INFO8010: Report template

Cédric Siboyabasore[1]

[1] *c.siboyabasore@student.uliege.be (s175202)*

I. INTRODUCTION

The problem I tackled consists in the semantic segmentation of certain objects and background types in an aerial image captured by a drone.
The data used to solve the problem statement was provided by Graz University of Technology and is freely available for academic purposes at [1]. It contains 400 annotated images of 6000 by 4000 pixels.
There are 22 labels plus two special labels ("unlabeled" and "conflicting") such that each pixel can be associated to a single class, those are: paved-area, dirt, grass, gravel, water, rocks, pool, vegetation, roof, wall, window, door, fence, fence-pole, person, dog, car, bicycle, tree, bald-tree, ar-marker and obstacle.

An example of image with its segmentation mask is given in FIG 1. The images were taken by high resolution camera mounted on drones. This was a project from the Graz university (Austria) which also distribute other associated datasets on request.
With this dataset, the university aims at increasing the safety of autonomous drone flight and landing procedures. This task is particularly innovative as few attempts were given at semantic segmentation from aerial views. It could be extended to other applications such as geo-sensing, in case of natural disasters to detect living beings, etc... The task of increasing safety by applying semantic segmentation could be also be used for autonomous driving, such that cars can identify and classify items in real-time and avoid accidents.

To tackle this problem, I implemented my own fully convolutional neural networks and I also downloaded ResNet 152 network that was pre-trained on ImageNet[2].

FIG. 1. Image and its segmentation mask

II. RELATED WORK

The dataset is provided by Graz University of Technology and aims at increasing the safety of autonomous drone flight and landing procedures. This project is quite innovative as the images of the dataset captures urban settings from a bird's point of view.

Researchers from the Ohio State University, University of Twente and Wuhan University collaborated on a semantic segmentation project based on an aerial videos dataset [3].

To solve this task I created some FCN (*fully convolution networks*) because they are particularly well suited for this task. Indeed their structure is able to work on may different image size as it looks on a neighboring are of every pixel to classify them. A well known example of FCN is U-Net (2015) [4] in the context of biomedical imagery which was trained on MRI images to detect tumorous cells apart from normal cells. Another example of this type of architecture comes from [5] in which the authors adapted state of the art image classifier networks (AlexNet, VGG16, GoogLeNet) to construct a FCN version. They showed significant improvement on the PASCAL VOC 2011 and 2012 datasets which is a strong motivation to use this approach.

Before I start with the main body of the work, I want to clarify that I put the Methods and Results sections together because a lot of my architecture choices were motivated by the results that I had.

III. METHODS AND RESULTS

To accomplish the segmentation task, I first used the PASCAL VOC 2012 dataset [6], which was a challenge divided into several competitions, one of which being a semantic segmentation competition. The dataset for this challenge is composed of photographs of objects in realistic scenes containing 20 object classes. For each image of the segmentation dataset, the prediction masks where each pixel is labelled with the ground truth class (a value greater than 0) or background (0) are also provided. The segmentation training set was made of 1464 images.
My plan was to perform transfer learning : I wanted to train a fully convolutional neural network on the segmentation dataset and then use this network as feature extractor for the network on the semantic drone dataset for smart initialization.
The images of the PASCAL VOC 2012 dataset all have a dimension (width or height) equal to 500, with the other one being smaller. The higher the resolution of an image, the longer it takes to load it. During the preprocessing of the PASCAL VOC data, instead of resizing the images, which would result in a loss of information, I augment the data by cropping the images several times at random locations. The cropped images have a fixed size and the same region is cropped in both input images and labels. To have crops which still look like the objects in the original images, I take 4 crops of arbitrary size 250x250 of the original image. I crop several times to have a representation of the 500xY original image (with $Y < 500$) and I chose to do it 4 times because it intuitively seems enough to not have crops overlapping on the same region of the original image.
Introducing randomness with random croppings makes the model less prone to overfitting because the model depends less on the position of the object in the image.
The first network I built for this dataset is a fully convolutional network for which the encoder part is composed of 4 convolutional layers each followed by a ReLU activation and a pooling operation:
`Conv1`, `Relu`, `Pool1`, `Conv2`, `Relu`, `Pool3`, `Conv3`, `Relu`, `Pool3`, `Conv4`, `Relu`, `Pool4`.
My intuition when building the network was to stack several convolution-ReLU-pooling patterns a few times in an identical fashion to have convolutions that start looking at patterns which are more and more global in the image. I was initially concerned with computation time and therefore to reduce it, I configured the convolutional layers parameters to have a Kernel size of $k = 4$, a padding $p = 1$, a stride of $s = 2$ and a dilation of $d = 1$ so that the output is of half the size of the input image :

$$\begin{cases} H_{out} = \frac{H_{in}+2*p-d*(k-1)-1}{s} + 1 = \frac{H_{in}}{2} \\ W_{out} = \frac{W_{in}+2*p-d*(k-1)-1}{s} + 1 = \frac{W_{in}}{2} \end{cases}$$

The pooling layers have the same parameters. `Conv1` increases the number of channels from 3 to 21 and this number is conserved in the rest of the network.

