Challenge 1: detecting and classifying aerial images

For the first step of our challenge we decided to use keras-retinanet as implemented here: https://github.com/fizyr/keras-retinanet. For this purpose, we followed the instructions given in the repository to clone it and install all the necessary packages.
Then, both on our own and with the help of a guide (https://github.com/jaspereb/Retinanet-Tutorial) we adapted the repository to our situation, creating the necessary folders and sub-folders, changing the classes inside the pascal_voc.py script etc.
The files needed for the training are in a folder called “Training_set”. Inside Annotations we have the xml files, inside JPEGImages the images and inside ImageSets/Main we have three txt files we created:

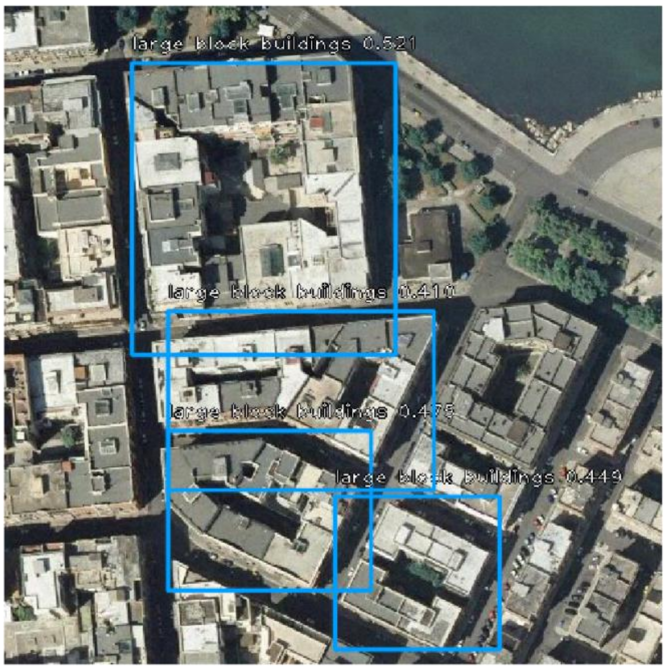
- trainval.txt lists every JPEGImages file without the extension
- train.txt is a subset of trainval.txt with all the images we trained on
- val.txt is a subset of trainval.txt with all the images we tested (validated) on

Moreover, to allow the training we had to modify the xml files, adding the “truncated”, “difficult” and “pose” field, changing the classes from numbers to strings and converting the decimal numbers from the format with the dot to the one with the comma. To see how we did it you can check the script xml_mod.py in our repository (https://github.com/cat-erina/Poste-Italiane).

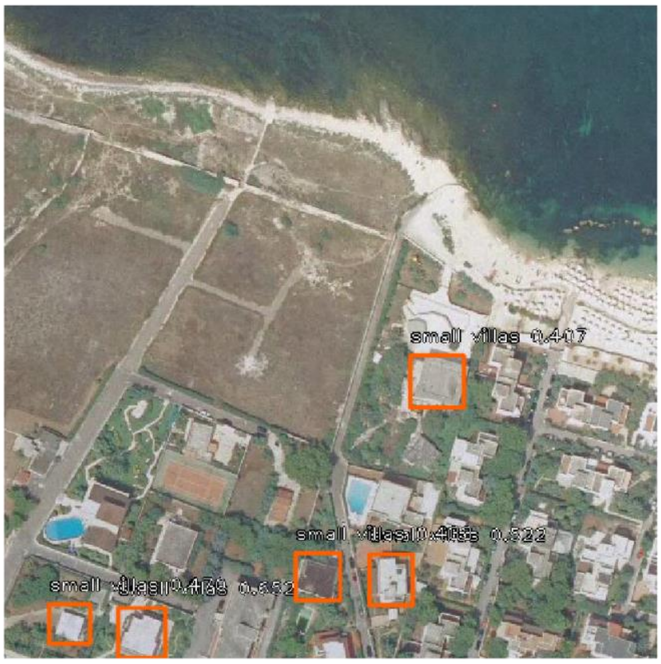
We were finally able to train the model, initially with a temporary value for step=100, to check if everything was working correctly. After 5 hours and 20 minutes we obtained 9 epochs of 100 steps each. Of course, this model wasn’t good enough, so we run the training again with 2000 steps for each epoch. After 12 hours we only managed to obtain 2 epochs. Of course, the model is still unstable but with the computational power available it wasn’t possible to improve it in a short time. These are the metrics we obtained:

