Object Detection with Fully Convolutional Neural Networks

Martin Bernardi and Zhanbota Bissaliyeva

I. Introduction

This laboratory aims to implement simple FCNs (Fully Convolutional Networks) for object detection in Tensorflow [1]. The objective is not the development of an optimized network for a difficult object detection problem, but instead to solve a simple problem that allows to learn several topics related to Deep Learning.

The problem itself consists in the detection and localization of a football ball in images, where the ball is always the same image with different transformations applied to it (random position, size, rotation and brightness). Other characteristics of the problem is that the location of the ball has no relation with the image itself, it can be located in random places and have a scale that has no relation with the distances and focal length used in the picture (for example the ball can be small and located over a cup of coffee on the image).

One of the objectives is to experiment with FCNs, which allows the inference of variable size images as input. Another objective is the development of a project of medium complexity with the possibility to train a big number of models, do hyperparameter search, checkpointing, saving and loading, and evaluation of models.

The selected problem was chosen to allow the easy generation of a synthetic dataset. This way it is possible to experiment and evaluate better the influence of the dataset in the training and inference of the neural network. Specifically it is possible to train a network with a given synthetic dataset and then evaluating the performance in another generated dataset with slightly different parameters. For example, train a network with fixed size football images and evaluating the performance in images with variable football size.

Finally, another advantage of the task chosen is that it can be solved in relatively small datasets and training times. The approach used in this work is to transform the object detection task into a segmentation problem that can be solved with FCNs. The input consists of the image together with a segmentation mask showing the pixels that belong to the ball. As an output, instead of obtaining the position of the object, a heatmap of probabilities is returned. As a result, postprocessing must be done to determine the position of the ball from the heatmap.

II. Literature

A. Dataset

The ball used is a single image that is pasted over a random background picture from a folder. For the background images there are not special requirements, so the COCO Val 2017 dataset with 5000 pictures was used. The ball images tested in particular are shown in Figure 1 together with an example of a generated image.

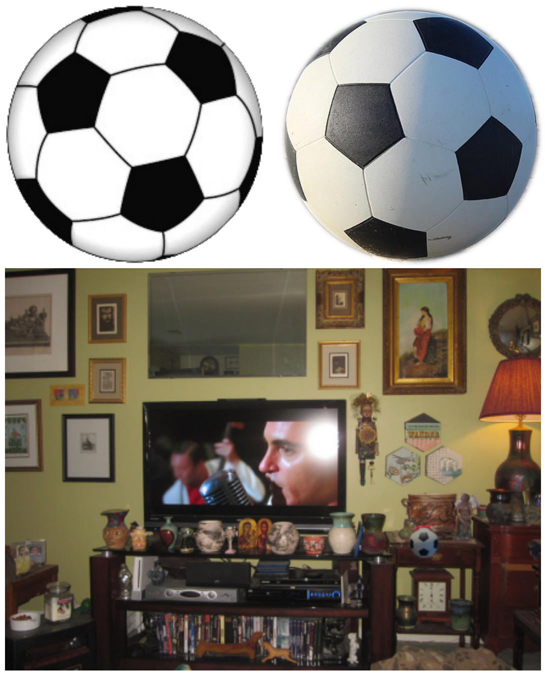

Fig. 1: *Object images used when generating datasets. The first one is used in the "simple" dataset and the second [4] in the more "complex" one. Below, an example of a generated image.*

The dataset is created in the go by the usage of a Python generator together with *tf.data.Dataset.from_generator()*, This means that the train and testing batches are generated when they are needed. Also, the background images are selected randomly, and depending on the quantity they may be used repeatedly or not.

We call the "simple" dataset to the case where the first image in Figure 1 is pasted with a fixed size, no rotation, and without changing the brightness of the ball. The "complex" dataset is where the second ball image is used, with all the transformations listed above. Overall, the generation process consists in the following steps:

- A background image is loaded from disk and it is cropped depending in the configuration.
- The ball is pasted in a random position, with size, rotation and brightness depending in the configuration.
- A ground truth image is generated. It is an image full of zeros except in for the pixels that belong to the ball.

- All the images of the batch are padded so they have the same size, by taking into account if there are any divisibility requirements. This is because the network may need images with sizes divisible by a certain number, for example 16.

For training, cropped images to a specific size were utilized, but for inference, full size images with the required padding are used. The manipulation of images in Python was done with the Pillow library [5].

