Object detection using Haar-cascade Classifier

Sander Soo

Institute of Computer Science, University of Tartu

sander92@ut.ee

Abstract

Object detection is an important feature of computer science. The benefits of object detection is however not limited to someone with a doctorate of informatics.
Instead, object detection is growing deeper and deeper into the common parts of the information society, lending a helping hand wherever needed. This paper will address one such possibility, namely the help of a Haar-cascade classifier.

The main focus will be on the case study of a vehicle detection and counting system and the possibilities it will provide in a semi-enclosed area - both the statistical kind and also for the common man. The goal of the system to be developed is to further ease and augment the everyday part of our lives.

1. Introduction and theory

1.1 Computer vision

Computer vision is a field of informatics, which teaches computers to see. It is a way computers gather and interpret visual information from the surrounding environment [1].

Usually the image is first processed on a lower level to enhance picture quality, for example remove noise. Then the picture is processed on a higher level, for example detecting patterns and shapes, and thereby trying to determine, what is in the picture [2].

1.2 Object detection

Object detection is commonly referred to as a method that is responsible for discovering and identifying the existence of objects of a certain class. An extension of this can be considered as a method of image processing to identify objects from digital images.

1.3 Simple detection by colour

One way to do so, it to simply classify objects in images according to colour. This is the main variant used in, for example, robotic soccer, where different teams have assembled their robots and go head to head with other

teams.
However, this color-coded approach has its downsides. Experiments in the international RoboCup competition have shown that the lighting conditions are extremely detrimental to the outcome of the game and even the slightest ambient light change can prove fatal to the success of one or the other team. Participants need to recalibrate their systems multiple times even on the same field, because of the minor ambient light change that occurs with the time of day. [3] Of course, this type of detection is not suitable for most real world applications, just because of the constant need for recalibration and maintenance.

1.4 Introduction of Haar-like features

A more sophisticated method is therefore required. One such method would be the detection of objects from images using features or specific structures of the object in question.

However, there was a problem. Working with only image intensities, meaning the RGB pixel values in every single pixel in the image, made feature calculation rather computationally expensive and therefore slow on most platforms.
This problem was addressed by the so-called Haar-like features, developed by Viola and Jones on the basis of the proposal by Papageorgiou et. al in 1998.

A Haar-like feature considers neighbouring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image.

An example of this would be the detection of human faces. Commonly, the areas around the eyes are darker than the areas on the cheeks. One example of a Haar-like feature for face detection is therefore a set of two neighbouring rectangular areas above the eye and cheek regions. [4]

1.5 Cascade classifier

The cascade classifier consists of a list of stages, where each stage consists of a list of weak learners.

The system detects objects in question by moving a window over the image. Each stage of the classifier labels the specific region defined by the current location of the window as either positive or negative – positive meaning that an object was found or negative means that the specified object was not found in the image.

If the labelling yields a negative result, then the classification of this specific region is hereby complete and the location of the window is moved to the next location.

If the labelling gives a positive result, then the region moves of to the next stage of classification.

The classifier yields a final verdict of positive, when all the stages, including the last one, yield a result, saying that the object is found in the image.

A true positive means that the object in question is indeed in the image and the classifier labels it as such – a positive result. A false positive means that the labelling process falsely determines, that the object is located in the image, although it is not. A false negative occurs when the classifier is unable to detect the actual object from the image and a true negative means that a non-object was correctly classifier as not being the object in question.

In order to work well, each stage of the cascade must have a low false negative rate, because if the actual object is classified as a non-object, then the classification of that branch stops, with no way to correct the mistake made. However, each stage can have a relatively high false positive rate, because even if the n-th stage classifies the non-object as actually being the object, then this mistake can be fixed in n+1-th and subsequent stages of the classifier. [5]

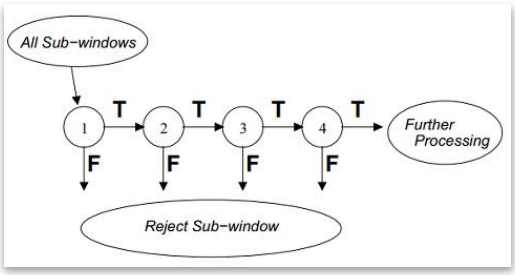

Image 1: *Stages of the cascade classifier*

2. Related work

2.1 PureTech Systems Car Counting

The system uses images received from IP and analog video cameras it to detect and count vehicles. Advanced background algorithms then filter any unnecessary and probable interference, such as shadows or lighting changes. When an object in detected, special filters make sure to minimize the chance to count nonvehicle items, for example humans and luggage. The finalized count is then outputted, based on the initial configuration - by floor or special zone.

