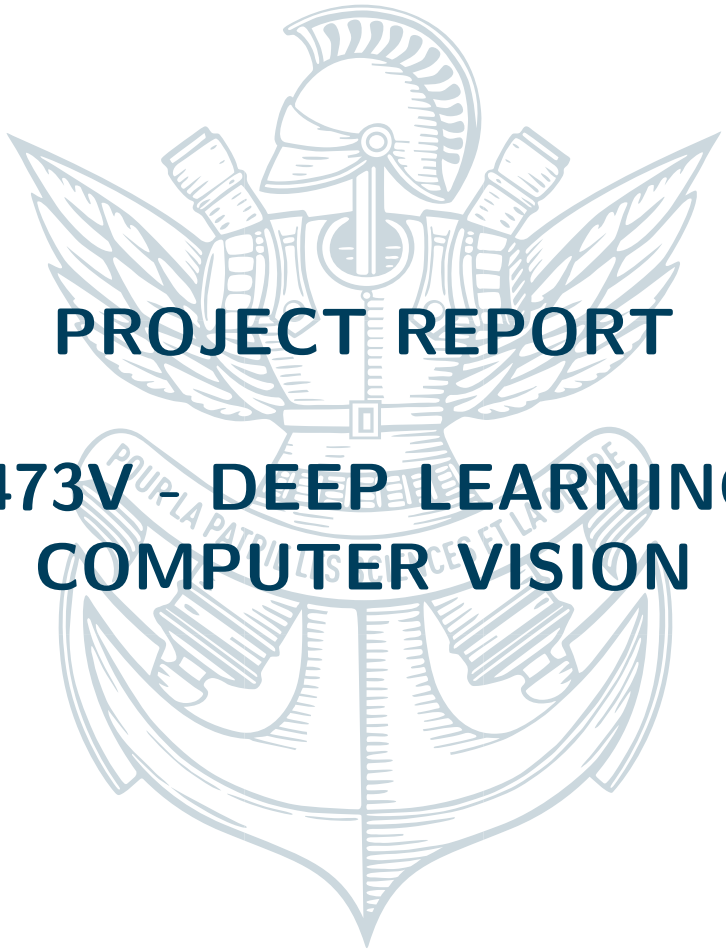


PROJECT REPORT

INF473V - DEEP LEARNING IN COMPUTER VISION

Airbus Ship Detection Project

Promotion X2019

Josselin Somerville Roberts
Valentin Descamps

1 Description of the problem

In this competition, it is required to locate ships in images, and put an aligned bounding box segment around the ships located. The boxes are stored using the **Run-Length Encoding format (RLE)** therefore they are not necessarily rectangular. The sample is large : a lot of images do not contain ships, while others may contain one or several ones. Ships can be of any size (from only a few pixels up to 200x200 pixels) and can be on any location over water (in the middle of the sea, next to a dock...)

Two files contain useful data : the training set contains nearly 200 000 images, while the testing set contains about 15 000 images.

Figure 1: An boat in plan sea, in a dock, a corrupted image

2 Description of our core work

2.1 Our method

First, we did some preliminary work on the data : eliminating the images without any ships in the training set (as they do not give relevant data, in fact a classifier will learn more from boxes that do not represent a ship next to a ship than from a box containing only water), coding functions to display the shapes, computing a rectangular box containing a given shape, to crop an image to a given box...

Once we had prepared the data, we worked on our predictor. The predictor is composed of two parts :

- A **FasterRCNN** Network responsible of detecting the rectangular bounding boxes of the boats
- A shape detector responsible of finding the exact pixels of the boat from a bounding box of a boat

Then to evaluate our predictor we have to :

- Predict and detect boat shapes in an image
- Compute the IOU of the rectangular bounding boxes computed by FasterRCNN with the ground truth boxes to assign each prediction to a ground truth boat

- Compute the number of correctly detected pixels (on the exact shapes and not the bounding boxes)
- Compute various metrics (f-score, precision, score, ...)
- Visualize a few examples

This allows us to check many metrics :

- the box detection score and the IOU for FasterRCNN as well as the classifier score.
- the precision of the shape detector (precision, f-score, fallout, ...)

