Study on State-of-Art Object Detection Based on Deep Learning

Francisco Serodio
VCI - MIEET
University of Aveiro
Email: francisco.serodio@ua.pt

Pedro Figueiredo
VCI - MIEET
University of Aveiro
Email: pfsf@ua.pt

Alexandre Figueiredo
VCI - MIEET
University of Aveiro
Email: alex.figueiredo@ua.pt

***Abstract*—In the last few years, a substantial progress in the field of computer vision has been witnessed, specially in the object detection field, mainly due to the usage of neural networks. In this paper we'll be discussing this recent evolution, as well as briefly explaining some of the algorithms that were responsible for this progress and the current state of the art when it comes to object detection.**

I. Introduction

Neural networks, despite their recent rising popularity, are not a new concept. Even CNNs, or convolution neural networks, which are shown in this work, were already being used in 1998 to detect digits [1]. But what was the reason for this recent popularity then? AlexNet. Back in 2010 the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* was launched and in just 2 years a major revolution in the field of computer vision was on. Why's that? In 2012 researchers Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton published a paper [2] that described a model which used a CNN (*AlexNet*) to classify images - this architecture won the competition with a 15.3% Top-5 error rate (rate of not finding the correct label when classifying images amidst its top-5 predictions) and the next best being an algorithm that scored 26.2% - in other words, it was groundbreaking and it completely changed the way image classification was made.

A. Brief Introduction to Deep Neural Networks

Within the domain of Machine Learning, there is a sub-field called Deep Learning which deals with algorithms that are inspired by the way our brain works and learns - and thus mimicking the brain's network of neurons - these are called Artificial Neural Networks. These systems work by having the input, an array of elements, go through a certain number of hidden layers that apply various operations - usually a multiplication by a weight - to each one of them, until they reach the output layer - this process is called forward passing (Fig. 1). The output that has the highest value is the result of the process, sometimes normalized to 1 or 0 with a *sigmoid* function. Afterwards the result is compared with the expected one, obtaining the Loss Function - the lower its value (error) is, the more robust the network will be - and thus, the objective is to minimize this function. For this, a technique called backwards propagation is used, which adjusts the weights of the hidden layers depending on the loss function values, in order to minimize the error. The systematic repetition of this process is referred to as "network training".

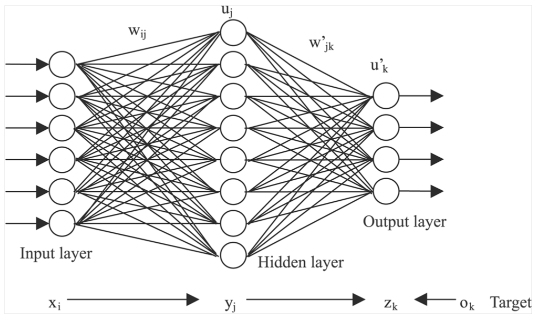

Fig. 1. Artificial Neural Network Architecture

B. Brief Introduction to Object Detection

The objective of the object detection networks is to obtain both the bounding boxes and the classes for the objects in the scene. Therefore, this two objectives can be evaluated separately: the location and the detection. To illustrate this problem, we can see in Fig. 2 how the algorithm is detecting the objects and classifying them. To measure how well or poorly a model is performing in these two tasks, a couple of metrics were created - mAP (Mean Average Precision) and IoU (Intersection over Union). The mean average precision (mAP) measures the accuracy of the detection and the intersection over union (IoU) measures how much area of a bounding box overlaps another one - usually overlap is considered good since it means it detected objects that are close to each other or mixed (e.g. man on a horse).

II. Methods

When it comes to the search strategy adopted to find the state of the art systems used in this field, we started by searching various digital library websites, such as IEEEXplore and Google Scholar, using keywords like "object detection", "deep learning" and "object detection using deep learning", and we found many different works related to this theme.

Afterwards we filtered these chronologically in order to show only the most recent ones, and we noticed that many of these papers were based on some algorithms like YOLO, SSD or R-CNN, amongst others. To get their original papers,