The video is processed at a central monitoring location, which means there is no need to make cuts into pavement or similarly preinfluence the environment in such a way, which is commonly needed for inductive loop traffic detectors, where the detectors need to be placed inside the pavement, which is a fairly common method of detecting vehicles behind traffic lights.

One problem with this system is, however, that it only detects cars as they enter or leave the parking lot. This in turn means that there needs to be such a camera or counter in front of every entrance and exit point, in order to ensure that none of the vehicles are missed, which means that all these cameras need to be bought, installed and serviced, which means that the specific client will face a possibly substantial expenditure in order to keep all the devices up and running. [6]

2.2 Autonomous Real-time Vehicle Detection from a Medium-Level UAV

The primary means for vehicle detection make use of multiple cascaded Haar classifiers. For the application, four separate Haar-cascade classifiers were trained based on sample vehicles categorized into four different positional orientation to the horizontal line, resulting in the clockwise offsets of 0, 45, 90, 135 degrees. The training set was thereby divided into disjoint subsets based on the nearest perceived angle of the vehicle wheelbase. A separate cascaded Haar-classifier was trained for each of the subsets.

The results are subsequently passed through a secondary disjunctive verification process, which means that a vehicle may exist in one or more of the input images, if one or more of the different orientation specific classifiers yields a positive result. The returned set of regions are then merged to resolve overlaps and output a singular set of detections from the inputs. Each region is also tested by various logical tests of width and height – is it actually probable for the image of this size to be a vehicle. [7]

2.3 Monocular Vehicle Detection and Tracking

The system developed involves three main steps.

Firstly, the preliminary car targets are generated with Haar-cascade classifier. Only the candidates that pass through all the stages are classified as positive and the ones that are rejected at any stage are classified as negative. The advantage is that the majority of the initial candidates are actually negative images, which usually cannot pass the first few stages. This early-stage rejection thus greatly reduces the overall computing time.

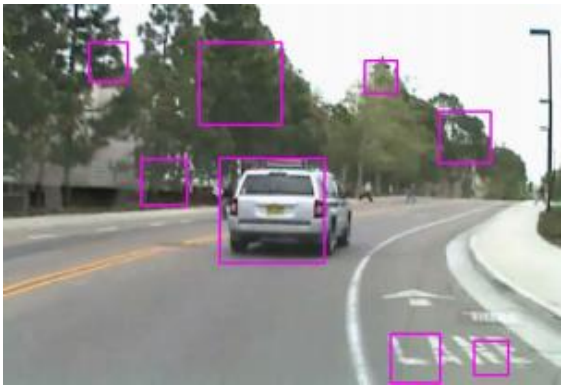

Image 2: Car candidates detected by Haar-cascade

After the first stage is complete, the target validation is used, which is based on car-light feature. Since there can be several false positives within the images output by the first stage of the system, which comes from the limitation of the training set. To reduce the false alarm rate, the algorithm uses the before mentioned car-light feature. Regardless of the shape, texture or colour of the vehicle, they all share a common feature – they all have red lights in the rear. Thereby assuming that most of the false positives detected by the first stage do not possess such a feature, the result can thus be refined.

Lastly, the results are further refined by Kalman tracking of the objects. The

main idea of this step is based on a three-stage hypothesis tracking. Firstly, if a newly detected area appears and lasts for more than a certain number of steps, a hypothesis of the object is generated. Then it is predicted where the next location of the object should be.

