UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CHARLES DANIEL RIBEIRO ARNOUD

Faster than the Fastest: Using Calibrated Cameras to Improve The Fastest Pedestrian Detector in the West

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Cláudio Rosito Jung

Porto Alegre
July 2015

ABSTRACT

In this thesis, we translated a state-of-the-art object detector (the Dóllar method) to the C++ programming language and explored ways to use camera calibration to improve its performance by reducing the amount of calculations necessary and to improve the results by taking away false positives. We developed these techniques in the context of pedestrian detection.

On data sets more aligned with video surveillance applications (the camera is high in relation to the ground and far from the area where objects are expected to be), we had great results across the board: the amount of scales in the feature pyramid is reduced by about half, the amount of times the classifier is applied is greatly reduced together with the number of false detections, all while keeping the loss in detection coverage manageable.

We also tested our detector in one data set that closely resembles the use of detection in robotics or self-driving systems for automobiles (camera closer to the ground plane and parallel to it). The results suggest the method needs adjustments to be applied to this type of setting. Although there was no loss in detection quality and both the number of scales in the feature pyramid and the number of false positives were reduced, the amount of classifier applications seems excessive. To avoid this problem, we need to adjust the Dense Detection phase of our method (subsection 3.2.2) to account for the fact that images created by these camera settings have a bigger range of possible pedestrian heights and more portions of the image are plausible to provide detections.

Keywords: Computer Vision. Pedestrian Detection.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Pedestrian detection is a particular case of the object detection problem, which is a very active research topic in recent years. It has applications in the fields of robotics, video surveillance and automobile safety and self-driving systems, to name a few. Since it gathered so much attention, this field (together with face detection) has been the main source of innovation in object detection (SZELISKI, 2010; BENENSON et al., 2014; DOLLÁR; BELONGIE; PERONA, 2010). As such, a variety of methods have been developed in this field with the intention of being applied to any other type of object, those are generic methods. These methods face challenges such as occlusion, high computation cost and false positives among the detections. If we intend to get the best results while detecting specifically pedestrians we can make use of assumptions about the scene geometry and object pose to reduce the influence of such difficulties, improving the detection quality and the performance of the detector.

Our work here aims to improve existing detectors using a calibrated camera and information about the dimensions of the targeted object in the world coordinate system. Since we assume the target object has its base on the ground plane and has a perpendicular posture in relation to it, we focus on pedestrian detection. Our method reduces the range of window sizes evaluated by the classifier, applies detection only in parts of the source image where it is plausible that a detection might occur based on the geometry of the scene and, later, tweaks the detection suppression step to avoid multiple detections of the same pedestrian while also avoiding the loss of legitimate detections.

The detector we chose to base our methods in is the one described in the paper named The Fastest Pedestrian Detector in the West (DOLLÁR; BELONGIE; PERONA, 2010). It provides us good detection quality, performance comparable to the state of the art and an easily available source code in the Matlab programming language. Note that the methods described here can be applied to any pedestrian detector provided we have access to the classifier itself, including any procedure it applies to the source image, and information about the camera. This work was also briefly tested in an automobile detector with good results, although that will not be explored here.

In chapter 2, we present some of the theory behind pedestrian detection and camera calibration. Next, we describe our method (chapter 3) and document some of the results found (chapter 4). In chapter 5, we draw conclusions about our findings.

2 FUNDAMENTALS AND RELATED WORK

In this chapter, we explain some of the building blocks required by our method. Section 2.1 focuses on pedestrian detection, while section 2.2 describes the operations we are able to make when we have a calibrated camera. We also give an overview of how to get the projection matrix of a camera in section 2.3.

Most of this chapter's content can also be found on Computer Vision: Algorithms and Applications (SZELISKI, 2010), it is a very complete textbook for computer vision courses. For a review of the state-of-the-art in object detection, we recommend the Ten Years of Pedestrian Detection, What Have We Learned? survey (BENENSON et al., 2014).