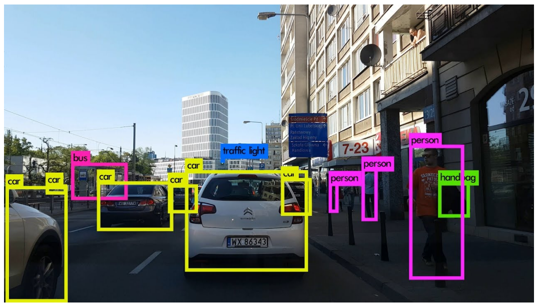

Fig. 2. Object Detection and Classification

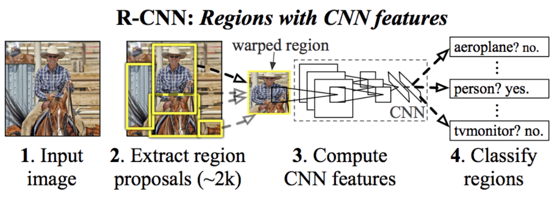

Fig. 3. R-CNN

we simply used Google and found out they were all on Arxiv open-source library. We read their papers and also googled more information about them, to then find some articles on Medium (Towards Data Science) explaining their evolution as well as, their pros and cons when compared to each other. Also alot of information regarding deep neural networks and few other details about object detection was also found on Medium.

III. Results

After our research, we concluded that the most relevant systems to discuss here were, as already mentioned above, YOLO, SSD and R-CNN, and their iterations over the last few of years:

A. R-CNN: Region-based Convolutional Neural Networks

R-CNN [3] is a special type of convolutional neural network whose purpose is object detection, classification and localization in images. So, the output is normally a set of bounding boxes matching the detected objects and the their respective classification.

The work flow of this method begins with a pre-train of the *CNN* followed by a selective search to extract 2000 category-independent region proposals, also called as *Regions of Interest* (*RoI*), per image. Non-maximum suppression helps to avoid repeated detection of the same instance in the image so that, afterwards, only the best bounding boxes remain and, by consequence, the other are ignored as they have large overlaps with the selected ones. Next, the region proposals are warped in order to have a fixed size required by the *CNN*. Following this, a fine-tuning of the CNN is made on previously warped regions for K+1 classes, where K is the number of different classes present in the selected regions and the extra class the background. This is due to the fact that most of the region proposals are just background (in this stage of tuning, the learning rate should be smaller).

Now, for every image region, the CNN extracts a feature vector that will be fed into a binary *SVM* (*Support-Vector Machine*), trained for each one of the classes, to classify the presence of objects.

After the explanation above its easy to understand that this method is slow and expensive, mainly because every image will have around 2000 region proposals, and every single one needs to be processed by a CNN in order to extract its features - the whole process involves the *CNN* for image classification and feature extraction, the top *SVM* classifier for identifying target objects and the regression model for tightening region bounding boxes.

To get rid of some of these problems, the method was improved over and over again throughout the years past its creation. Below we present the R-CNN iterations that solved some of the initial problems:

1) Fast R-CNN: This method [4] starts by pre-training a CNN on image classification tasks, like R-CNN does, but instead of feeding the 2000 region proposals of the image to the CNN, it only feeds the original input image so that a convolutional feature map is generated. This change allows computation sharing which speeds up R-CNN.

Afterwards, using the obtained convolutional feature map, the region proposals are identified, and warped into squares and replacing the last max pooling layer of the pre-trained CNN with a *RoI* pooling layer. The *RoI* pooling layer outputs fixed-length feature vectors so that it can be fed into a fully connected layer. From the *RoI* feature vector, the method uses a *softmax* layer not only to predict the class of the proposed region but also the offset values for the bounding box.

Thanks to this, Fast R-CNN is much faster than R-CNN in both training and testing time due to sharing computation that is possible in this approach. In other words, it's not necessary to feed 2000 region proposals to the convolutional neural network every time - the convolution operation is done only once per image and a feature map is generated from it.

However, this improvement of speed is not too substantial, because the regions selected are generated by another model, which takes a toll in terms of computation power and time.

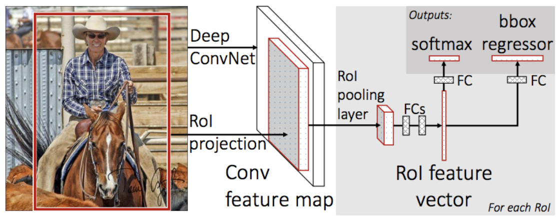

Fig. 4. Fast R-CNN

2) Faster R-CNN: R-CNN and *Fast R-CNN*, presented above, use selective search to find out the region proposal of a given image. This method [5] is slow and a time-consuming process that has a big impact in the performance of the network. In order to solve these problems, another variant of *R-CNN* called *Faster R-CNN* was developed. This method consists in a speedup solution of the *Fast R-CNN* where there is no selective search algorithm and the network itself learns the region proposals.

To achieve this, the region proposal algorithm is integrated into the *CNN* model with the intuit of creating only one, unified model composed by a *Region Proposal Network*, which is then fed into the *RoI* pooling layer like in the *Fast R-CNN* iteration.

