Logo detection in a video

Benoit Audigier

November 2, 2018

1 Introduction

The aim of the project is to find logos of a Sysnav company in a video. Deep learning and computer vision are used for this matter.

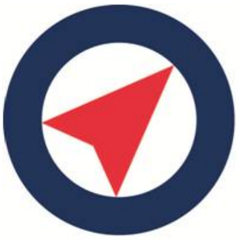


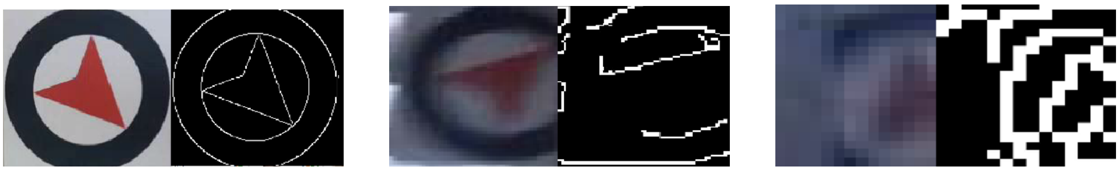
Figure 1: Sysnav logo

2 First Approach

The first approach consists in using computer vision with the very specific shape of the logo. To that extent, an edge detection using the OpenCV library is used. Every frame is converted in black and white, blurred and the gradient of the intensity of the pixel is analyzed to detect the edges. The Canny Edge Detection [1] implemented by OpenCV is used.

However, the level of maxVal and minVal are not that easy to determine. Those two values represent the thresholds that determine if the gradient is high enough or not high enough to be an edge. If the value is in between, the connectivity is taken into account. Indeed, the distribution of the gradient really depends on the images of the video, very different following the movement of the camera. That's why no efficient way of calibrating these values was determined; the images are often way too blurry because of the movement of the camera or the bad luminosity (see 2). Calibrating the values on the level of luminosity of the image was not efficient: the distribution was too different from one frame to another to correctly detect the edges - especially on the blurry images.

Detecting the circles inside the picture reveals to be too difficult to calibrate; a first focus on where to look before using computer vision is required for proper results. To that extent, deep learning is used.



a. The edges are clear, shapes are easy to detect.

b. The edges less clear, the background interferes

c. The edges are too blurry, the quality is not good enough, shapes are not perceptible.

Figure 2: Edge detection on several crops of images more or less blurred and small with the same thresholds.

3 Deep learning

The deep learning approach relies on the use of a neural network already trained and adapted from the ImageNet and COCO classification problem, as described in the documentation of Google's Object Detection API [2]. The idea is to retrain it with the local problem of our logo detection, and apply it frame by frame to detect boxes where the logo is likely to be.

The pros of that method is that it doesn't rely exactly on edge detection; it's more subtle, the features extracted from the frames are more diverse and more elaborate. It would also be adaptable to any object recognition, not only the logo problem, and this quite easily.

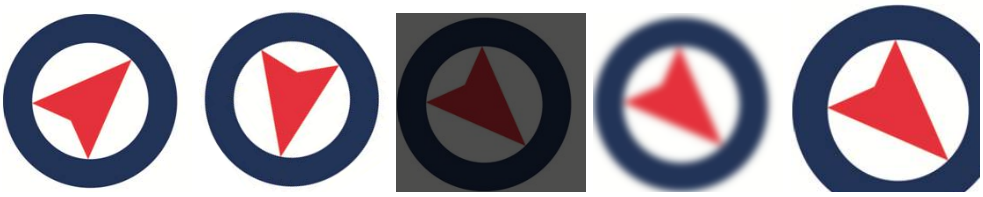
The cons are the fact that the model is very hard to explain; it's not trivial to determine why a box has been selected and one has not. This is a recurring problem with deep learning, even though a recent article on random forest applied to meta level training dataset is giving clues on interpretability of hidden layers of a Deep CNN network [7]. Furthermore, the computation is bigger than with a simple edge detection. The training part can also be problematic; this model has been trained during 12 hours on a CPU but could give better results with a lower training rate/more computation power (no GPU available here).

3.1 Dataset

3.1.1 Data augmentation

The first thing to consider is the fact that a dataset to train the neural network. For this, data augmentation is used. A short video pretty clear of the logo is first shot, in order to have labeled frames zoomed on the logo. Then, several methods are used to have a dataset as diverse as possible. The dataset is not modified directly, but on the fly. The images, when regrouped in batches are applied some random transformation. The efficiency of simple techniques such as the one described in the section are still quite efficient and relevant [6]