B. Network architecture

Any FCN model consists of convolution blocks of 2D convolution layers (Conv2D and Conv2DTranspose) and the required regularization (Dropout and Max Pooling), which prevents overfitting and helps in quick convergence. Convolutional layers should have same padding to preserve dimension and output a final segmentation map. Meaning, the model will learn the mapping from the input image to its corresponding segmentation map through the successive transformation of feature mappings.

Something to consider is the dimension of the images; in this case, each Max Pooling layer outputs an image with half the size while each Conv2DTranspose doubles the size. Initially, one does the opposite of the other and an image of the original size should be obtained, but this is only true if the intermediate images never have an odd size. This is the reason why depending in the number of layers, it is required that the image dimensions are divisible by 2, 4, 8, 16, etc.

An example of a fully convolutional network is the U-net, which is a famous network that is used for semantic segmentation. A modification done was that now the size of input images is variable, and that the last layer is not for multiclass classification but for binary classification: The loss function was changed from SparseCategoricalCrossentropy to BinaryCrossentropy, and the last layer now has only 1 filter with sigmoid activation. The implementation of U-Net is based in [3].

C. Postprocessing

The output of the neural network consists of an image with a probabilities of a given pixel to be part of the ball. To be able to get the position and size of the ball, the center of mass of the pixels is calculated after the application of a threshold. The calculation of the center of mass is done as follows:

```
total = mask.sum()
h, w = mask.shape
x_i = np.arange(0, w) # [0, 1, 2, 3, 4...]
y_i = np.arange(0, h) # [0, 1, 2, 3, 4...]

x = (x_i * mask.sum(axis=0)).sum() / total
y = (y_i * mask.sum(axis=1)).sum() / total
```

The disadvantage of this method is that it works only if there are no false positives far from the object. In some cases, small blobs will be incorrectly detected and will move considerably the center of mass. This is discussed and a solution is proposed in Section III-E.

D. Tooling

A big part of the work consisted in the development of software that simplifies the process of training and evaluating models. The features include:

- The posibility to train and save models. Together with the weights of the model itself, a JSON file is included with the history of the evaluation metrics for each epoch, the hyperparameters used, the parameters of the dataset, the network architecture, etc.
- Functions to evaluate and plot the models, examples of these plots are shown in the various Figures in this report. Plots of the learning metrics for each training epoch can be done. Also, example images can be generated and shown to evaluate visually the inference of the model.
- Functions to list the models saved in the computer, allowing to filter and search for them depending on the contents of the JSON file.
- The dataset generator and the postprocessing functions described in the previous sections.

The code is available in Github. For model training in Google Colab, it is possible to clone the repository and import modules from the code. This way, models can be trained and later downloaded to the computer for evaluation

Thanks to the tooling created it was possible to train over 70 models (in our computers and Google Colab), copy them to a personal computer and finally compare and evaluate.

The possibility to do training checkpointing to resume an interrupted training was tested but finally not implemented.

III. Results and explanation

A. Simple FCN for simple dataset

In this section we explore the problems and performance of the simple FCN of 12 layers when using the simple dataset: 4 sets of Conv2D+MaxPooling, followed by 4 Conv2DTranspose layers. One of the parameters to choose is the size of images to use when training, which will have a direct impact in the balance of the classes. This is because the ball has a fixed size of 50x50, so if the image is big, most of the pixels are of class zero.

Using big images not only makes the training slower because of the computer requirements, but also produces that in the first epochs the network will learn to predict everything as zeros before starting to give good results as observed in Figure 2. In the opposite case, when using very small images, the network will tend to give lots of false positives when evaluating in full scale images.

It was observed that the number of layers in the neural network should have some correspondence with the size of the ball. This can be because the convolutions and pooling operations chosen operate in a small region of 3x3 and 2x2 respectively, and if we want to learn global features of the entire ball, a certain number of layers will be needed to allow the information to propagate.

In any case, the results obtained are very satisfactory, as shown in Figure 3 the ball is segmented perfectly in most of the images. The pixel by pixel precision evaluation in 160 images is 99.29%, and the recall 99.449%.