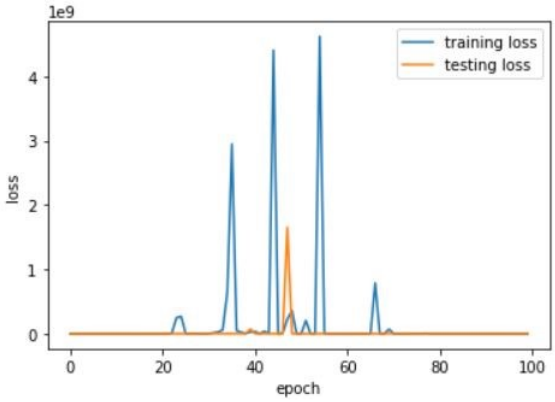

FIG. 2. Losses for the first FCN architecture

I repeated the convolution-ReLU-pooling pattern 4 times because after the 4th pooling layer, the image reaches a minimum size of 1x1.
The decoder part of the network followed the same scheme as the encoder, except that the convolution layers are replaced by transposed convolutional layers and the pooling layers are replaced by unpooling layers to upsample the feature map back to the original image resolution.
`ConvTransp1`, `Relu`, `Unpool1`, `ConvTransp2`, `Relu`, `Unpool3`, `ConvTransp3`, `Relu`, `Unpool3`, `ConvTransp4`, `Relu`, `Unpool4`.
They have the same parameters as their corresponding layers in the encoder and the unpooling layers also take the size to which they must upscale the image, because ambiguity is possible if we divided an odd image dimension by 2 and rounded it in the encoder.
The output of the network is a tensor of 21 channels with each channel being a binary matrix of dimensions 250x250 where the value of a pixel is 1 if the number of the channel corresponds to the class of the pixel.
The loss I first used to train the model was the classical cross-entropy loss.
Adam and Stochastic Gradient with Momentum are the default optimizers in Deep Learning. However, with SGD, you have 1 more parameter alpha which is tricky to adjust. For more simplicity, I first used Adam with the default learning rate of 0.001 and I train and test the model during 100 epochs.

After training this model and testing it on the validation set provided in the Pascal VOC segmentation competition, we can see on the losses curves on Figure 2 that at some points there's a numerical instability and a big jump in the losses values. The network might seem to recover after but it can be seen on Figure 3 that it doesn't recover the pixel-accuracy it had before. This is due to an exploding gradient.
To cope with this gradient issue, I added a Batch normalization after each ReLU in the fully convolutional neural network. The architecture becomes:

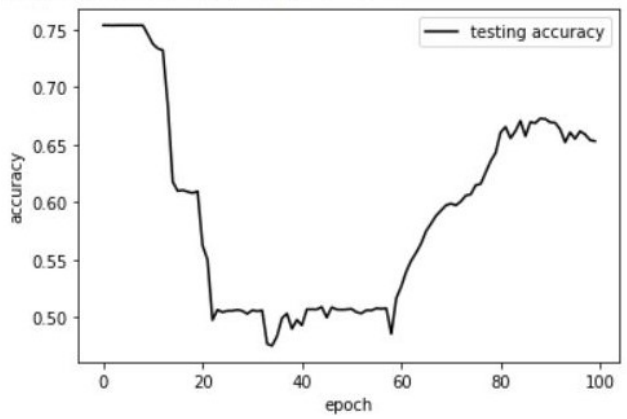

FIG. 3. Losses for the first FCN architecture

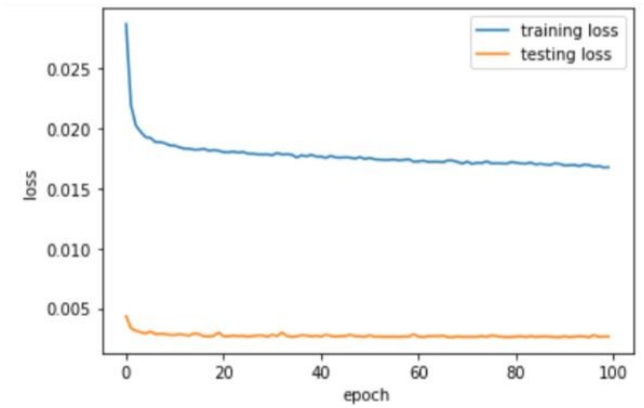

FIG. 4. Losses for the first FCN architecture with normalizations

`ConvTransp1`, `Relu`, `Batchnorm`, `Unpool1`, `ConvTransp2`, `Relu`, `Batchnorm`, `Unpool3`, `ConvTransp3`, `Relu`, `Batchnorm`, `Unpool3`, `ConvTransp4`, `Relu`, `Batchnorm`, `Unpool4`.