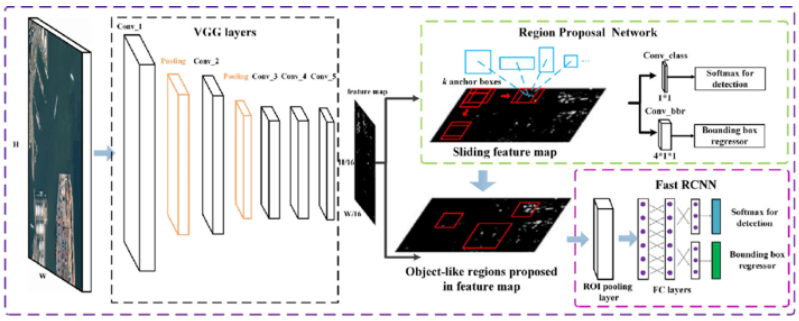
2.2 Our code architecture

Our notebook is divided into the following building blocks :

- Visualising the data
- Building a predictor
- Comparing a few shape detectors
- Conclusion

2.3 FasterRCNN

Our neural network is based on Faster R-CNN.



Input	Operator
$224^2 \times 3$	conv2d
$112^2 \times 32$	bottleneck
$112^2 \times 16$	bottleneck
$56^2 \times 24$	bottleneck
$28^2 \times 32$	bottleneck
$14^2 \times 64$	bottleneck
$14^2 \times 96$	bottleneck
$7^2 \times 160$	bottleneck
$7^2 \times 320$	conv2d 1x1
$7^2 \times 1280$	avgpool 7x7
$1 \times 1 \times 1280$	conv2d 1x1

Figure 2: The architecture of MobileNetv2

Faster R-CNN consists of multiple networks :

- First, there is a backbone : MobileNet v2. Here we have many convolutions layers with weights pretrained (otherwise the training would never end) to extract features from our images
- Then we have an **anchor generator** : this simply returns all the possibles boxes (before deformation) in an image. We will be using the standard AnchorGenerator provided by PyTorch. Since we have boats with shapes that vary a lot we have to make sure that our anchor generator can provide both small and big bounding boxes. In reality it seems like boxes of size 512 might not be relevant here but since there only a few of them we decided to keep them just in case (if we want to zoom in an image and then resize it)

- Then we have a **Region Proposal Network** : This is simply two dense layers with 1024 hidden neurons. This network will predict offsets and distortions to apply to each bounding boxes generated by the anchor generator.
- After that we have a **RoI pooling Layer** : This is simply a pooling layer with a pool size depending on the input size. This permits to feed the image cropped to the bounding boxes generated to an output of constant size (which is required for the next step)
- Finally we have a **classifier** : This is simply two dense layers with 1024 hidden neurons. We only have two classes here : ship or background.

2.4 Comparing a few shape detectors

We worked on images from the training set to compare a given plain image with the pixels assimilated to a boat, that we tried to automatically detect using convolutions. The main idea was, after cropping the image to a given bounding box centered on a boat, to use the canny function from cv2. This allowed us to get the outline of a boat, with could then be exploited.

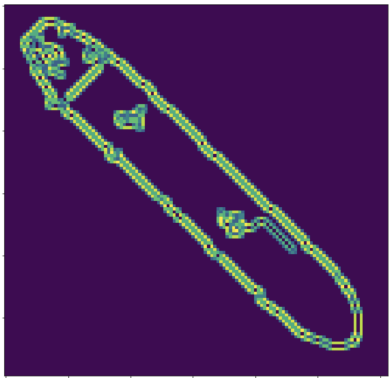


Figure 3: An example of the result obtained with cv2.canny

The general idea was then to "fill" the outlines to get two types of pixels : black ones outside the boat and white ones inside the boat. The difficulty was that outlines were not often as clear as the image above, with sometimes big gaps or stray pixels in the sea.