Like in the methods presented previously, the image is provided as an input to a convolutional network (*Region Proposal Network*) which provides a convolutional feature map. To substitute the selective search algorithm, a separate network it's used and then the predicted region proposals are reshaped by a *RoI* pooling layer that also classifies the region proposals.

There's also another iteration made to R-CNN: Mask R-CNN. Without getting into much detail, Mask R-CNN is basically an extension over Faster R-CNN. Faster R-CNN predicts bounding boxes and Mask R-CNN essentially adds one more branch for predicting an object mask in parallel. This improves Faster R-CNN performance quite substantially, making this method compete with other state of the art models once again.

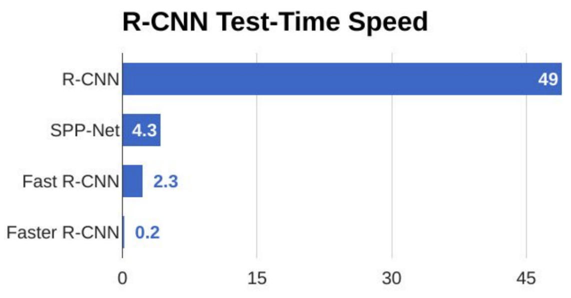

Fig. 5. Comparison of test-time speed of object detection algorithms

B. *SSD: Single Shot MultiBox Detector*

The SSD [6] method is based on a feed-forward convolutional network that creates a fixed-size collection of bounding boxes and detects object classes within those boxes.

As the name suggests, SSD only takes a single look at the image in order to analyze it - and does it using the VGG-16 architecture due to its high performance when it comes to image classification. Its location and detection is done in a single forward pass on the convolutional network, similar to the YOLO [7] model [III-C]. To understand how the *CNN* used in SSD works, we listed and briefly explained the main detection structures present in the network:

1) Multi-scale feature maps for detection: Using a convolutional feature layer atached to the end of the network, its possible to decrease the size of the image and detect classes in multiple scales. Each detection is different for each layer.

2) Convolutional predictors for detection: Using VGG-16 architecture, each feature can produce a fixed amount of predictions. In this detection method, a $3{*}3{*}nchannels$ kernel is used, which can detect the category or shape offset relative to the default bounding box coordinates.

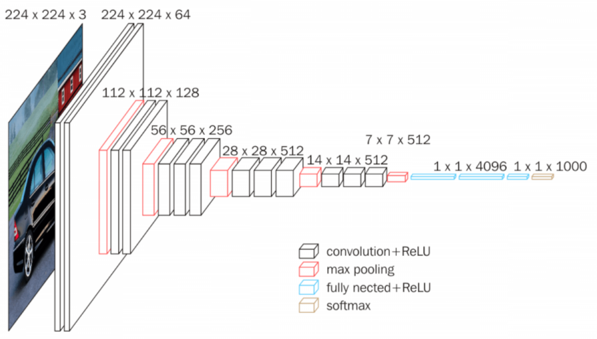

Fig. 6. A simple VGG-16 architecture

3) Default boxes and aspect ratio: A set of default bounding boxes are added to each feature map cell, predicting the offset relative to the default bounding box and category.

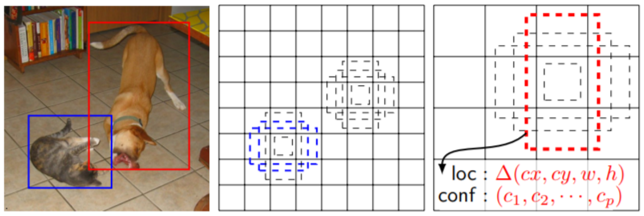
loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

Fig. 7. Example of a default box and aspect ratio prediction

4) Training the network: The main stand out of SSD training is the fact that it used ground truth information (labels) assigned to a specific detector in the outputs, [III-B1,III-B2,III-B3].

This said, it starts by matching the ground truth information to the default bounding box with the best IoU, starting at 0.5. This is a much better strategy then starting the prediction in random coordinates. However, having the best IoU isn't enough to predict correctly. Being a multibox method, it can calculate the overall objective loss, L, of the image by using the weighted sum:

$$L = \frac{L_{conf} + \alpha * L_{loc}}{N}$$

where N is the number of matched default bounding boxes, L_{conf} the confidence loss, α balancing contribuitor and L_{loc} the location loss.

But like the R-CNN methods, sometimes there's too many boxes in the region. For this the non-maximum suppression method is used - this means that boxes with a confidence loss and IoU lower than certain threshold are discarded. With SSD it's possible to get over 74% of mAP (depending on the dataset used).