As can be seen on Figures and 5, the losses are much more stable and are decreasing and the reached accuracy is high, reaching around 0.76. I visually assess the output of the model of an image of the testing set containing 3 chairs and a table and I compare it with the true output predicted mask: We can see that that a lot of the classes output by my models were wrongly predicted. The model doesn't correctly detect and delimit the chairs and the table.

The reason the accuracy was so high is that most of the pixels in the true segmentation mask are labelled with the background class. Only assessing the model on the accuracy curve might therefore a bad idea, assessing the predictions with the human eye might be a better alternative.

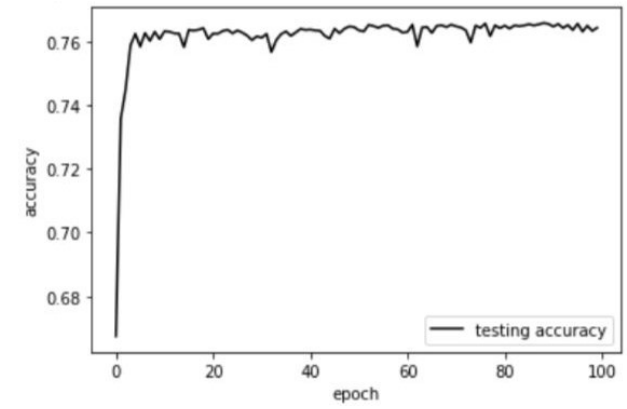

FIG. 5. Losses for the first FCN architecture with normalization

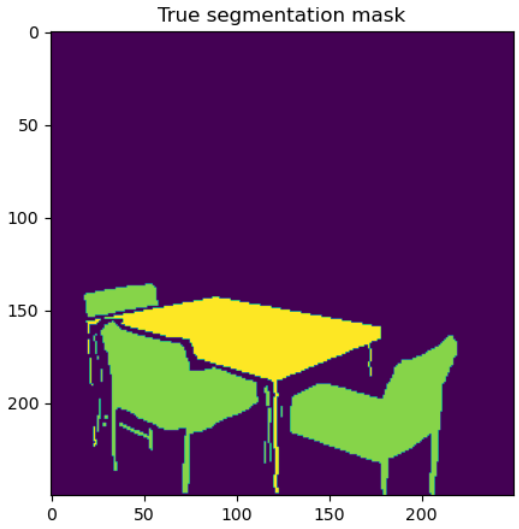

FIG. 6. True segmentation mask of a picture containing chairs

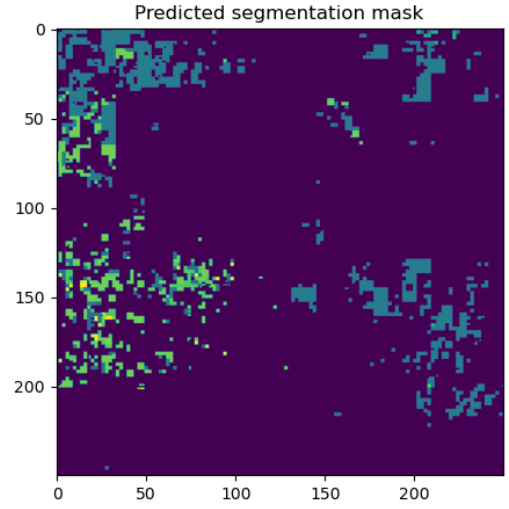

FIG. 7. Mask predicted by my first FCN, using cross-entropy

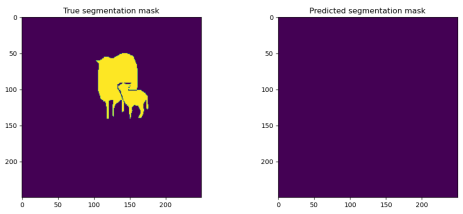

FIG. 8. Mask predicted by my first FCN, using Dice loss

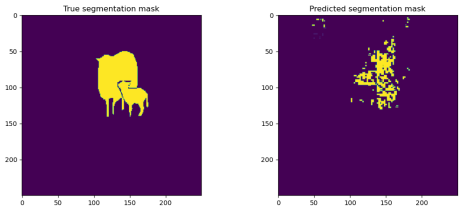

FIG. 9. Mask predicted by my first FCN, using cross-entropy

One modification we can apply to the learning algorithm is to change the loss to a loss which might be more appropriate for a semantic segmentation problem. A loss widely used for semantic segmentation is the Dice Loss, which is based on the Sørensen-Dice coefficient and is suited for high unbalanced dataset [7]. This is the case here since there are more background classes than others. For a multi-class segmentation problem, the formula [8] of the dice loss is given by