We tried four strategies :

- Coding a function from stratch, that fills in the outlines. The main idea was to study the image line by line and, for each line, coloring the pixels between the white one the most to the left and the white one the most to the right. To solve the problem of stray pixels, we calculated averages of distances between the right and the left pixels (as a boat often has a constant width) to exclude pixels that are too far away.
- Using a convolution with a particular kernel to fill in the gaps in the outlines, then using the function watershed from morphology in the skimage.
- Using color histogram threshold to keep only the pixels above a given value. Then a morphology function is applied, which gives the rough form of the boat. To fill in the gap we have two

different ways to do : either doing a linear regression on the top line and bottom line of the boat shape obtained, and filing the pixels between these two lines to obtain a rectangle

- Either filling in the gaps by focusing on the top pixel on abscissa x, the top pixel on abscissa x+40, and filling all the pixels between these two (and the ones under)

3 Results

3.1 Automatically detecting the pixels of a boat

The first strategy gave us encouraging results as most boats from images we tested in the training set were successfully colored. Yet the coloration was partial most of the time, so the function had to be ameliorated.

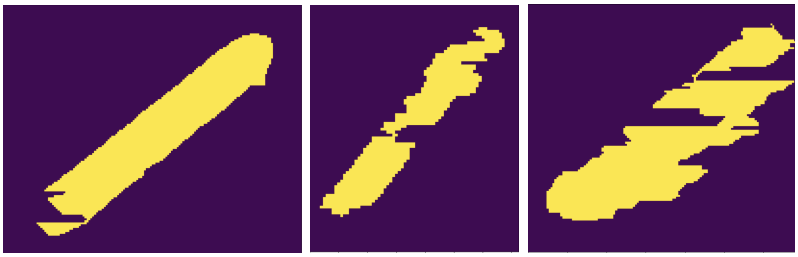


Figure 4: A boat successfully filled with our function "from scratch", one a bit less pretty, one really less pretty

One of the problems we faced was to separate boats next to one another, as we can see on the last image above.

The second function gave us nice images and showed us that convoluting was useful to complete outlines containing gaps

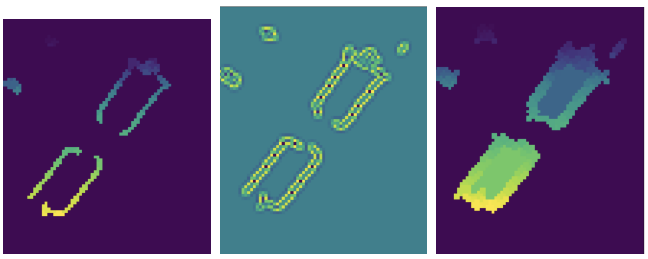


Figure 5: A boat detected with canny, applying watershed on this boat, a boat detected with canny then convoluted with a chosen kernel, applying watershed on this boat

One of the issue we faced on these two functions was the values to chose for the thresholds in cv2.canny : to get some good results, we had to adapt these parameters for each image. Yet we had some good results on average with th1=200 and th2=100. These were the values used for the images above.

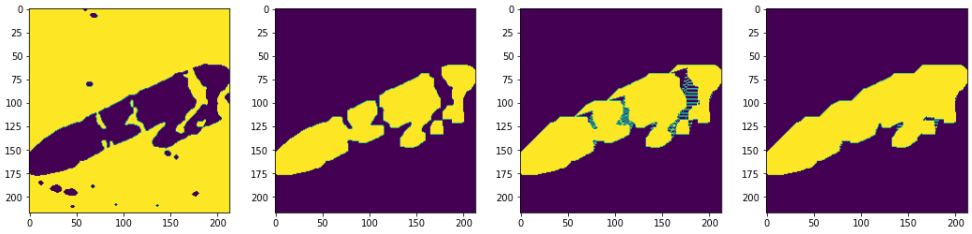


Figure 6: Strategy 3 : gaussian noise and threshold on the image of a boat, morphologic outline detection, adding lines to fill the gaps, filling the gaps

