Aircraft Detection from satellite images using Single Shot Multibox Detector

Andrea Di Mauro
Polytechnic University of Turin
s288048@studenti.polito.it

Abstract

Object detection applied to satellite images is one of the most innovative technologies used in earth observation activities. The information obtained from satellite images with this technique can be used in various fields such as urban planning, road traffic control and agriculture. Among all the possible recognizable entities within a satellite image, airplanes also represented an interesting object of investigation for the purposes of air traffic control, route optimization and even safety reasons. Unlike other common object detection data, satellite images have usually larger dimensions and for this reason they have a much greater information density and therefore represent a more difficult challenge. The most common architectures for this task known in the literature can be easily used, with different approaches, results and performances. This paper deals with the Aircraft Detection task with a Single Shot Detector network.

1. Introduction

Recognition of aircraft within a satellite image is not a simple task due to the different weather conditions of the area captured by the image, the different forms that the aircraft themselves may have (military, civil, ultralight, cargo, etc..) as well as the different backgrounds in which the entity may find itself (airport, sea, desert, dismantling base etc..). For this reason, it is necessary to train one's architecture on a large set of images and above all containing many different situations within it. Another important factor at play for the performance of the task is the resolution of the input images and the size of the objects: different architectures that have been proposed tackle the problem in a different way even taking the original size of the image as input; the Single Shot Detector analyzed in this paper takes as input an image with dimensions of 300x300 and thanks to its structure it is able to recognize objects at different scales. The model outputs a fixed number of predictions in the form of horizontal Bounding Boxes which are then decoded and passed to an evaluation function to measure the performance obtained. The Single Shot Detector was built based on the original paper by Liu et al. [2] that proposed the model, but slightly modified based on the code found in a free available GitHub repository[1]. Due to the small size of the free datasets found online, the data augmentation phase was fundamental for training the network on a dataset that is not too poor in images.

2. Abbreviations

SSD - Single Shot (Multibox) Detector
AP – Average Precision
IoU – Intersection over Union
VGG-16 – Visual Geometry Group 16

3. Dataset and data augmentation

The network training was performed on a single dataset resulting from the mix of 3 different datasets downloaded for free from the web:

- Airbus Aircraft Detection dataset: https://www.kaggle.com/airbusgeo/airbus-aircrafts-sample-dataset
- CGI Planes dataset: https://www.kaggle.com/aceofspades914/cgi-planes-in-satellite-imagery-w-bboxes
- DOTA dataset: https://captain-whu.github.io/DOTA/dataset.html

The first and third datasets respectively contain satellite images related to aircraft detection and object detection challenges: their characteristic is to contain original images, not modified for the purpose, with depiction of scenes in different location and weather conditions. While the first contains only images with civilian transport aircraft, the third dataset also contains other models of civil and non-civil aircraft. Finally, the second dataset contains satellite images on which various figures representing the same aircraft model but with different inclination levels have been added with photo editing tools. The data to be taken as