$$DL(y, \hat{p}) = 1 - \frac{2y\hat{p} + 1}{y + \hat{p} + 1}$$

where y is the true pixel value, $\hat{p}$ is the output probability of the predicted class and 1 is added in the denominator to prevent division by 0.
To implement this loss, I used one-hot encoding to make the output shape match the shape of the predicted segmentation mask.
The outputs of the fully convolutional network trained with the Dice Loss were deplorable, as the model now always outputs the background class for every pixel. An example is given in Figure 8.
We can see on Figure 9 that the model trained with the cross-entropy loss tested on the same example outputs a better prediction. Even if the predicted mask doesn't show two sheeps or three chairs and a table in Figure 7, the model trained with the cross-entropy loss seems to have more or less located the sheeps and understood that pixels in that area have the same class, which is different from the background class. These observations made me choose to discard the Dice loss and dig deeper into the problem while using the cross-entropy loss.
The problem may not come from the loss used to train the model, but from the architecture of the network itself. My intuition to reduce the input image to

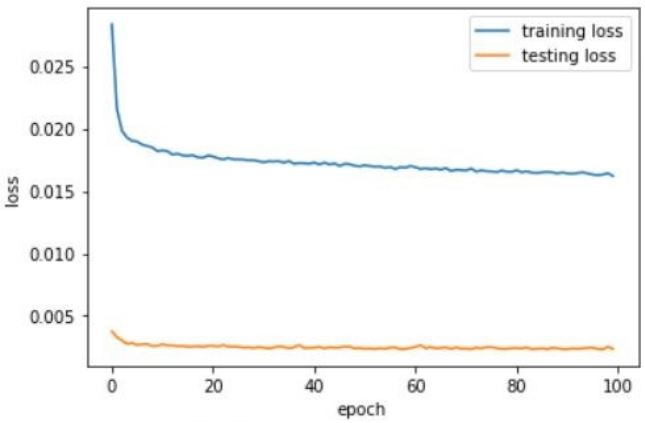

FIG. 10. Loss of my 2nd FCN, using cross-entropy loss

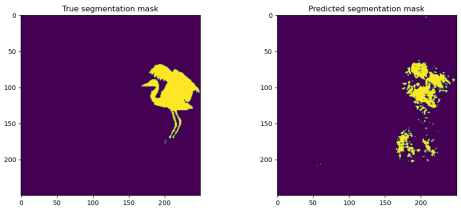

FIG. 11. Mask predicted by my 2nd FCN, using cross entropy

size 1x1 in the encoder was to take advantage of the convolution-pooling stacking to extract more features. My main concern with the convolutional layers was that keeping the image at its original size would maybe be too computationally expensive. Therefore, I had parametrized the Convolutional layers such that their outputs is half the size of their inputs. But reducing the image at each layer until size 1x1 maybe discards too much information and maybe the reason of my poor predictions.

Therefore, I built a second fully convolutional network where I parametrized the convolutional layers and transposed convolutional layers with Kernel size $k = 3$, padding $p = 1$, stride of $s = 1$ and a dilation of $d = 1$ so that the output size of a convolutional layer is equal to the size of its input image :

$$\begin{cases} H_{out} = \frac{H_{in} + 2*p - d*(k-1) - 1}{s} + 1 = H_{in} \\ W_{out} = \frac{W_{in} + 2*p - d*(k-1) - 1}{s} + 1 = W_{in} \end{cases}$$

Only the pooling layers will be responsible for the reduction of the feature map. I keep the same number of layers. The output size of the encoder part won't shrink to 1x1, which would hopefully help keeping the global structure of the image.