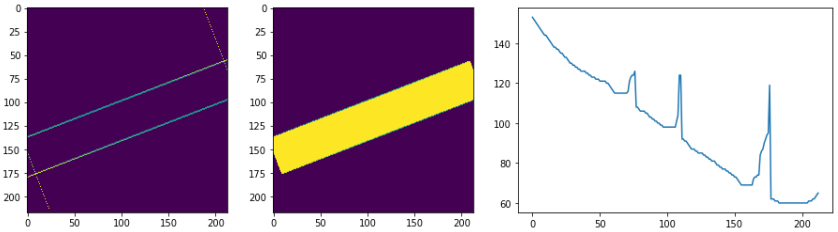


Figure 7: Strategy 4 : linear regression on the edges of the boat, filling between the lines, graph of the linear regression on the top of a boat

The results of strategies 3 and 4 are presented above. They are very much acceptable these are the ones we used for the final comparison. The graph presented shows us that the outlines detected are mainly linear apart from some gaps that we want to fill

3.2 The neural network

Here are the results obtained with our neural network :

On the first two images following, our neural network detected the boats successfully. On the third one : the boats were detected by Faster R-CNN but not with our outline detection. On the fourth one : the boats was detected by Faster R-CNN and by our outline detection, but it merged the ship with some parts of the dock. On the fifth one : three boats closed to one another are merged into one shape On the sixth one : all the shapes are detected but some are completely irrelevant

Overall the training set, after two epochs, we obtained the following results for Faster R-CNN over 200 validation pictures containing ships :

- An average detecting rate of 29%
- An average score of 92% for the class of detected boats
- An average IoU of 81% for detected boats

For our segmentation architecture, we obtained the following results :

- For the 1st strategy, we detected 22% of the pixels
- For the 2nd strategy, with watershed, we detected 29% of the pixels

Figure 8: Boats detected by our neural network

- For the 3rd strategy, we obtained 9% of the pixels
- For the 4th strategy, using linear regression, we detected only 10% of the pixels

The values seem very low but it can be seen on the images that a boat is either well detected or not detected at all, which explains a lot of pixels not taken into account in the percentages above.

The results may seem a bit disappointing, but we had a lot of difficulties training the network on kaggle : we found out after a lots of days struggling that some images were corrupted, and the images had presented in a dataloader with very specific instructions. What's more, the file regularly crashed for no reasons. At the end of the day, we only managed to do two epochs. Yet the results seem promising, and we will try to compute our neural network again for tomorrow.

4 Conclusion

To conclude, after some times of adaptation to understand the way images had to be dealt with, to code basic and extra functions, we managed to implement a neural network that gave promising results to successfully detect boats on a given image.

Predictably, our network works much better on images in plain sea, yet it gave promising results on some images of docks. What's more, our network works quite well on big boats, yet it struggles with smaller boats which are much more frequent on the training set. This is an explanation to the particularly low results obtained.

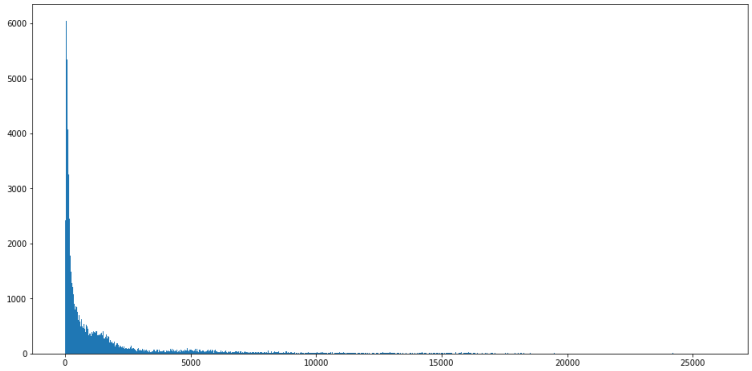


Figure 9: Histogram showing the areas of the boats in the training set

We could think of several way to improve our program :

- First of all, it needs much more training. We could have done it in a much better way if we hadn't encountered so many issues on kaggle. Maybe we could have used different classes adapted to the size of the boats.
- We could think about using non-pretrained neural networks. Yet we didn't think it was a priority as we had already done such a thing on TD7
- We explored alternative way to train our neural network, for instance using semantic segmentation. Yet, we thought it was adapted to a whole new projet.

On the whole, studying on this project was a very enriching experience. It was also a good rewarding work, as it is very pleasant to see a network that we coded in large parts giving promising results on the testing set.