3. Object detection using Haar-cascade classifier

This section will highlight on the work conducted on the author's research in the field of object detection using Haar-cascade classifier. The experiments were conducted mainly on the parking lot located on Kivi street side parking lot of Raatuse Shopping center. The location was chosen mainly for the ease of access and security for the hardware required to gather information.

3.1 Hardware

Initial testing was conducted with Qoltec WebCam15QT NightVision 30Mpx USB camera. The device was chosen due to its alleged high capabilities, especially the 30MP camera.

As it later turned out, the information provided was faulty, and the quality of the camera was in fact much poorer. For this reason, the device was replaced with Logitech HD Webcam C525 which offered much clearer images than its predecessor.

The camera was programmed to take pictures every five minutes, to minimize the impact on the storage capacity and duplicate images, since the changes during five minutes in the parking lot were observed to be minimal.

If the object is not detected for a certain number of frames, the hypothesis is discarded. This method can thusly eliminate false positives that do not last long enough and still keep track of objects that are missing for only a short period in a detection step. [8]

3.2 Software

Several programs were developed in the course of this paper, ranging from a simple convert to grayscale and get size of picture to recorder, detector and PosCreator.

3.2.1 Recorder

Recorder application was a simple application which after every 5 minutes tries to take a picture. If it can, then a picture is saved to a folder of the corresponding date with the filename of the corresponding time. If it cannot, then it simply cuts the connection within 30 seconds and will simply wait for the next 5 minutes. This ensures that if there is a problem with taking a picture, which would cause the program to "freeze", then it is simply stops the program and tries again later, instead of potentially waiting until the power runs out someone manually stops the program. This is a must-have feature is such an application, due to the fact that several hours' worth of image gathering would be wasted due to any simple problem that halts the execution of the recorder thread.

3.2.2 Detector

The initial design of the detector application was quite simplistic. Firstly, the detector would load the classifier

and determine it is not empty. If it is, then it simply exits with an error message. Then the image in question is loaded and same procedure is followed. Then classifier is applied to the image, which outputs an array of rectangles, which correspond to the detected positions of the objects, in this case automobiles. The program would then draw bright red rectangles in the locations of the detections and also add a text to the image, which could for example identify the classifier used, since one classifier would usually detect one thing.

3.2.2.1 Background subtraction

However, as shown by the testing process and the literature, the classifiers trained can produce errors – either false positives or false negatives, as described above.

In order to minimise the false positive rate originating from the imperfections of the classifier, an additional layer was added to the algorithm, before the classifier is applied to the image. This layer has additional knowledge of the complete background. In this case it would be an image of only the parking lot and everything that would normally be in the parking lot, except for the cars themselves. This knowledge can be applied to attempt the filtering of the background from the image from which we would like to detect vehicles. The background subtraction type used was MOG.

MOG (abbr. from Mixture of Gaussians) is a technique used to model the background pixels of an image like a mixture of Gaussians of different weight that represent the pixel intensity. If a match is found for the pixel of the new image with one of the Gaussians then the point is classified as a background pixel. If no match is found, then the pixel is classified as the foreground pixel. [9]

Other algorithms, such as MOG2 were considered, but MOG was finally chosen due to the simple fact that clearer results were obtained by using MOG.

MOG gives us the background mask, so in order to apply it to the original picture, one would simply need to compute the bitwise and between the original image and the mask provided.

MOG is, however, not perfect. If we were to just take the mask provided by the default MOG background extractor, then the output for one image of the parking lot would be rather low quality, as illustrated on image 3. Although a person may differentiate the regions of cars in the image, a cascade classifier proved unable to properly comprehend the regions of cars on a similar image.

Image 3: Output using MOG with default parameters

3.2.2.2 Background subtraction augmentation

In order to amend this issue, different augmenting features had to be used. The ones chosen were eroding and dilating.

Dilation is a way to make the bright regions of an image to “grow”. As the kernel (small matrix used for image processing) is scanned over the image, the maximal pixel value overlapped by the kernel is calculated and the image pixel in the anchor point of the kernel (usually at the centre of the image) is replaced by the maximal value.