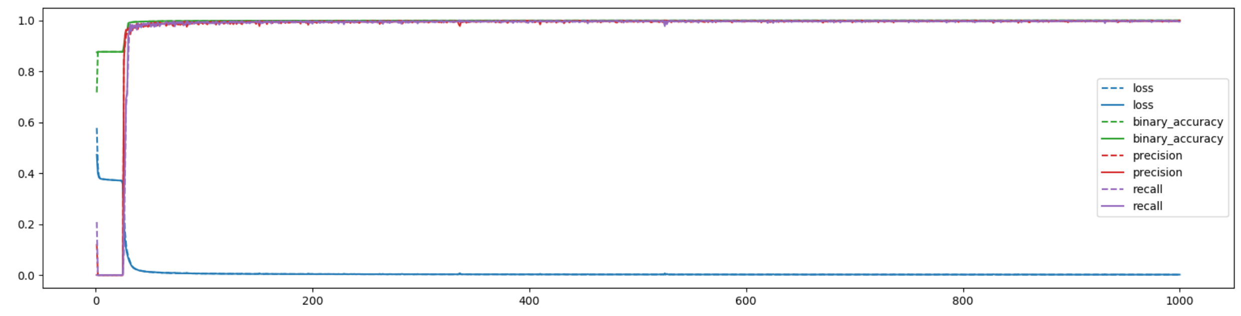

Fig. 2: *Training of simple FCN. Observe that at first, the network predicted all zeros. Dashed lines are evaluations in training set.*

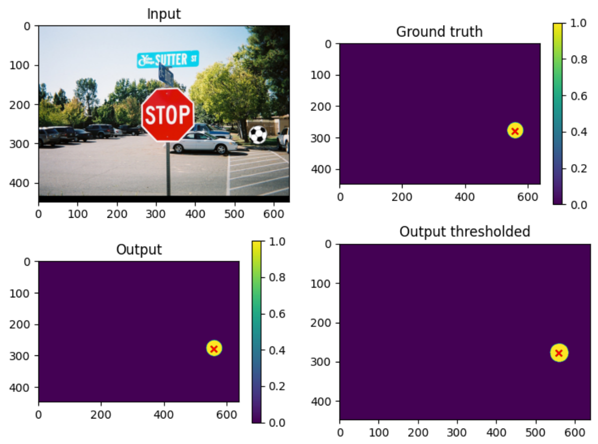

Fig. 3: *Example of a prediction from the simple FCN. A red cross is shown in the detected position.*

B. U-Net for simple dataset

The U-Net architecture is used for the same dataset, and as expected, the results are very good. In Figure 4 it is possible to observe the training of the network with 32 filters in two different size of images. Precision values obtained are around 99.11% and recall of 99.94%.

Later, networks with different number of filters were trained. Good results are obtained for any amount of filters, although as observed in Figure 5 more filters is better. A particular case is when 4 filters were used, where the network failed to train properly and gave predictions with regular pattern artifacts (see Figure 6). When training the network again from scratch we obtained good results as expected. In Figure 7 a example for the simplest network of 2 filters is shown, demonstrating that this architecture is good for most cases. In any case, we consider the usage of 16 as the best compromise between training time and performance, so it is used for future trials.

Next, to better understand the behavior of the network, tests were done in a dataset with different parameters. When using ball images of size 75x75 instead of 50x50, the prediction shows false holes in what should be the ball (see Figure 8). If the ball is further increased to 100x100 the network predicts a completely empty image. In a similar manner, when reducing the size of the ball, the predictions start to show problems when the size is smaller than 35x35.

When rotating the ball, it is still always detected and located properly but the segmentation is not a perfect circle anymore (see Figure 8). When decreasing the brightness, it starts to show problems at 0.6 and below 0.5 the network is no longer usable. Finally, when using the other ball image, no detections are done.

C. U-Net for complex dataset

When training the U-Net of 16 filters in the complex dataset of variable sized balls, with rotation and brightness changes, it was observed that it works satisfactorily. Figure 9 shows the training and Figure 10 shows an example. Precision is 95.92% and Recall 98.87%.

D. Simple FCN for complex dataset

As a final test, we returned to the simple FCN architecture to test the performance in the complex dataset. It was required to increase the number epochs to 300 (see Figure 11). The results were also quite satisfactory (see Figure 12) but considerably worse than in previous cases. A precision of 79.87% and a recall of 97.50% requires some way to ignore false positive detections as shown in the next section.