By using enough batches, we can make sure that the probability of using all the possible combinations is pretty high. For example, if we apply 2 transformation, let's call A_i the event ”the possibility i did not happen”, $\forall i \in \llbracket 1 \,;\, n \rrbracket$ (with our configuration, there are only $n = 4$ possibilities). We denote p_i the probability of A_i (with our configuration, if we use the same probability $q = .5$, we have $\forall i \in \llbracket 1 \,;\, n \rrbracket$, $p_i = .75$). Then, for a batch of size m we can the probability of not



a. Original b. Rotation c. Luminosity d. Blurred e. Cropped

Figure 3: Different transformations. They are not combined in this sample, which is the case in reality during the training.

having used all the possibilities:

$$\begin{aligned}\Pr(\cap_{i=1}^{n}\overline{A_i}) &= 1 - \Pr(\cup_{i=1}^{n} A_i) \\ &\geq 1 - \sum_{i=1}^{n} \Pr(A_i) \\ &\geq 1 - \sum_{i=1}^{n} p_i^m\end{aligned}$$

which is around 98% for $m = 20$ in this example. This is of course a simplistic situation. In the real use, the number of batch is very high compared to the value of each p_i, enough for this probability to remain in the same area - and be sure that the data augmentation was entirely exploited.

The transformations (see 3) are used to make sure that the neural network has seen as many real life cases as possible. Here are what is used:

- Random crops. The new image must remain big enough to have most of the logo on the frame. This simulates the logos that are partially cropped in the video.
- Random blurs. This simulates the movements of the camera.
- Random luminosity adjustments (brighter and darker).
- Random rotations, random horizontal and vertical flips.
- Random resizing.

3.1.2 Train, validation and test set

The dataset is then separated between a training set (70%), a validation test(20%) and a test set (10%). The later is not touch during any part of the training, and used to judge the performances after training to make sure we're not overfitting. We move then forward to the training phase.

3.2 Training

The neural network used is a NaSNet architecture [8]. All the layers but the latest one is frozen, and the last one is replaced by a dense layer with one category output with a sigmoid activation function and a .5 threshold. Weight decay (0.9) is added to avoid overfitting; no dropout is used. A Nadam optimizer (Adam with a momentum component [3]), with a learning rate at 0.001, and a constant for numerical stability $\epsilon = 1$.

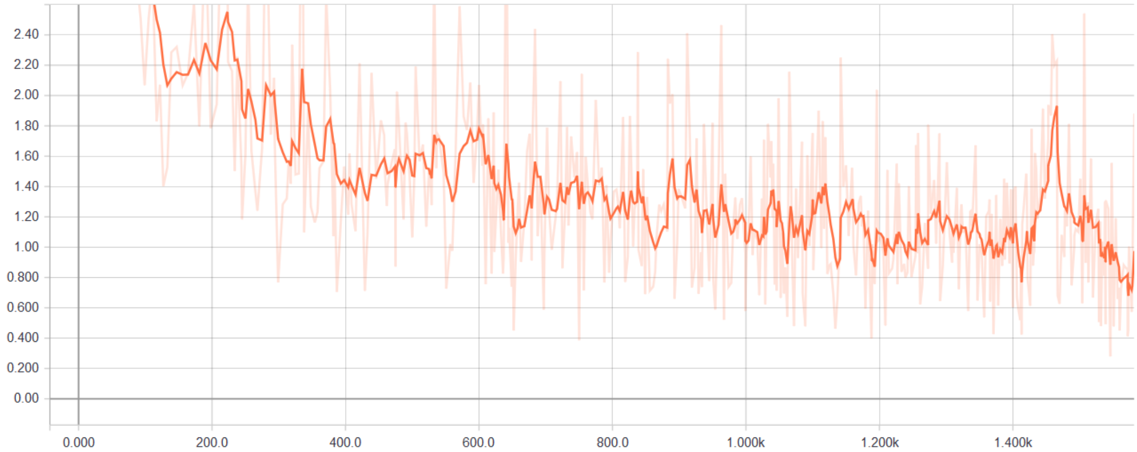


Figure 4: Loss evolution during the training. We can see that the training is not complete yet after less than 2,000 epochs.

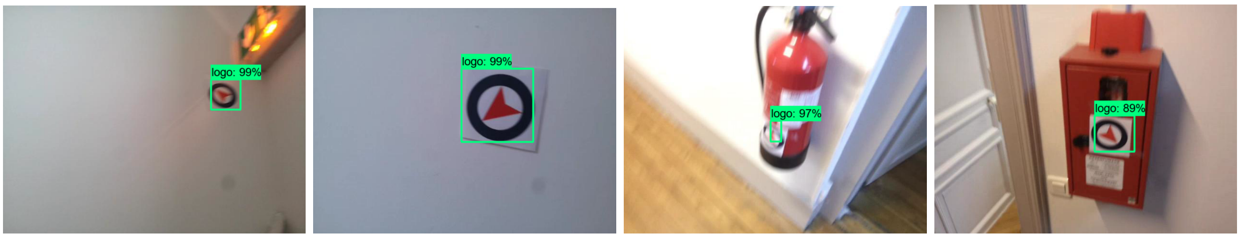


Figure 5: Sucessful predictions on test images.