The losses curves of my second fully convolutional network in Figure 10 do not seem that much different from those of my first fully convolutional network in Figure

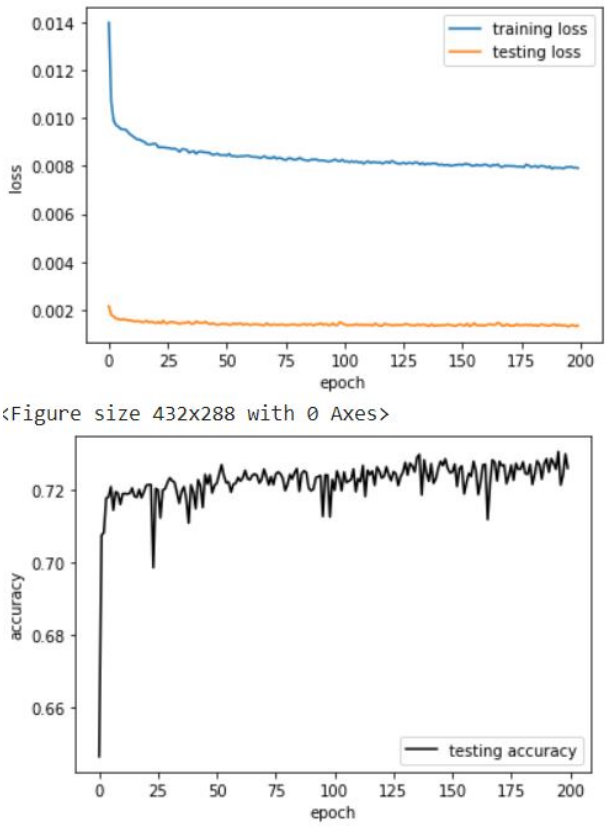

FIG. 12. Losses of my 2nd FCN + additional layer on 200 epochs, using cross-entropy loss

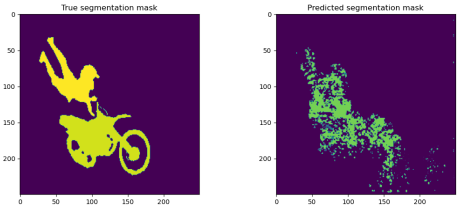

FIG. 13. Mask predicted by my 2nd FCN + additional layer on 200 epochs, using cross-entropy

III. On Figure 11, we can see that the model has amore or less located the bird.
I trained the same model on 200 epochs and added another convolutional layer in the encoder (and therefore another transposed convolution layer in the decoder) but no real improvement can be observed whether it is in the losses curves (Figure 12) or in the predicted masks (Figure 13).

Even though I have not succeeded in implementing an architecture that could provide a more or less accurate prediction of the Pascal VOC images, the model was able to correctly locate some objects and output the right class for some pixels of these objects. These results could be encouraging for transfer learning:
the next step is to use the last model which was trained

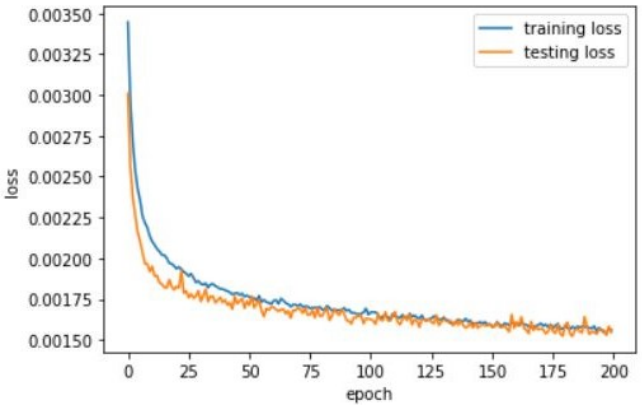

FIG. 14. Losses of the FCN which benefited from transfer learning

FIG. 15. Original image

on the Pascal VOC dataset by initializing its parameters to retrain the same architecture on the semantic drone dataset.
After loading the pre-trained model, it is necessary to map the number of classes of the Pascal VOC dataset with the number of classes of my dataset. To do that, I added a transposed convolutional layer which maps the 21 input channels to 26 output channels. The images of the drone dataset were captured using a 6000 by 4000 pixels camera. Because of this high resolution, the training process without any resizing would be very memory intensive. I tried to do it, but my operating system rejected my request by returning a RunTimeError because my computer couldn't allocate 18144000000 bytes. Therefore, all images were resized to a 600x400 resolution.
To help the model focus on smaller classes, each image was cropped into 2 by 2 array of 300x200 images without any further resizing. This step improved the size of the training data and the training time and testing time because I stored the different parts on disk. The preprocessing also included random horizontal and vertical flips.