Erosion works similarly to dilation, but instead of the maximal, it computes the local minimum over the area of the kernel, thus making the dark areas of the image larger.

If one were to apply dilation to the mask provided by MOG, then the areas of the mask which are not zeros would get larger, thus improving the overall quality of the image.

This can however raise a new issue, namely the fact that the small noisy areas present in the original mask could grow larger and have a negative effect on the provided mask.

For this reason, the once dilated mask is eroded with a kernel with a smaller size, so that it would not nullify the result provided by the dilating but still reducing the amount of noise produced by the dilation process, thus providing a symbiotic relation between the two operations.

Image 4: MOG with dilation and erosion

The results provided by this sort of background filtering were improved. Since a lot of the false positives provided by the original detections were in fact on the background part, such as the trees, pavement etc., which is always there, then the algorithm discarded these areas before the Haar-cascade classifier would be applied. However, the regions created by the background removal created additional problems, such as the classifier mistaking the grey to black regions as the positive image.

3.2.3 Regional merging

In addition, yet another layer of filtering was added, inspired by the “Autonomous Real-time Vehicle Detection from a Medium-Level UAV” article. This layer was added after the Haar-cascade has done its work. Since it is possible for the classifier to detect many parts of a car as several distinct cars, the results of the classifier regions must be merged. The merge cannot however just detect whether the two regions have an overlap of any size, because the overlap can simply be the result of two adjacent cars, which would thereby be merged into one result. To avoid this unwanted behaviour, the two regions are merged only if the overlapped area is at least a certain

percentage of one of the regions. This constant was observed to yield the best results if the value was approximately 40-60%, meaning that if the overlap of the two regions made up at least 40-60% of the overall area of one of the regions, then the two regions were merged and considered as one region from there on. An example of this procedure is shown on image 5, where green lines denote the original detection and the red line the final output of the detector.

Image 5: Regional merge in action

Due to the probabilistic nature of the Haar-cascade, the false positives may also include detection, which logically cannot contain an automobile – namely when the rectangle identified as a positive result is either too large or too small. Such a result would seriously compromise the result set, because the algorithm described for the merging of areas would simply merge all these areas into one large area. In order to avoid this, the results of the Haar-cascade are first prefiltered, so that areas that are too large to contain just a single car are removed from the results. This size constant was found by observing the largest truck that was closest to the camera and adding 20% to the size.

3.2.4 Training cascade

The training of the cascade proved to be no easy task. The first necessary bit was to gather the images, then create samples based on them and finally starting the training process. The opencv traincascade utility is an improvement over its predecessor in several aspects, one of them being that traincascade allows the training process to be multithreaded, which reduces the time it takes to finish the training of the classifier. This multithreaded approach is only applied during the precalculation step however, so the overall time to train is still quite significant, resulting in hours,days and weeks of training time. [10, 11, 12]

Since the training process needs a lot of positive and negative input images, which may not always be present, then a way to circumvent this is to use a tool for the creation of such positive images.

OpenCV built in mode allows to create more positive images with distorting the original positive image and applying a background image. However, it does not allow to do this for multiple images.

By using the Perl script createsamples to apply distortions in batch and the mergevec tool [11], it is possible to create such necessary files for each positive input file and then merging the outputted files together into one input file that OpenCV can understand.

Another important aspect to consider is the number of positives and negatives. When executing the command to start training, it is required to enter the number of positive and negative images

that will be used. Special care should be taken with these variables, since the number of positive images here denotes the number of positive images to be used on each step of the classifier training, which means that if one were to specify to use all images on every step, then at one point the training process would end in an error. This is due to the way the training process is set up, as described in section 1. The process needs to use many different images on every stage of the classification and if one were to give all to the first stage, then there would be no images left over for the second stage, thus resulting in an error message. [10, 11, 12]

The training can result in many types of unwanted behaviour. Most common of these is either overtraining or undertraining of the classifier. An undertrained classifier will most likely output too many false positives, since the training process has not had time to properly determine which actually is positive and which is not. An output may look similar to image 6.