[1] https://github.com/Socret360/object-detection-in-keras

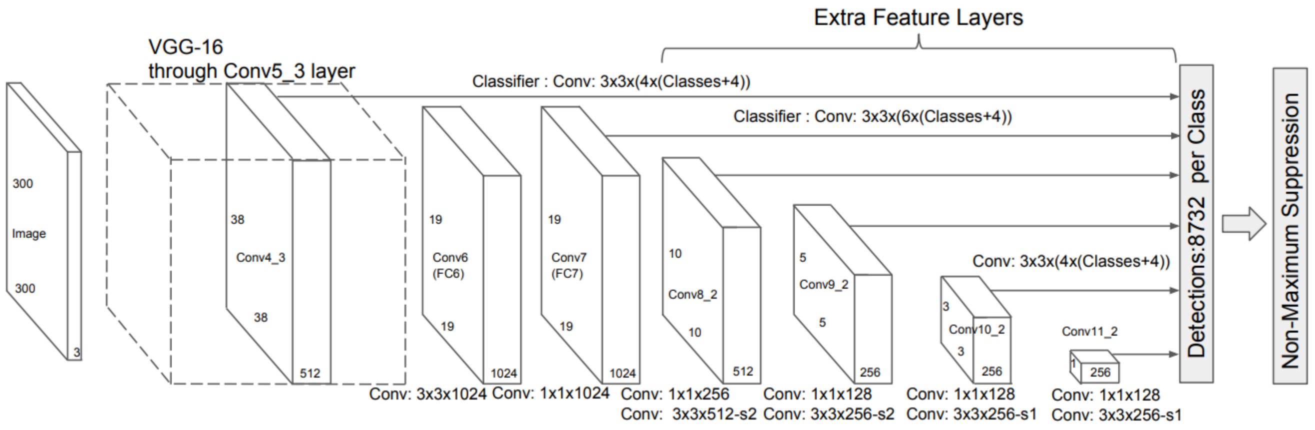

Figure 1: Single Shot Multibox Detector Architecture. Font: see reference **[2]**

input are the images and for each a file containing the coordinates of the horizontal bounding boxes and the label of the object depicted. For this task it was only necessary to consider objects classified as *airplane*; only in the third dataset was it necessary to carry out 2 operations of filtering: only the images containing at least one "plane" type label were considered and oriented bounding boxes were converted in horizontal ones (see next section). All this information was unfortunately encoded in the 3 datasets in different ways and formats. A network independent python script that uses *imgaug* [2] library was written specifically to standardize data in a single format, group the 3 datasets into one and perform data augmentation at the same time (also bounding boxes conversion for dataset 3 mentioned above). The dataset in its final format therefore consists of 3 folders (*train*, *validation* and *test*) each containing a JSON file with the same name as the folder in which it is contained. Each JSON file collects for each image the coordinates of the bounding boxes referred to the origin of the image (upper left corner) and its original dimensions. Apart from the second dataset which already presented a split in test, train and validation, the other datasets were treated as follows: considering the total number of images in each dataset, 30% was used for the test, 90% of the remaining 70% was used for training and the rest as validation. The final composition of the dataset is: 1554 training images, 424 validation images and 81 testing images.

4. Oriented Bounding Boxes conversion

In the DOTA dataset containing oriented bounding boxes it was necessary to implement a form of conversion in horizontal bounding boxes to make it uniform with the other data. In order to recreate a Bounding Box that entirely contained the identified object, the pair of points *(xmin, ymin)* and *(xmax, ymax)*, in red in the Figure 2, were considered to construct the respective horizontal Bounding Box.

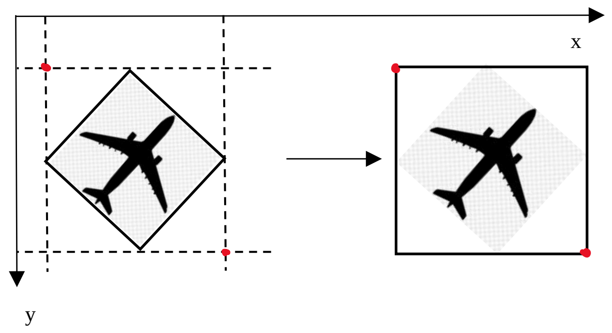

Figure 2: oriented bounding box conversion into horizontal

5. Approach

The approach initially used is inspired by the work done and illustrated by Shubhankar Rawat in his article [1] used for an Aircraft Detection task with excellent results. Among the many architectures available and known in the literature to be excellent with object detection activities (Fast RCNN, Faster RCNN, YOLO...) for this paper the SSD was chosen, not only for its performance but also for its very intuitive structure. After the development of the specific function for the creation of the dataset and the data augmentation, the next step was therefore to create an SSD model based on the original paper by Liu et al. [2] and then complete the work by adding the parts of the code relating to training and evaluation. The loss has in fact been implemented using the functions developed specifically for the use of an SSD with Keras in a GitHub repository[3]. It has been chosen to use the functions implemented in this repository to have an

[2] https://imgaug.readthedocs.io/en/latest/

[3] https://github.com/pierluigiferrari/ssd_keras

example of reference training[4]. The repository mentioned on the previous page, instead, was followed as a model for training, evaluation and setting of hyperparameters. The first training runs were performed taking into consideration the following hyperparameters: 50 epochs of 125 iterations each and an Adam optimizer set with an initial learning rate value of 0.001, epsilon 1e-8 and momentum 0.9. The use of an SSD model built from scratch based on the original paper was very useful from a formative point of view but at the same time proved to be a real challenge from the training point of view: after solving an initial incompatible shapes error due to the use of Keras' default VGG-16 network as backbone, a further problem to be addressed was that of the divergence of the validation loss. The problem was solved by introducing in the convolutional layers a kernel regularization of type L2 with a lambda factor of 0.0005. In fact, finally, the model was very similar to the one built in the first repository, so it was decided to use that directly.