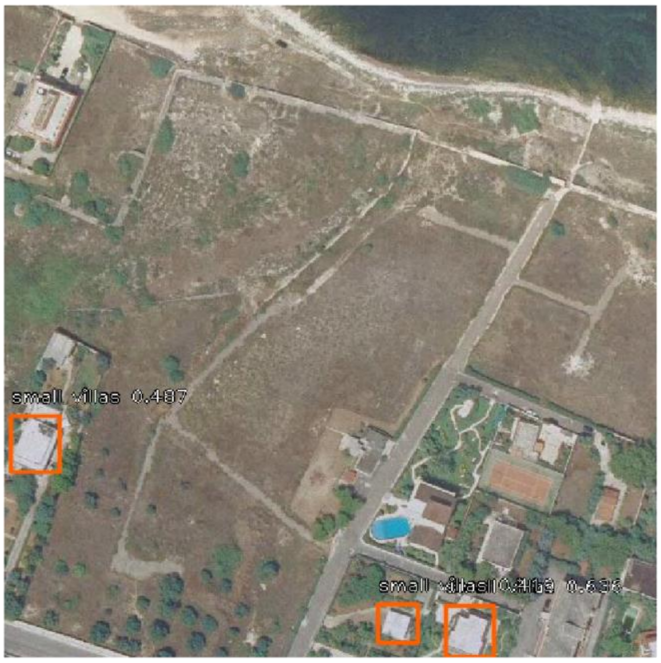
- Loss: 2.6424
- Regression loss: 1.5244
- Classification loss: 1.1180

The average precision is 0.305 for large block building and 0.1656 for small villas.


large block buildings 0.521
large block buildings 0.410
large block buildings 0.475
large block buildings 0.449

Even with such a small amount of training the model is able to detect and classify correctly some of the building, although the confidence scores are low. This is why we decided to create the final csv for step 1 including all the detected objects with a score greater than 0.4. We also resized images and this is why the final coordinates may look different.
As all the other files mentioned here the csv and the script wrote to create it can be found in our repository respectively with the names “Output_2.csv” and “TinoLoScriptino.py”.





This is an example of how our csv is structured:

	Image	xMin	yMin	xMax	yMax	Class	Confidence
0	000001384.jpg	9,890329	104,693565	44,273582	139,49585	2	0,53931487
1	000000071.jpg	0,0	295,0603	236,11963	457,61212	1	0,503542
2	000000059.jpg	161,66324	404,44473	190,45729	442,0092	2	0,47904372
3	000000059.jpg	175,58505	348,48685	204,2708	385,18182	2	0,46009654
4	000000059.jpg	265,8855	172,69504	306,0974	206,14308	2	0,4519028

Finally, for step two we used the information of the csv to estimate how many people lived in the area of every image. To do this we first calculated the area of every building/villa detected doing:

area = ((xmax-xmin)/2)*((ymax - ymin)/2)

The division by 2 is needed because the images are in scale 0.5, so to convert the measure in pixel to measures in meters we have to divided them by 2. Then, for every object detected we estimated how many people live there depending on their class. The area of large block buildings was multiplied by 0.065 (population density) and the area of small villas was multiplied by 0.010 (population density).
Finally, we summed all the results of buildings or villas of the same image and estimated the population of every image. The result is a csv called "Output_part2.csv" as well as the file that executed it: "Part2.py"

Improvements to get a better model include for sure training it for more epochs and with more steps, as well as considering how to deal with the truncated borders of the images.