Image 6: Depiction of the results of undertrained classifier

The opposite effect may be observed if too many stages are trained, which could mean that the classification process may determine that even the positive objects in the picture are actually negative ones, resulting in an empty result set.

Fairly undefined behaviour can occur if the number of input images are too low, since the training program cannot get enough information on the actual object to be able to classify it correctly.

One of the best results obtained in the course of this work is depicted on image 7. As one can observe, the classifier does detect some vehicles without any problems, but unfortunately also some areas of the pavement and some parts of grass are also classified as a car. Also some cars are not detected as standalone cars.

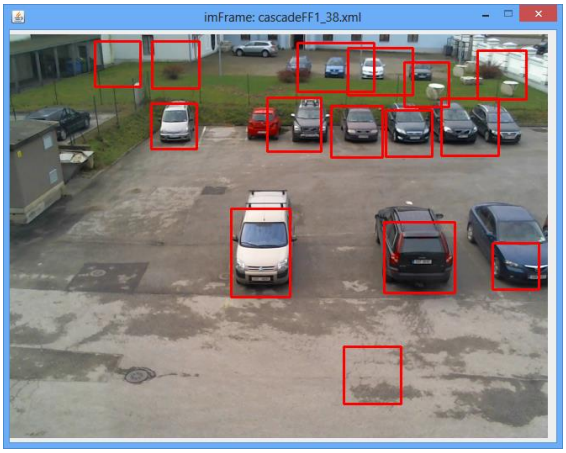

Image 7: Best solution obtained by the author

The time taken to train the classifier to detect at this level can be measured in days and weeks, rather than hours.

Since the training process is fairly probabilistic, then a lot of work did also go into testing the various parameters used in this work, from the number of input images to the subtle changes in the structuring element on the

background removal, and verifying whether the output improved, decreased or remained unchanged.

For the same reason, unfortunately the author of this work was unable to produce a proper classifier, which would give minimal false positives and maximal true positives.

4. References

[1] "Vallaste e-teatmik," [Online]. Available: http://vallaste.ee/index.htm?Type=UserId&otsing=5027. [Accessed October 2013].

[2] S. Nagabhushana, "Introduction," in *Computer Vision and Image Processing*, New Age International (P) Ltd., Publishers, 2005, p. 3.

[3] V. E.-C. Nathan Lovell, "Color Classification and Object Recognition for Robotic Soccer under Variable Illumination," Griffith University.

[4] V. Jones, "Rapid object detection using a boosted cascade of simple features," Computer Vision and Pattern Recognition, 2001.

[5] T. M. Inc., "Train a Cascade Object Detector," [Online]. Available: http://www.mathworks.se/help/vision/ug/train-a-cascade-object-detector.html#btugex8. [Accessed Nov 2014].

[6] "Car Counting," PureTech Systems, [Online]. Available: http://www.puretechsystems.com/solutions-car-counting.html. [Accessed Nov 2014].

[7] T. P. Breckon, S. E. Barnes, M. L. Eichner and K. Wahren, "Autonomous Real-time Vehicle Detection from a Medium-Level UAV," 2008. [Online]. Available: http://breckon.eu/toby/publications/papers/breckon09uavvehicles.pdf. [Accessed Nov 2014].

[8] Y. Wang, "Monocular Vehicle Detection and Tracking," University of California, [Online]. Available: http://acsweb.ucsd.edu/~yuw176/report/vehicle.pdf. [Accessed Nov 2014].

[9] T. Bouwmans, F. E. Baf and B. Vachon, "Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey," Bentham Science Publishers, 2008. [Online]. Available: https://hal.archives-ouvertes.fr/file/index/docid/338206/filename/RPCS_2008.pdf.

[10] OpenCV, "Cascade Classifier Training — OpenCV 2.4.9.0 documentation," [Online]. Available: http://docs.opencv.org/doc/user_guide/ug_traincascade.html. [Accessed December 2014].

[11] C. Robin, "Train your own OpenCV HAAR classifier,"

[Online]. Available: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html. [Accessed December 2014].

[12] N. Seo, “OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features),” [Online]. Available: http://note.sonots.com/SciSoftware/haartraining.html. [Accessed December 2014].