After continuing to train the model on the semantic

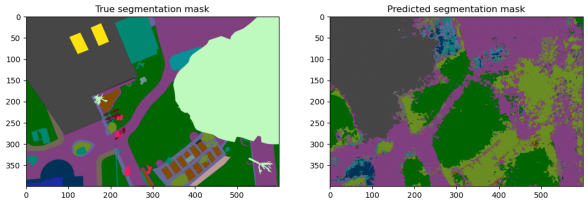

FIG. 16. True mask vs mask predicted by pre-trained

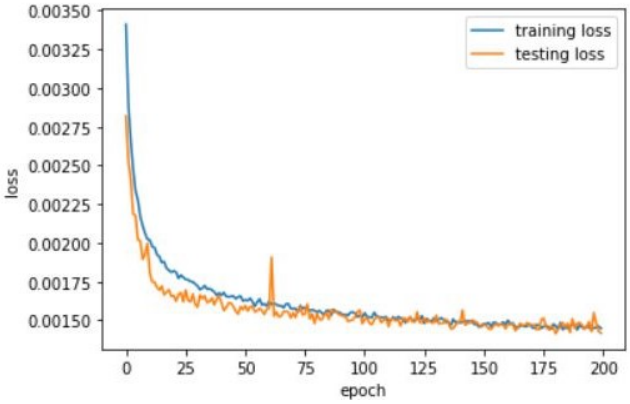

FIG. 17. Losses of the FCN which was trained from scratch

drone dataset during 200 epochs, the losses curves in Figure 14 is obtained.
After testing the model on the 600x400 image in Figure 15, the mask on the right in Figure 16 is obtained. We can see that the model has learned to delimit the different objects and to label some of them correctly. We can see that the model has not recognized the smaller classes such as the humans (normally in red) or the windows on the roof.

When training the network from scratch but also on the 300x200 images, the losses curves which are given in Figure 17 do not seem different from the ones of the pre-trained model given in Figure 14. The prediction given on the right in Figure 18 do not either seem to differ from, except for a few details

On some images, the fully convolutional network trained from scratch seems to detect smaller classes better than the pre-trained one (Figures 19 and 20.

This comparaison between pre-trained fully convolutional network and fully convolutional network trained from scratch might have been erroneous somewhere because transfer learning usually produces better performance than if we trained from scratch.
The architecture of the model I built or the pre-processing of the data was maybe not appropriate for the Pascal VOC dataset. Since this model yielded bad segmentation predictions, the parameters learned by

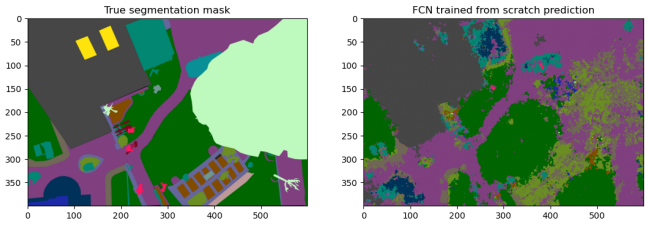

FIG. 18. True mask vs mask predicted by trained from scratch

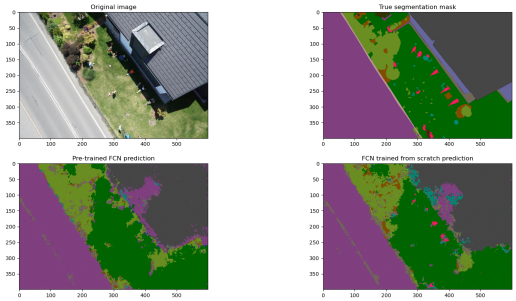

FIG. 19. Comparisons of predictions

the model might not have been correct for a semantic segmentation problem and using these parameters as initial parameters of the model trained on the semantic drone dataset might have 'penalized' the model and this might explain why smaller classes are not detected.