E. Postprocessing

As shown in previous sections, in some cases it becomes necessary to ignore blobs due to false positive detections. A simple solution is to select the biggest connected region of pixels and calculate the center of mass of it. Before this task, an initial morphological erosion of radius 2 is done to remove the smallest objects and to separate potentially connected blobs. Examples can be observed in Figure 13.

IV. Conclusion

With respect to the selected problem. It was observed that the detection of a fixed object can be done easily with FCNs, even if there are variations in size, rotation and brightness. In the future it should be interesting to observe the influence in noise, to determine if the network is learning mainly from low level or high level features.

Right now, the problem can also be solved by doing convolutions of a template against all the image instead of using Deep Neural Networks. It would be interesting to obtain

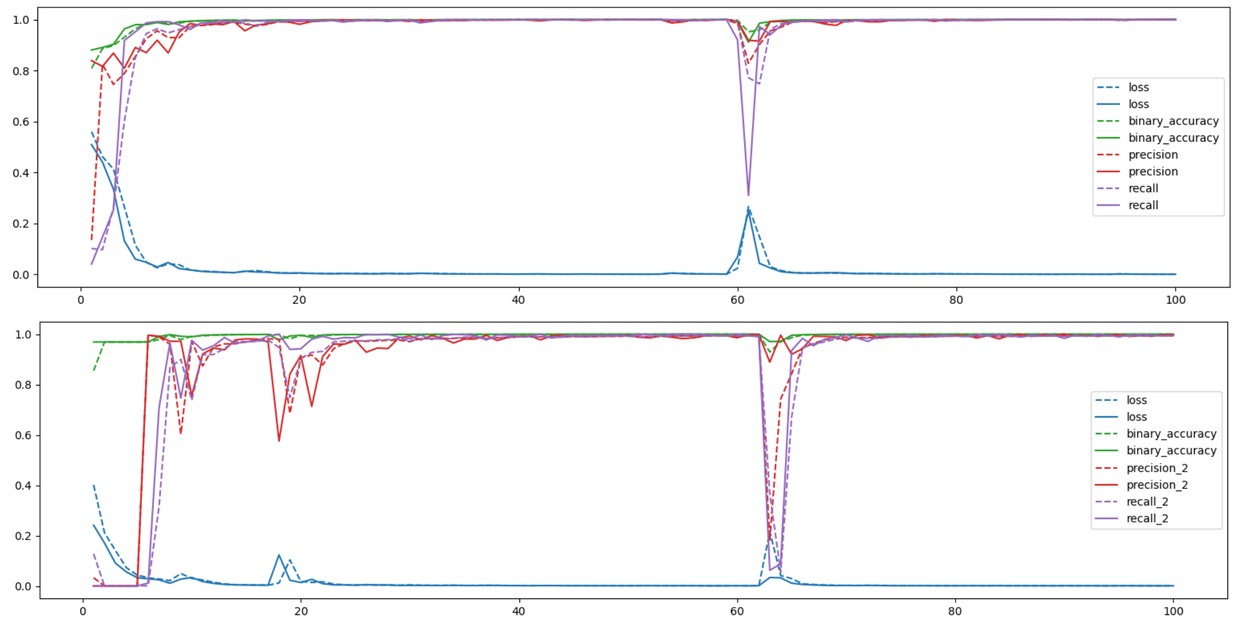

Fig. 4: *Training of U-Net of 32 filters with images of 128x128 and with images of 256x256. Observe that at first, the second network predicted all zeros. Dashed lines are evaluations in training set.*

a dataset of different pictures of a ball to give more variability to the object, but for that kind of problems state of the art object detection networks like YOLO or R-CNN should be used. Another thing to consider is that it would be possible to create different kinds of ground truth masks. For example, the circle in the ground truth mask can be smaller than the size of the ball, we think that this can provide more accurate positioning.

Finally, the tooling used was not only useful for facilitating the solution of the problem itself, but also as an introduction to the software implementation problems related to Deep Learning. As future work it would we useful to study MLOps solutions that are being used in the industry.

References

[1] https://www.tensorflow.org/tutorials/images/cnn

[2] Long, Jonathan & Shelhamer, Evan & Darrell, Trevor. (2015). Fully convolutional networks for semantic segmentation. 3431-3440. 10.1109/CVPR.2015.7298965.

