Final project report

Course: Computer Vision 2020

Francesco Pham (1234004)

Automatic tree detection and recognition

This report covers my final project on object detection and recognition.
To achieve the objectives of this project, the approach I decided to follow is to use the cascade classifier originally proposed by Paul Viola and Michael J. Jone in 2004 for the face recognition. The classifier is already implemented in the OpenCV library under the name **CascadeClassifier.**

Notes

The input images need to be stored inside the 'data' folder. If you add some images, remember to execute the 'cmake' command because CMakeLists.txt is configured to copy the content of 'data' directory inside the build directory.
Inside the dataset_generation directory you can find my python script for the data augmentation and dataset generation plus a small subset of the dataset I used to train the detector.

Dataset generation

The dataset I used to train the classifier was manually created by me by downloading the pictures from google images. The number of images I managed to collect was 120 positive samples of tree images and 144 negative samples of non-tree images.

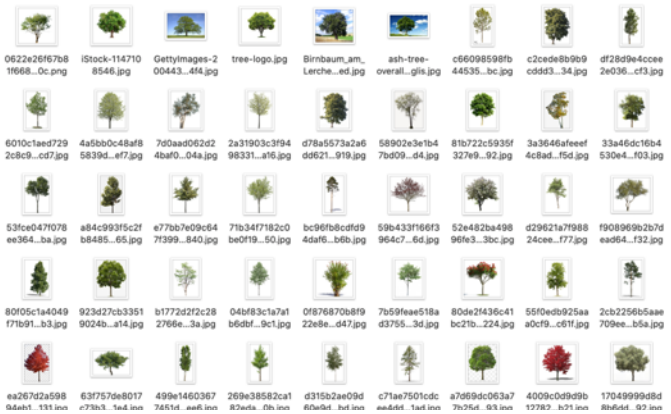


Figure 1 - Positive samples

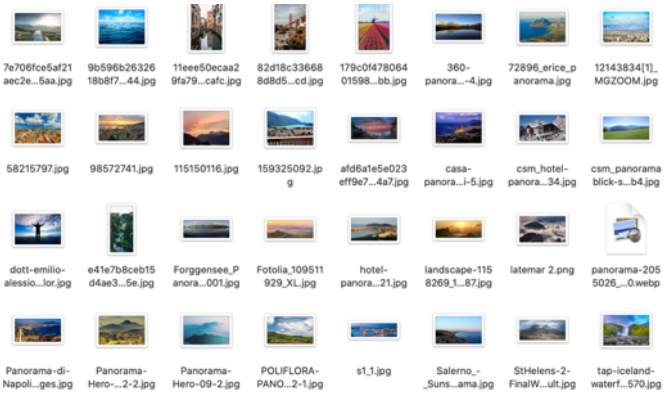


Figure 2 – Negative samples

In order to increase the diversity of the collected data I performed a data augmentation using the Keras library by applying some image processing operations such as rotation of ±20 degrees, shear, zoom, horizontal flip and brightness change. The final dataset consists of 450 positives and 500 negatives.

Figure 3 - Augmented dataset

Finally, by using python code I generated two text files positives.txt and negatives.txt containing the list of paths to the positive and negative samples, these files are needed for the training process.

Training

First of all, I executed the following command to generate the positives.vec file containing all the positive samples with size 50x50.

```
opencv_createsamples -info positives.txt -vec positives.vec -w 50 -h 50 -num 496
```

Finally, to perform the training process I used the following command:

```
opencv_traincascade -data data -vec positives.vec -bg negatives.txt -numPos 450 -numNeg 500 -numStages 24 -w 50 -h 50 -featureType LBP
```

This command runs the training of the cascade classifier using the positive and negative samples provided. After some tests tuning the parameters, I ended up using 24 stages. I also tested both LBP and Haar feature types and decided to use LBP because the local binary patterns features are much simpler, therefore the learning is much faster than using Haar features. The whole process lasted about 35 minutes. The training produced an output file cascade.xml which is our cascade classifier.

Now let's visualize the trained cascade, to see which feature it selected. OpenCV provides a handy tool which is `opencv_visualisation` which shows the features it detects on a sample image.

Image preprocessing

In order to improve the performance of trees detection of the cascade classifier I decided to execute some image preprocessing first on the input image before passing it to the cascade classifier. The preprocessing is executed inside the `preprocess_image` function and includes operations such as histogram equalization in RGB space, bilateral filtering, and a change of brightness of selected pixels inside a specific range of hue and value in HSV space. The reason of doing the latter operation is because I noticed a better detection when the trees are in contrast with a brighter background, which is reasonable because the detection is performed on the grayscale image and color information are not taken into account. The selection of the range in HSV space is aimed to select the sky and other stuff that are not related to trees and brighten up those pixels in order to have more contrast with the trees. The parameters like bilateral filter sigma, hue and value ranges are selected with a trackbar and stored inside `DetectionParams` struct.

Figure 4 - Preprocessed image

Figure 5 - Not preprocessed

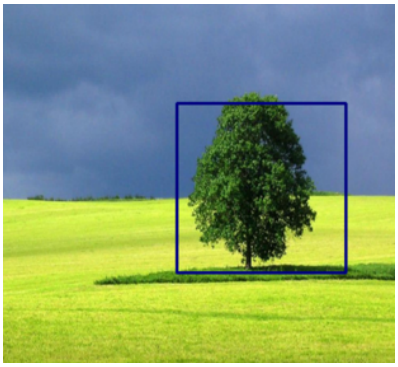
Detection

The detection phase is implemented inside the function `detect_and_display`. The function of the cascade classifier responsible of doing the detection is `detectMultiScale` which takes some parameters such as how many neighbors each candidate rectangle should have to retain it and the minimum possible object size. These parameters are selected with a trackbar and stored inside a `DetectionParams` struct. Finally, a bounding box is drawn around each detected tree and the result is shown both on the preprocessed image and on the original image.

Results

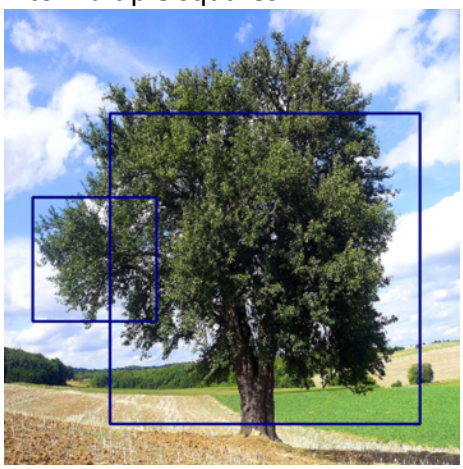
After some time tuning the various parameters, the results I get are quite satisfactory. The trees that are clearly visible in a contrasting background are correctly detected. However, there are some false positive and some false negatives, and sometimes there are multiple detection for a single tree.

Here are some correctly detected trees:


In the following image just one tree clearly visible in front of a sky background is detected:

In the following images there are two detections of the same tree, keep in mind that the window I selected is a square, which may be the reason why long vertical trees may be split into multiple squares:




In images with no trees it looks like no false positives are detected:


In addition to the test images provided by the professor, I added some pictures that I personally took of trees around my house.