6. Model

The original SSD architecture as shown in Figure 1 is composed by two main components: a backbone model and SSD head. Backbone model usually is a pre-trained image classification network as a feature extractor. For this study has been used a VGG-16 trained on ImageNet from which the final fully connected classification layer has been removed. The SSD head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the final layers' activations. In an SSD there are two key components to obtain the final predictions: grid detection and default boxes. The feature maps are produced by the convolutional layers: *conv4_3*, *fc7*, *conv8_2*, *conv9_2*, *conv10_2* and *conv11_2* where the $n \times n$ dimensions of each layer represent the dimensions of a grid. To produce detections of different shapes to each grid cell are assigned N numbers of default boxes with its center placed at a certain offset from the grid cell (usually center). In SSD, the width and height of each default box can be retrieved through two values: the aspect ratio of the box and the scale of the box with respect to the input image size. Finally, the number of predictions is fixed at *(total_default_boxes, num_classes + 1 + 4 + 8)* where *total_default_boxes* is the total number of default boxes across all feature maps and *num_classes* are the number of classes.

7. Loss

SSD combines a regression loss and a classification loss with an α value as a scaling factor for the localization loss:

(1)

$$L(x,c,l,g) = \frac{1}{N}\left(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)\right)$$

where c, l, g represent the class, the predicted bounding box and the ground truth respectively. L_{loc} , defined with the following formula:

(2)

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k}\, smooth(l_i^m - \hat{g}_j^m)$$

is the sum of Smooth L1 loss across all bounding box properties (cx, cy, w, h) for matched positive boxes. L_{conf} is defined as:

(3)

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log\left(\hat{c}_i^p\right) - \sum_{i \in Neg}^{N} \log\left(\hat{c}_i^0\right)$$

where x_{ij}^{p} is the matching indicator between default box *i* and ground truth box *j* of category *p*, $\hat{c}_i^p$ is the softmax activated class score for default box *i* with category *p*.

6. Experiments

The experiments conducted on the network, using Google Colab platform, have been numerous but have made it possible to identify which are the improvement factors that can be applied in such a way as to improve

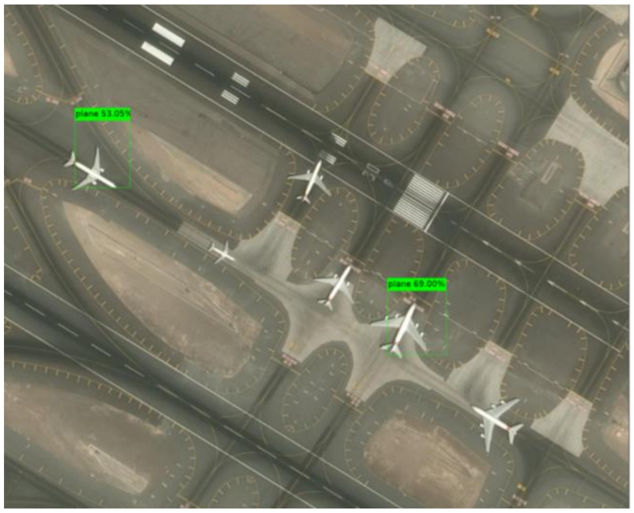

Figure 3: output example with 6250 iterations (50 epochs with 125 iterations per epoch)

[4] https://github.com/pierluigiferrari/ssd_keras/blob/master/training_summaries/ssd300_pascal_07%2B12_training_summary.md