[3] https://colab.research.google.com/github/VidushiBhatia/U-Net-Implementation/blob/main/U_Net_for_Image_Segmentation_From_Scratch_Using_TensorFlow_v4.ipynb#scrollTo=fKGrAcaOEK4E

[4] A. Scott Fulkerson, CC BY-SA 4.0, via Wikimedia Commons

[5] https://pillow.readthedocs.io/en/stable/

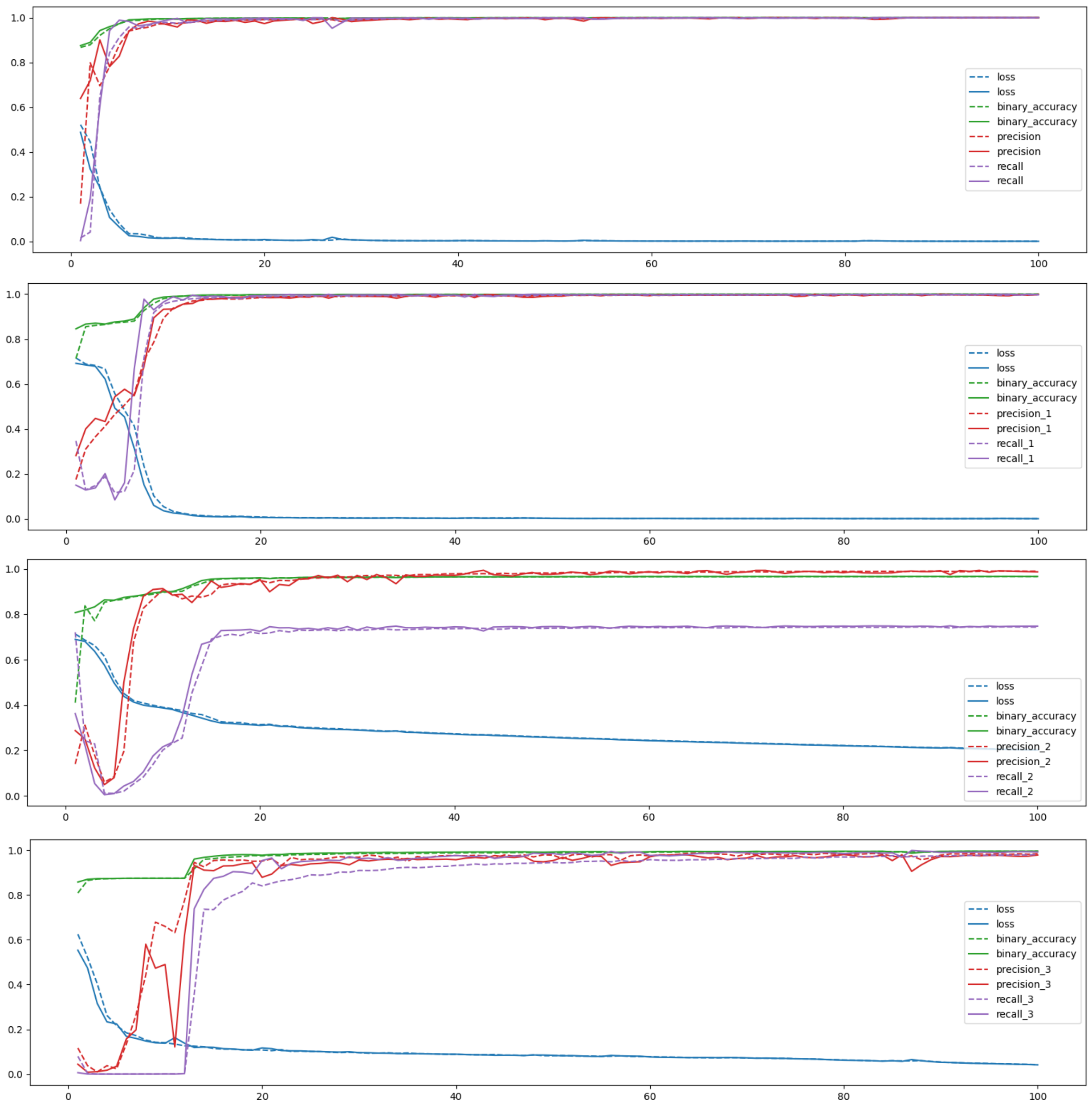

Fig. 5: *Training of U-Net with images of 128x128 and different size of filters: 16, 8, 4 and 2. Dashed lines are evaluations in training set.*