2.1 Pedestrian Detection

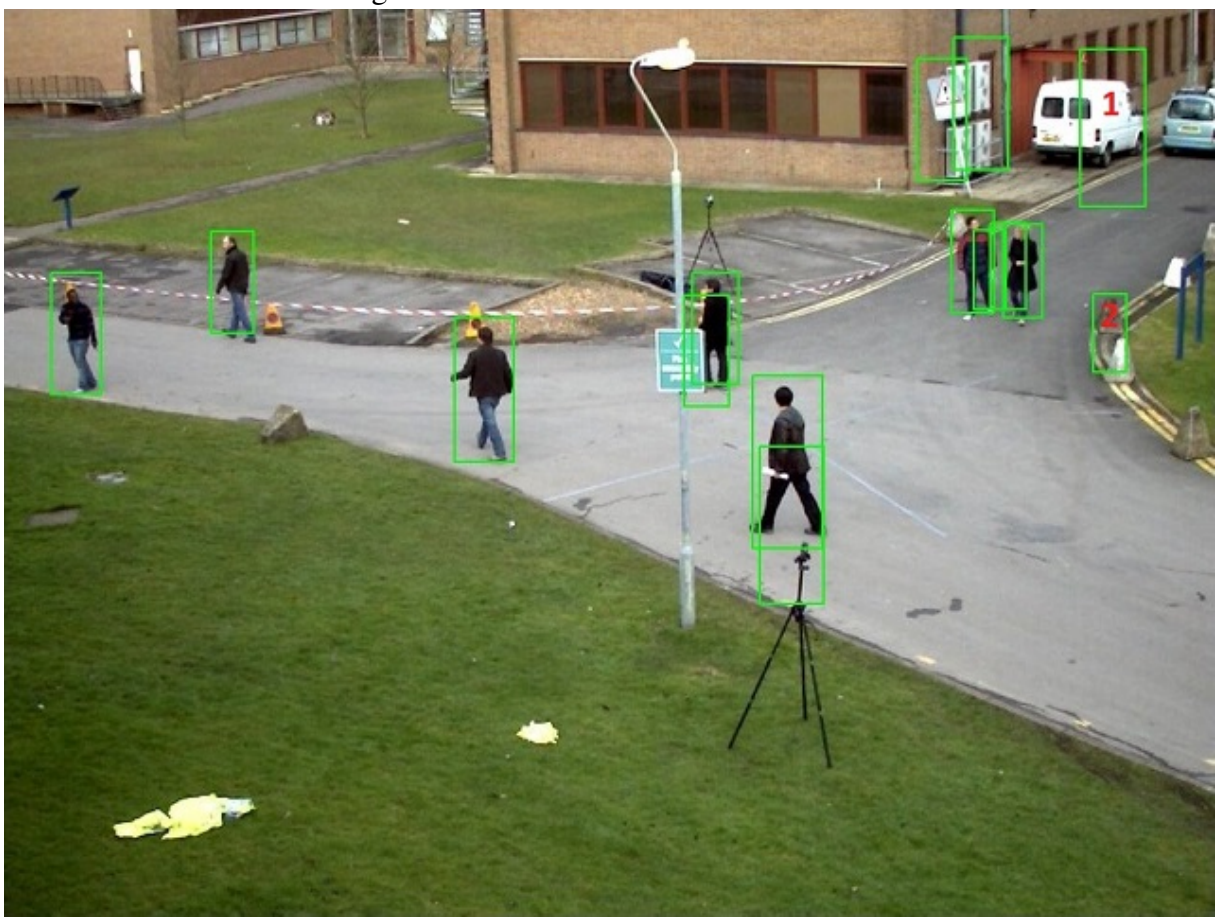
The pedestrian detection problem is a specific case of the object detection problem. Like the general case, it is usually solved by reducing it to multiple applications of the object classification problem (classifying an image as belonging to the target class or not). This is done by applying the trained classifier to different patches of the source image.

To detect pedestrians of different heights and positions, these patches tend to cover all portions of the image and to vary in dimensions (scale) by quite a lot. The process of applying a classifier to all of these patches in the source image is what is called sliding window detection.

Sliding window detectors are very common and show great results for generic images, but even good ones usually present two flaws that we try to reduce by using camera calibration: high computation cost and false detections. This is because the classifier is applied to windows covering the whole image in all of the chosen scales and lots of these applications might prove to be useless or, even, hurtful to the overall quality of the detections. In the case of objects that don't vary too much in real world dimensions, we might test and even detect candidates that would have unrealistic dimensions. We saw it happen in several tests of classifiers throughout the development of this work (as exemplified by figure 2.1).

The two most common approaches used in sliding window detectors are the ones that use an image pyramid (subsection 2.1.1) and the ones that use a classifier pyramid (subsection 2.1.2). The previously mentioned Dóllar method (subsection 2.1.3) is a hybrid of both. Figure 2.2 shows a side-by-side comparison of the three.