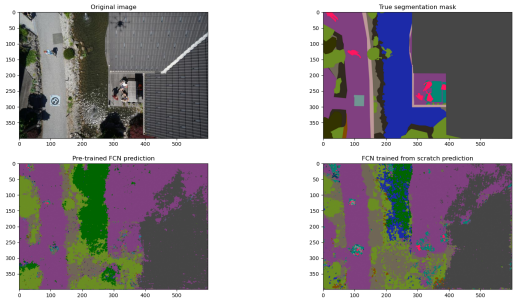

FIG. 20. Comparisons of predictions

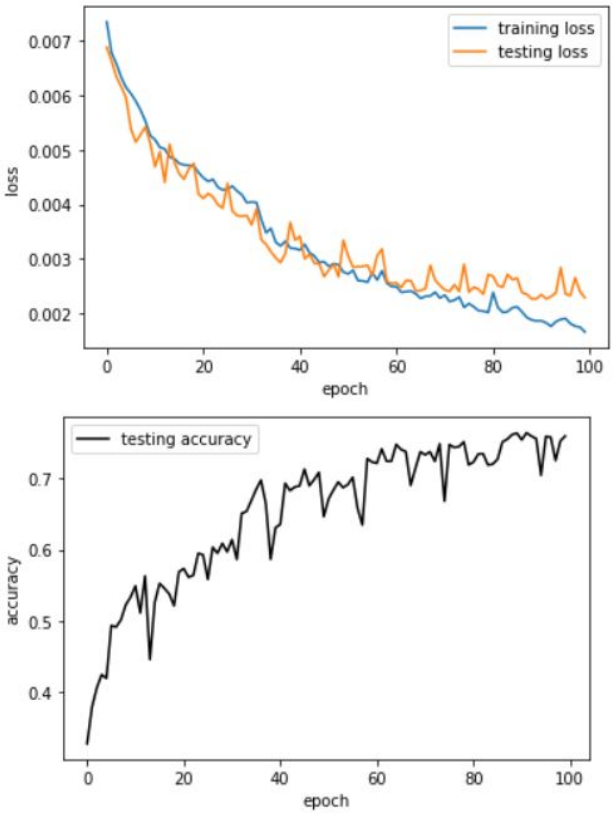

FIG. 21. Losses and accuracy of FCN with ResNet 152 as backbone

I then performed transfer learning with another architecture: I downloaded a ResNet152 network which was pre-trained on ImageNet (1000 possible classes) [2]. Since this ResNet was trained for image classification and not on segmentation, I removed its two last layers (`AdapativeAvgPool2d(output_size=(1, 1))` and `Linear(in_features=2048, out_features=1000, bias=True)`. The input images of ResNet must be of size 224x224. Since I removed the two last layers of ResNet, the output size 7x7 with 2048 channels. Since I want to predict 224x224 segmentation masks with 24 classes, I add 5 transposed convolution layers at the end of ResNet with a Kernel size $k = 2$ and a stride $s = 2$. I only use transposed convolution layers and no unpooling layers because I would have needed the sizes to which the image must be upsampled and for that, I would have needed the indices provided by the pooling layers inside ResNet. I progressively decrease the number of channels of the transposed convolutions from 2048 to 24 because inside the ResNet, they progressively increase it to 2048. Because of the depth of ResNet, I trained the network during 100 epochs otherwise it would have taken too much time. Given the losses curves in Figure 21, it might have been a good idea because the network seems to start overfitting at the end.

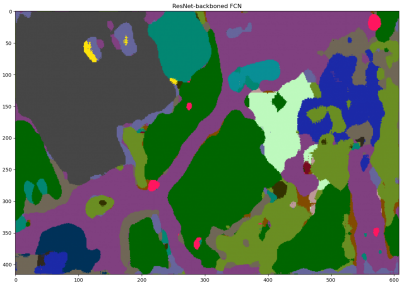

FIG. 22. Segmentation mask

-

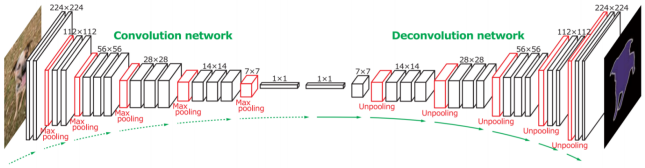

FIG. 23. Image and its segmentation mask

The prediction of the network on the image in Figure 15 is given in Figure 22. The prediction is a lot better than the predictions of the pixel classes of the previous architectures. There also seems to be less problems with the detection of smaller classes : the model is able to detect the car whereas it was not really made possible by the previous architectures.

I built a fourth fully convolutional network which had more parameters than the first ones; for the convolution network I took inspiration from VGG16 [9], which the authors of [10] based their fully convolutional network on.
It was a complex succession of convolutional layers, batch normalizations, ReLU activations and pooling. There's a total of 13 convolutional layers and more parameters. Its architecture is shown in Figure he deconvolution network therefore was a succession of the same operations as in the convolution part except that it had 13 transposed convolution layers instead of the convolution layers.

Such as in paper [10], I first resized the images to a 250x250 resolution, then for each image in the training set I take 3 random crops of 224x224 (which is a large enough subset of 250x250) that I feed to the fully convolutional neural network.

We can see on the masks in Figures 26 and 27 predicted by the fully convolutional neural network inspired from VGG16 on a 600x400 image that the model can also detect smaller classes.