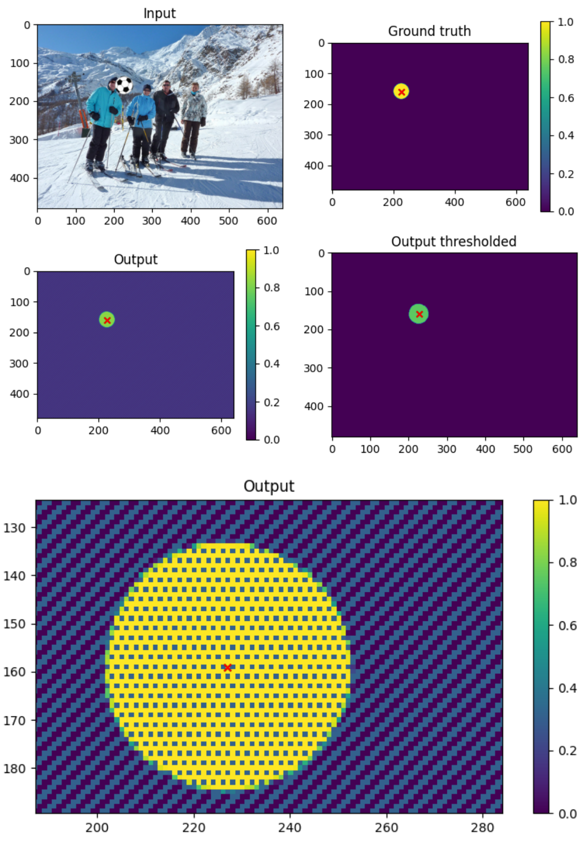

Fig. 6: *Case of the network of 4 filters*

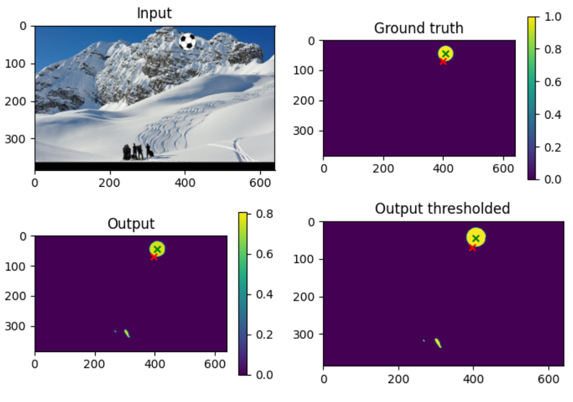

Fig. 7: *Case of the network of 2 filters*

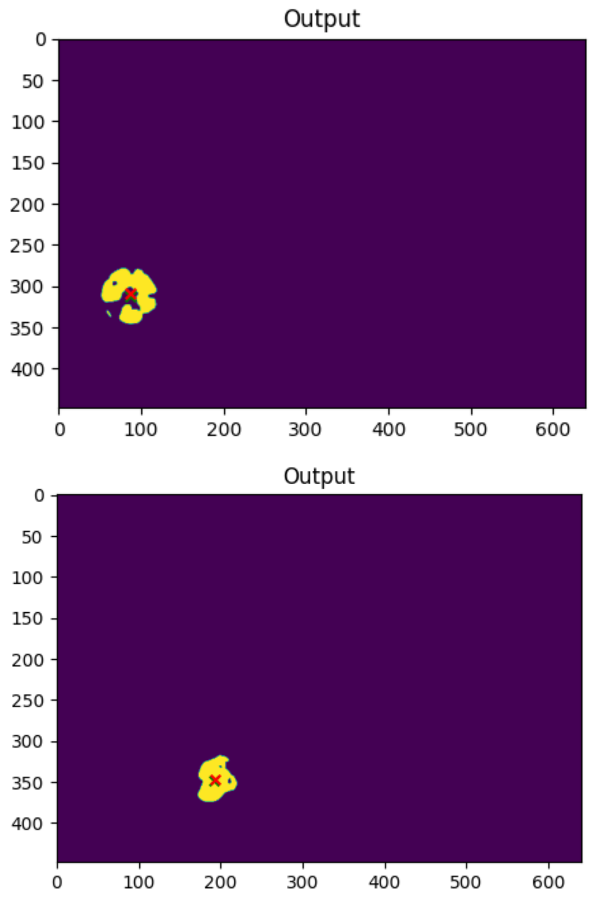

Fig. 8: *Predictions of balls of 75x75, and then predictions of normal sized rotated ball. Both done with networks trained in 50x50 unrotated balls.*