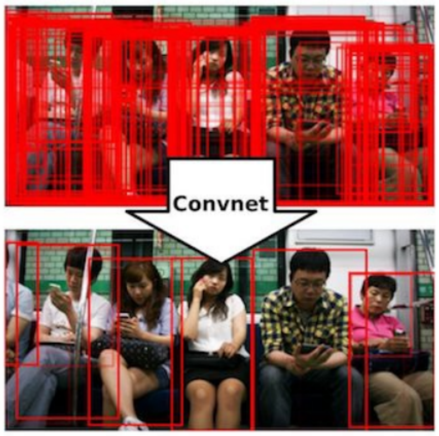

Fig. 8. Non-maximum suppression application example

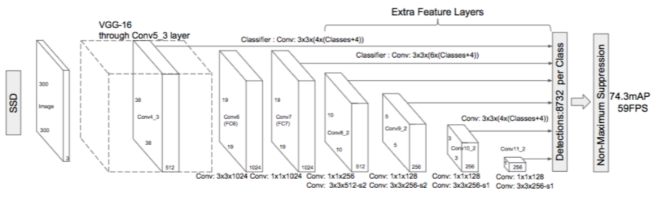

Fig. 9. Simplified SSD method of prediction

Another iteration of SDD was made called *DSSD : Deconvolutional Single Shot Detector* which solved some issues that the original model had, along with improving its performance. Due to the small input size, SSD does not work well on small objects. With deconvolution path, DSSD solves this problem.

C. YOLO: You Only Look Once

YOLO method is a single shot multibox detector much like the SSD method. Though they share very similarities, like taking a single shot of the image to analyze it, using convolutional anchor boxes (bounding boxes) and multi-scale training, they also use some different strategies to detect and classify. (Fig.2 is an YOLO output)

1) Dimension Clusters: instead of hand picking the default bounding box for each map, k-means clustering on the trained bounding boxes will automatically find a good default bounding box.

2) Direct location prediction: With SSD, using bounding boxes, the early iterations are unstable. To correct that, YOLO uses the direct location prediction. This means that the location prediction is constrained in a way that the parameterization its easier to learn, making the network more stable.

3) Batch Normalization: With this the convergence is improved and some forms of regularizations are eliminated.

Instead of using VGG or VGG-16, YOLO uses its own architecture - **Darknet-19**. This architecture is more faster than the the others but slightly less accurate. While VGG-16 requires 30.69 billion floating point operations for a single pass, the Darknet-19 only requires 5.58 billion. This makes the architecture faster. However VGG-16 have 90% accuracy and Darknet-19 have 88% accuracy. This makes Darknet-19 faster but less accurate.

There's also two more iterations of this method, *YOLO9000: Better, Faster, Stronger* and *YOLOv3*

IV. Discussion and Conclusion

To finalize, we can conclude that all of the algorithms discussed in this work have their pros and cons, and to make a direct comparison between each one of them isn't the most correct procedure to apply when analyzing these models. This is because most of the tests found on various papers (including their actual papers) and articles, are made in different settings, which leads to misleading results - even if they use the same dataset. Still we can say that some models perform better in some applications then others - this is the case with real-time applications where the FPS (Frame per Second) count has a higher priority then the mAP score for example (e.g. in a autonomous driving setting). Here the most appropriate model is probably YOLOv3, since it runs in 22 ms at 28.2 mAP, and as accurate as SSD but three times faster. The same can be said in cases where mAP is more important then the FPS count (e.g. when detecting cancer cells in a tissue sample).

There's many other algorithms that we didn't mention but that also compete or even surpass some of the works we talked about in this paper. The reason for only choosing three algorithms was mainly due to the time we had to research and to study each model, and so we had to make a choice on which ones to talk about. Although R-CNN and its iterations are already a little bit behind (not Mask R-CNN though) when compared to other models, we still chose to include them in our work because they had a big impact in the way these type of systems are evolving, and so we thought it was important to mention and to explain how their structure evolved.

In terms of limitations related to our work, we didn't feel like there was any major obstacle during our research for the state of the art in this field, although if we knew about Arxiv before, the time needed to complete our research would have been shortened by a significant amount. This is due to the fact that most, if not all, the algorithms discussed in this work have their papers stored in Arxiv, and are easily accessible with a simple search similar to the one we did on Google Scholar or IEEExplorer.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," pp. 2278–2324, Nov 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” pp. 1097–1105, 2012. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[3] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[4] R. Girshick, “Fast r-cnn,” in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: single shot multibox detector,” *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[7] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767