. In our experiments, however, this addition always resulted

in a poor performance of our network. Overfitting avoidance is already provided by data augmentation and batch normalization thus adding

- dropout layers simplifies the network too much by removing information and results in a poor overall precision.
- Weight Initialisation: In our solutions we experimented also by changing the weight initialisation. We observed that in all models He initialisation always gave us higher performances.

2. <u>Pretrained network</u>

1. VGG19

Another solution we experimented with was using a pretrained network as encoder and building a custom decoder to upsample the embedded image and obtain its segmentation. We used VGG19 as an encoder without retraining it, and added 5 blocks of convolutions and upsampling to build the decoder. We used the same convolution block as in UNet for the decoder and added skip connections with intermediate outputs of the VGG19.

As in the previous cases we used early stopping and data augmentation to avoid overfitting, in addition to the batch normalisation layers in the decoder.

We trained this network incrementally, 10 epochs at a time. It converged around 35-40 epochs giving us a score of approximately 0.52. While having a pretty good segmentation accuracy, it still wasn’t able to outperform our custom UNet that lead to our highest score on the leaderboard.

2. ResNet

This final solution is the one that gave us the highest results on the leaderboard. We implemented a UNet architecture using as pretrained encoder ResNet with ImageNet weights.

For this model we built a custom decoder connected to the ResNet encoder made of 5 upsampling blocks. Each block, as for the previous experimented models, is made of a convolution layer, a batch normalization layer and an activation. In order to get a UNet architecture we added skip connections from ResNet to the decoder.