Figure 2.1: Common Detection Problems



This image is the result of applying the original The Fastest Pedestrian Detector in the West code to a frame in the PETS 2009 data set (chapter 4). Detection 1 is an example of a detection that is too tall in real-world coordinates, detection 2 is an example of a detection being too short in real-world coordinates. The figure also contains several instances of multiple detections of a single pedestrian.

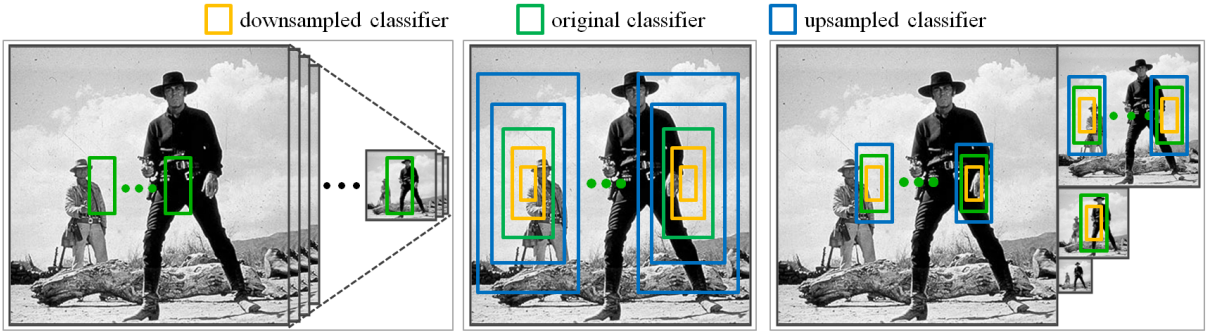
2.1.1 Image Pyramid Detectors

Patches have fixed dimensions, but are retrieved from different scales of the source image. This is the traditional approach, where a densely sampled image pyramid is created, then features are extracted from each of the pyramid's images. These features are, later, analyzed by a sliding window classifier. This approach (used in (DALAL; TRIGGS, 2005)) results in great detection quality, but the creation of such image pyramid has immense computational cost.

2.1.2 Classifier Pyramid Detectors

Patches change dimensions and the source image is used only in its original dimensions. The features extracted from the source image can be easily rescaled and are easy to compute so

Figure 2.2: Detector Types



(a) dense image pyramid (b) classifier pyramid (c) proposed hybrid approach

This figure was taken from the paper which describes The Fastest Detector in the West (DOLLÁR; BELONGIE; PERONA, 2010). In (a), the features are extracted from all scales; in (b), scale-invariant features are extracted from one scale and upsampled/downsampled for all others; in (c), the features are extracted from some scales and upsampled/downsampled for the scales close to the ones from which features were extracted.

there is no need to create an image pyramid. This approach was first explored by P. Viola and M. Jones in the context of face detection (VIOLA; JONES, 2001). This method provides fast multi-scale detection, but most of the useful image features for object detection cannot not be used in it (because they are not scale-invariant), limiting its applications.

2.1.3 The Fastest Pedestrian Detector in the West

The Dóllar detector combines the two previous approaches by calculating image features in a sparsely sampled image pyramid and approximating these features for all of the other scales of the feature pyramid.

What makes this approach work is the fact that it is possible to get accurate results for approximations of a great number of possible image features used in object detection (including the widely popular histograms of oriented gradients (DALAL; TRIGGS, 2005)), provided the source for the approximation is in a scale that is sufficiently close.

Extracting features from the octaves of the source image and approximating all other scales from those features makes this method orders of magnitude faster than the ones that use dense image pyramids and, according to the source paper, only sacrifices about one to two percent of detection quality. This is the main reason we chose this detector as a base for our work, we seek to combine good performance both in terms of detection quality and in terms of computational cost.

2.2 Perspective Camera Model

The perspective camera model projects 3D scenes into an image plane. The light rays captured by the lens pass through an aperture and meet on a single point inside the camera, the center of projection, causing a perspective (projective) transformation. This is the most used camera model in computer graphics and in computer vision because it closely relates to the human eye, where the lens would be the cornea, the center of projection would be the retina and the aperture would be the pupil.

2.2.1 The Projection Matrix

The distance between the center of projection and the image plane along the principal axis is the focal length f. The point where the principal axis collides with the image plane is called the principal point (u_0, v_0).