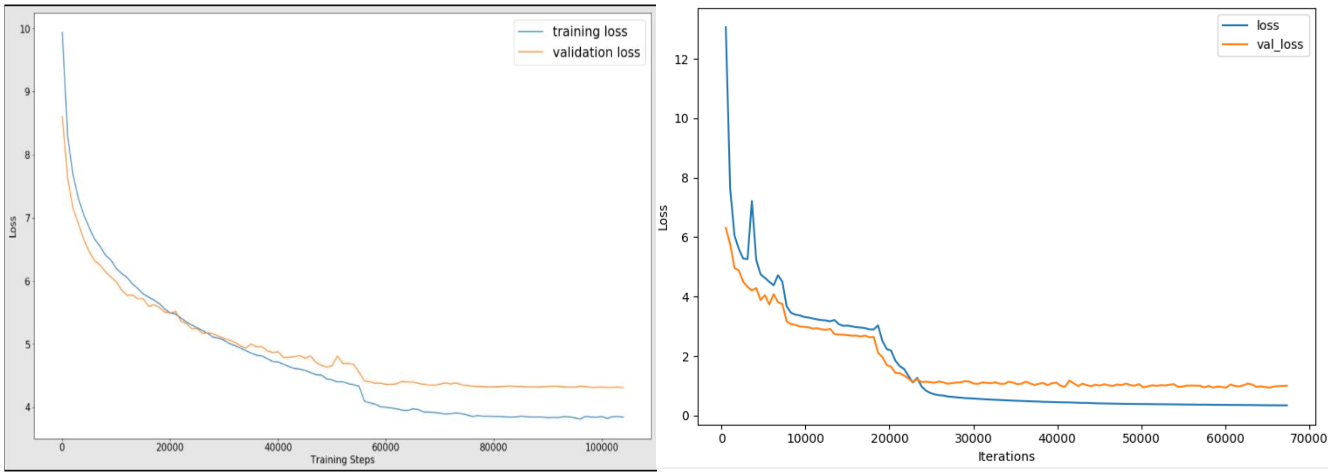

Figure 5: from left to right respectively development of the losses related to the number of iterations for the reference training example and this study

performance. The output of the network consists of Bounding Boxes with the aim of identifying objects of a single *airplane* class (in the example images represented as "plane" for graphic convenience compatibly with the average size of a box together with the confidence score). The hyperparameters identified and modified from time to time were *number of iterations*, *number of total images* and *learning rate*. In general, the results showed that the first two hyperparameters mainly contributed to the performance improvement. For the training part, a notebook[5] containing an example of training that uses the same loss was taken as a reference: overall the behavior of the losses is similar (see Figure 5) considering both the loss on the training set and on the validation set. The management of the learning rate among the epochs was decisive in improving these both loss values: in a first set of runs the learning rate was set manually in the various training phases, in a second it was treated in a totally automatic way, using a callback, for the first 50 runs to avoid that a too low learning rate was immediately reached and the training drastically decreased in speed (this would have decreased the number of total epochs due to the time limits of the Colab platform). The maximum number of epochs reached is 130 always due to the limits of the platform, but although the loss continued to decrease, the validation loss reached a stall point thus demonstrating a growing difference between the two since epoch 50 approximately. The next section illustrates the results of the 3 best runs (even if not temporally consecutive – see Table 1).

Run	# of iterations	Learning rate interval	# of images	Min loss	Min val_loss	AP50 (%)
#1	67340	[1e-3, 1e-6]	2059	0.3407	0.9396	16.40
#2	48750	[1e-3, 1e-6]	1854	0.5306	1.0032	13.59
#3	6250	[1e-3, 1e-5]	756	3.3401	4.5210	9.90

Table 1: table of the results of the 3 best runs

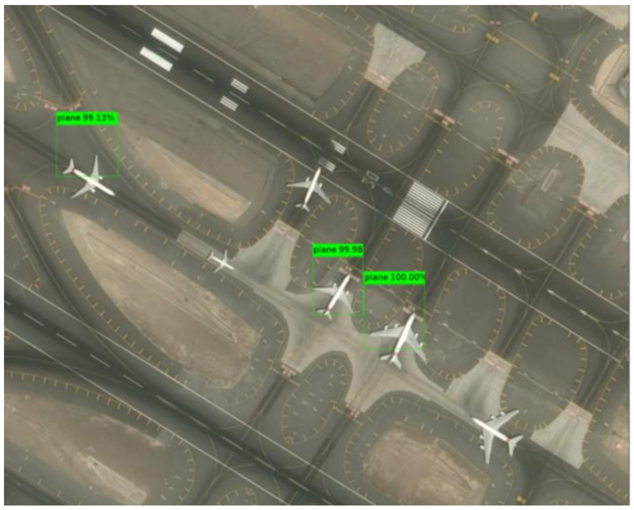

Figure 4: output example with 67340 iterations (130 epochs with 518 iterations per epoch)