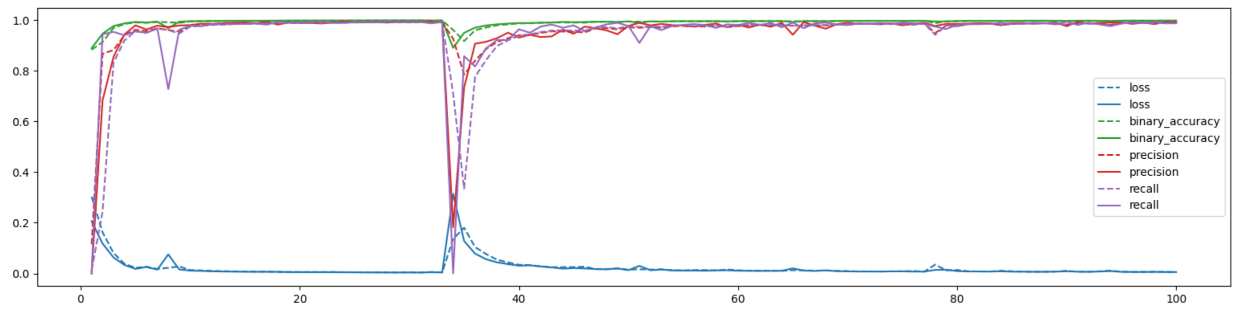

Fig. 9: *Training of U-net network with hard dataset. Using 100 epochs of 100 batches.*

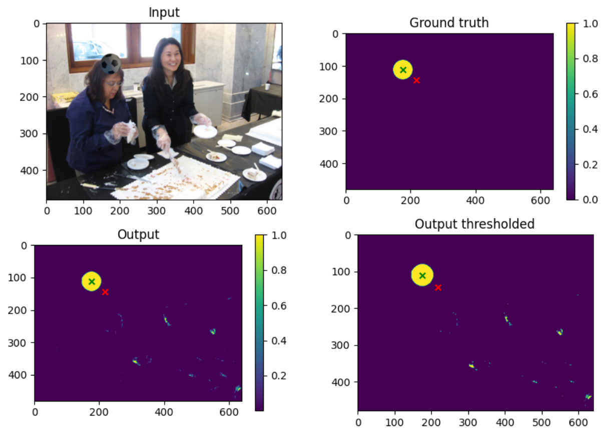

Fig. 10: *Example of a prediction of U-Net in the complex dataset. The results are satisfactory but smaller blobs have to be ignored.*

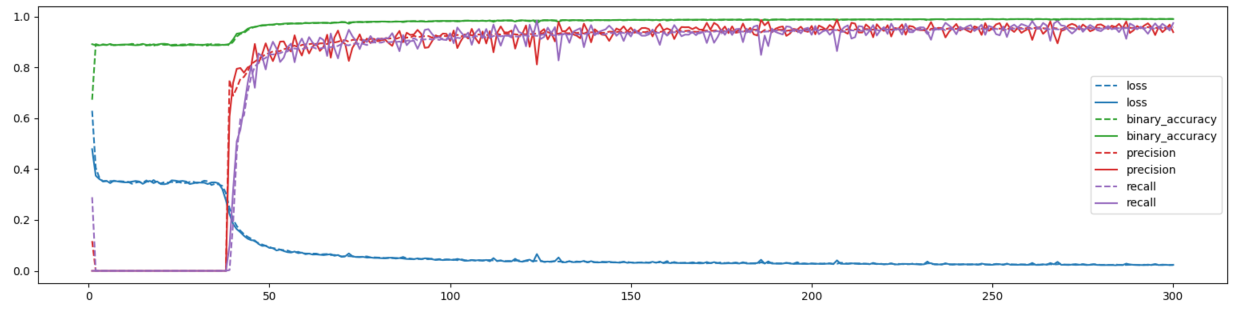

Fig. 11: *Training of FCN network with hard dataset. Using 300 epochs of 100 batches.*