The training phase is performed on a CPU for 12 hours. This is definitely not optimal. A smaller or decreasing learning rate could give better results but requires too much computer power. The parameters could have been optimized in a better way but due to time and computation power constraint, the values of the literature have been used.

3.2.1 Results

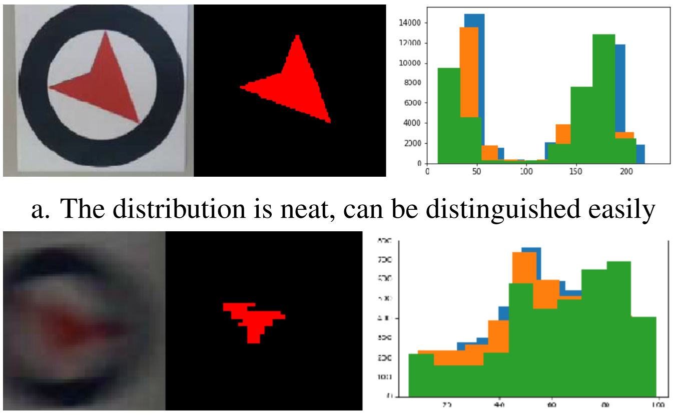
The results from this methods are mixed. With this learning rate, we barely reach a loss below 1 (see 4), and the training could be ran for more epochs (around 10,000 would be better). That said, it performs better than the simple edge/circle detection previously used (see 5). It seems not to be that much bothered by the blurriness/bad luminosity of a few frames. There are however too many false positive that need to be reduced using another independent method. This is why another filter is added to the process.

3.3 Back to computer vision

The technics used are design in order to detect false positives from true positives. The edge detection and shape recognition cannot be used since blurry images may be real logos. That's why others arguments are used.

3.3.1 Format

This one is pretty straightforward; the logos should be a square or not too far from it depending on the angle of the camera. this allows to remove a few false positive that have a ratio height/width



a. The distribution is neat, can be distinguished easily

b. The distribution is more mixed; the amount of red remains effective.

Figure 6: Images with detection of red and the distribution of the colors.

above 2 or bellow $1/2$. This filter is quite fast to apply (the complexity is $\mathcal{O}(1)$)

3.3.2 Color distribution

One thing that hasn't been used so far is the color distribution. Indeed, all the logo images should have - if the image is not too blurry - a pic in the color distribution in blue, white and red. However, this is not always the case, especially when the camera is moving too fast and the luminosity is not good enough.

However, it is possible to try to detect the amount of red in the picture, with what is considered red quite permissive ($red > 40$ and $red > \max(2 * green, 2 * blue)$). Indeed, a positive image should have at least a little bit of red, but must not be more than a 25% of the picture (number determined by using positive and negative samples, see 6).

4 Next steps

4.1 Tracking

The time factor is not used here. The frames are treated in an independent way, without any correlation nor order. A tracking algorithm could remedy this problem, such as using a K Nearest Neighbours Kalman filter [4]. This method would use an already detected object, and try to follow the evolution from one frame to another. This is not trivial, since the camera is moving, the blurriness is definitely not helping the case.

The study of the trajectory could also be something used. Indeed, if we detect an object for a few frames, we can have an idea of where to look for coming up frame, and feeding the network with a restricted zone to identify the box (the normalization would be more effective that way). Extrapolating the position from a linear regression of the latest positions and cropping the image with a slightly bigger size than boxes previously detected could refine the new box detection.

4.2 Offline Treatment

If the treatment is done offline (or with a reasonable delay), knowing the ”future” can really be useful in the detection of false positive and false negative. Indeed, an object not detected in between two frames where the object is can be recovered afterwards. The other way around, an object detected only for one frame with no continuity could be removed.

4.3 Real time treatment

The model used here does not allow the real time processing, at least on CPU. A few amelioration could improve this, such as jumping a few frames and extrapolating the position of the logos in the middle. Otherwise, lighter network could be used, such as the YOLO [5] have better performances (reaching 45fps or 155fps for the light version); this is however always a trade off between speed and accuracy.

References

[1] John Canny. *A Computational Approach to Edge Detection*. 1986.

[2] Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K. *Speed/accuracy trade-offs for modern convolutional object detectors*. 2017.

[3] Timothy Dozat. *Incorporating Nesterov Momentum into Adam*. 2015.

[4] Dan Iter, Jonathan Kuck, Philip Zhuang. *Target Tracking with Kalman Filtering, KNN and LSTMs*. 2016.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[6] Jason Wang; Luis Perez. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017.

[7] Xuan Liu, Xiaoguang Wang, Stan Matwin. *Interpretable Deep Convolutional Neural Networks via Meta-learning*. 2018.

[8] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le. *Learning Transferable Architectures for Scalable Image Recognition*. 2018.