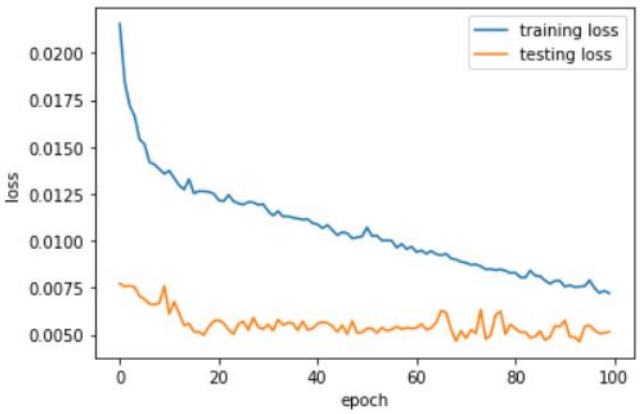

FIG. 24. Losses of FCN inspired from VGG16

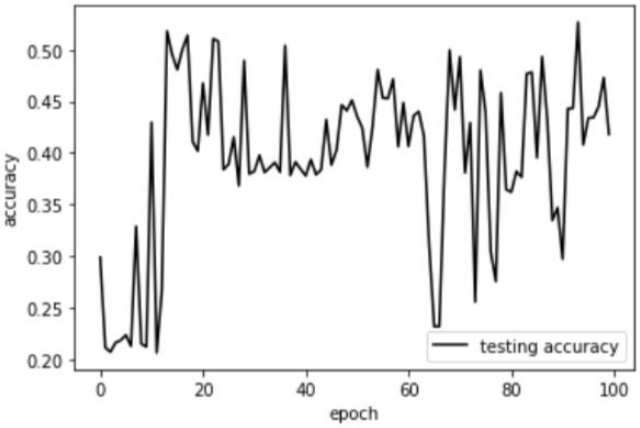

FIG. 25. Losses of FCN inspired from VGG16

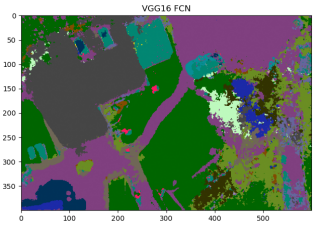

FIG. 26. Mask predicted by FCN inspired from VGG16

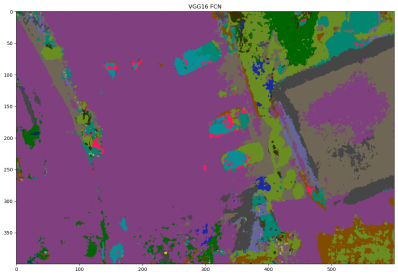

FIG. 27. Mask predicted by FCN inspired from VGG16

IV. DISCUSSION

The previous section showed that the networks I architectured myself from scratch performed worse than famous existing solutions such as the VGG-16 inspired fully convolutional neural network and the fully convolutional network with a ResNet152 backbone.

A first limitation comes from the image size, as the images are capture using a 6000 by 4000 pixels camera (which is a bit larger than 6K) the training process was very memory intensive. I had to drop the resolution to 600 by 400 to be able to train the network and the batch size was still very limited. In addition to the available memory of the device, the training was essentially memory bound: forward and backward passes of the network were much faster than loading time the images. As it takes about 500ms to load a single image it would be impossible to perform inference directly on the drone in real time.

Another limitation might be the fact that I did not vary the learning rate enough, I kept it at its default value of 0.001. I could have tried to change it to see the effects, either manually or by using a learning rate scheduler,...

I should have tried other architectures on the Pascal VOC dataset to have one that predicted the pixels better. I could have pre-trained the last VGG-16 inspired fully convolutional network on the Pascal VOC dataset and then transfered it on the drona dataset which would have yielded better results than if the drone dataset was trained from scratch.

I could also have used more losses than the cross-entropy loss and the Dice Loss. A lot more losses for semantic segmentation were proposed in [8].

I could also have tried another way of testing my images: since I trained some of my models on the 2 by 2 300x200 crops, for the testing I could have output the predictions for each of the 4 crops of an image and then assembled them and compared it with the direct prediction of the 600x400 image.

[1] Graz University of Technology. http://dronedataset.icg.tugraz.at.

[2] Princeton University Stanford Vision Lab, Stanford University. https://www.image-net.org/.

[3] Ye Lyu, George Vosselman, Gui-Song Xia, and Alper Yilmaz. Uavid: A semantic segmentation dataset for UAV imagery. https://www.researchgate.net/publication/341766272_UAVid_A_semantic_segmentation_dataset_for_UAV_imagery.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[5] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[6] Mark Everingham. Visual object classes challenge 2012 (voc2012), 2012. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html.

[7] Tom Vercauteren Sébastien Ourselin M. Jorge Cardoso Carole H Sudre, Wenqi Li. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations, 2017.

[8] Shruti Jadon. A survey of loss functions for semantic segmentation, 2020.

[9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[10] Bohyung Han Hyeonwoo Noh, Seunghoon Hong. Learning deconvolution network for semantic segmentation, 2015.