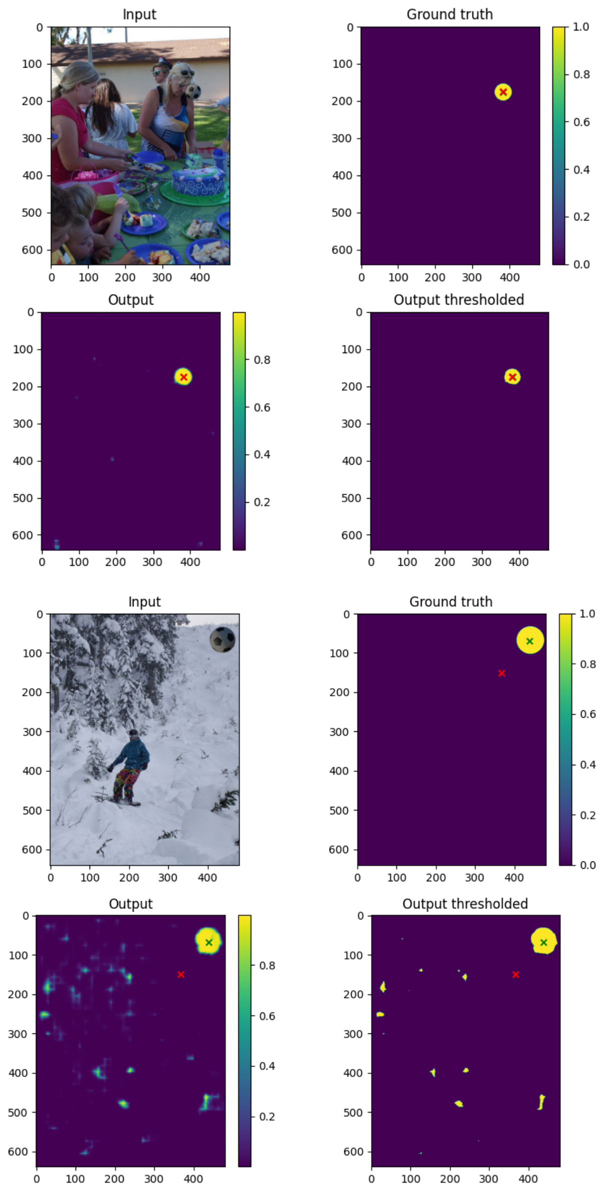

Fig. 12: *Examples of a predictions. The results are satisfactory but smaller blobs have to be ignored.*

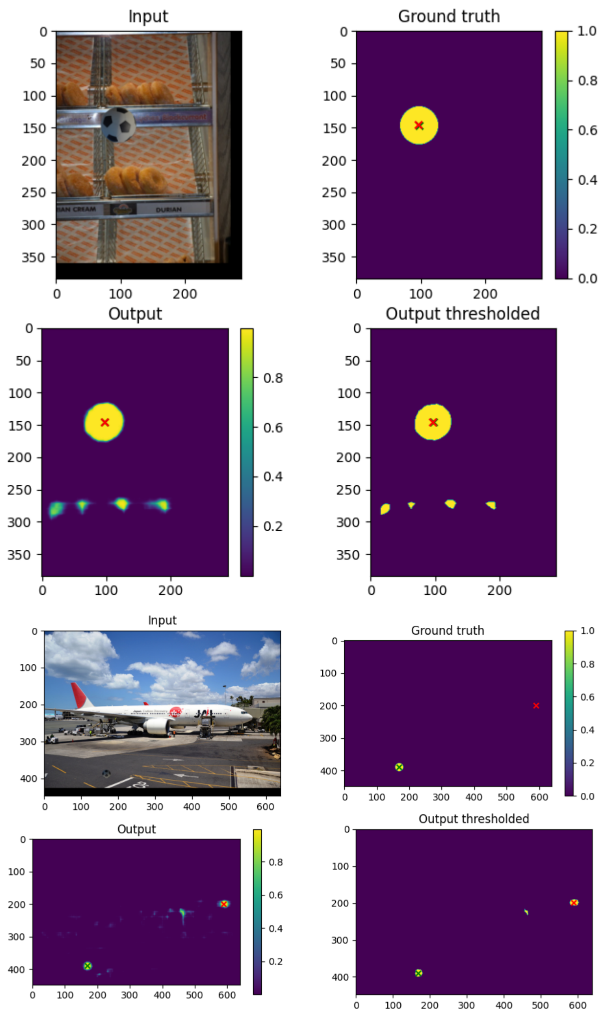

Fig. 13: *Examples of improved postprocessing. In the first case, the small blobs are correctly ignored but in the second one there was a false positive bigger than the real ball.*