[5] https://github.com/pierluigiferrari/ssd_keras/blob/master/training_summaries/ssd300_pascal_07%2B12_training_summary.md

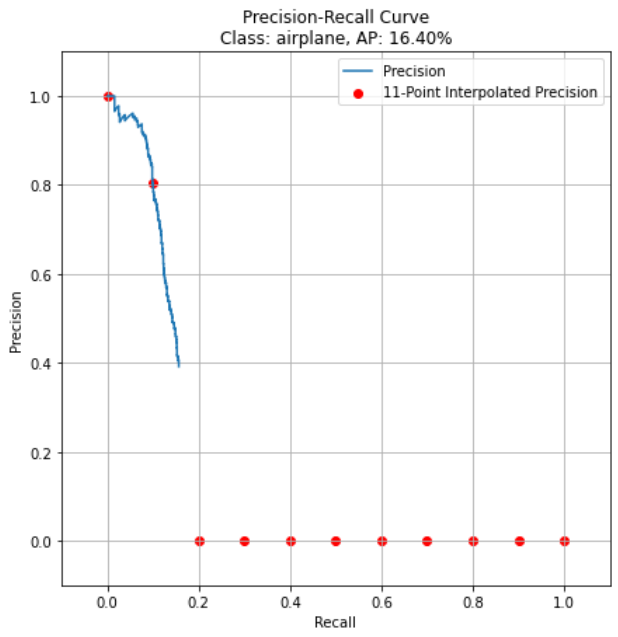

Figure 6: precision-recall curve referred to IoU 0.50

7. Results

The Average Precision referred to an IoU value of 0.50 was considered as an evaluation metric. Although the best run obtained does not have a high AP value (see Figure 6), it is possible to justify the result with several explanations: the table demonstrates how the number of images was fundamental for increasing performance. From run #3 to run #2 a new dataset was used, but from run #2 to run #1 the data augmentation performed on the training set only affected performance. The growing gap between the loss and the validation loss also indicates a variance problem, which can be solved by increasing the examples. The continuous decrease of the loss even up to the last epoch suggests instead a training with more epochs (at least to reach 102k iterations). On the other hand, the works illustrated in the article by S. Rawat [1] and in the paper by A. V. Etten [3] show how the values of *# of iterations* and *# of images* used in this paper are a small percentage of those necessary to obtain excellent performances.

8. Conclusions

Object detection is a very interesting branch of deep learning and although it represents a real challenge for beginners, it helps to learn a lot about this area as for the construction of a network suitable for the purpose it is necessary to consider and study different aspects. Although the performances achieved by this paper are not satisfactory, dealing with this type of architecture has allowed me to highlight how important it is not only to build a network but also to know how to manage and control it; the study of already made implementations was also fundamental to define the architecture even better and understand the regularization and normalization techniques. Further demonstration of how theory and practice are two very distinct, albeit related, things. A real challenge, however, was the construction of the dataset: finding freely accessible datasets with good images for this task was complicated but the greatest effort was to standardize all the data coming from the different sets into a single format. Possible future developments for this project are certainly the use of longer training, larger datasets and the recognition of different objects: after all, planes represent a small percentage of the information contained in a single image.

9. References

[1] S. Rawat, "Airplanes Detection for Satellite using Faster RCNN," 20 June 2019. [Online]. Available: https://towardsdatascience.com/airplanes-detection-for-satellite-using-faster-rcnn-d307d58353f1.

[2] W. Liu, D. Anguelov, C. Szegedy, C.-Y. Fu and A. Berg C., "SSD: Single Shot MultiBox Detector," 29 December 2016. [Online]. Available: https://arxiv.org/abs/1512.02325.

[3] A. V. Etten, 24 May 2018. [Online]. Available: https://arxiv.org/abs/1805.09512.