If the sensor is not perpendicular to the optical axis, the camera has a skew s. The image might also be scaled along both the x and the y axes, so we can represent the focal length as the tuple (f_x, fy) to cover that possibility. These are the camera's intrinsic parameters and are commonly represented by the calibration matrix K.

$$K = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The calibration matrix represents the camera's geometric and optical properties, to represent its orientation in the world we use a rotation matrix R and a translation vector t. The translation vector is the position of the world coordinate system's origin in camera coordinates and the rotation matrix holds the transformation necessary to align the axes of the two coordinate systems. The camera's extrinsic matrix is a 3x4 transformation matrix that holds R and t.

$$[R|t] = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \tag{2.2}$$

Now, we have all the necessary information to represent the translation of points in the

world into points in the image applying the following equation:

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = K[R|t] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{2.3}$$

where (u', v', w) is the resulting point in the image coordinate system and $(x, y, z, 1)$ is the source point in the world coordinate system, both are represented in homogeneous coordinates to simplify matrix calculations. In this representation, a coordinate is added to the points, so we extract the real image coordinates by dividing the first two coordinates by the last one:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u'/w \\ v'/w \end{bmatrix} \tag{2.4}$$

making (u, v) the recovered point in the image coordinate system.

The product between the intrinsic and extrinsic matrices of a camera (shown in equation 2.3) is its projection (or camera) matrix $P = K[R|t]$. The access to it is what we allude to when we talk about a *calibrated camera* in the context of computer vision. Also in equation 2.3, it is possible to know the location of a pixel in the world coordinate system if we set it to a plane. For the purpose of our work here, we want to know the location of pedestrians, assuming their feet are on the ground plane ($z = 0$), therefore we often calculate the position of image points in the ground plane using the following equation:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 0 \\ w \end{bmatrix}, \tag{2.5}$$

where $(u, v, 1)$ are the known pixel coordinates of the image point in question, in homogeneous coordinates; $(x, y, 0, w)$ will be the resulting point in the ground plane; and P is the camera's projection matrix. Since we know the resulting point has coordinate $z = 0$, we can suppress the third column of P and the third coordinate of the point in the world coordinate system. We end

up with

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{14} \\ P_{21} & P_{22} & P_{24} \\ P_{31} & P_{32} & P_{34} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = H \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}, \tag{2.6}$$

where H is the 3x3 homography matrix of the ground plane. The advantage of using this matrix is that it is invertible, so we can find the location of an image point in the ground plane with reduced computational cost using

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = H^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{2.7}$$

with $x = x'/w$, $y = y'/w$ and $z = 0$.

We can also calculate the world height of a bounding box by choosing a point in its top line (u, v_t) and a point in its bottom line (u, v_b). Assuming that the bottom point lies on the ground plane, its world coordinates $(x, y, 0)$ can be obtained using equation 2.7. For a person with height h_w assumed to be standing, equation 2.3 can be used to project the head point (x, y, h_w) back into image coordinates. Comparing the vertical coordinate of this point to v_t and solving for h_w leads to

$$h_w = -(v_b - v_t)/(P_{23}(H_{31}^{-1}u + H_{32}^{-1}v_b + H_{33}^{-1}) - P_{33}v_t(H_{31}^{-1}u + H_{32}^{-1}v_b + H_{33}^{-1})), \tag{2.8}$$

where h_w is the resulting world coordinate system height of the bounding box, P is the projection matrix of the camera and H^{-1} is the inverse of the homography matrix derived from the projection matrix.

2.3 Camera Calibration

In the last section, we showed what we are able to do once we have a calibrated camera. Here, we will briefly explore how to obtain the camera's projection matrix.

Pose estimation (also called extrinsic calibration) is the traditional approach to camera calibration. It requires a set correspondences between points in the image and their known locations in the world. The projection matrix can be estimated by solving a system of linear

equations knowing from equation 2.3 that

$$u_i = \frac{P_{11}x_i + P_{12}y_i + P13z_i + P_{14}}{P_{31}x_i + P_{32}y_i + P33z_i + P_{34}} \tag{2.9}$$

and

$$v_i = \frac{P_{21}x_i + P_{22}y_i + P23z_i + P_{24}}{P_{31}x_i + P_{32}y_i + P33z_i + P_{34}}, \tag{2.10}$$

where (u_i, v_i) are the pixel coordinates of point i, (x_i, y_i, z_i) are its coordinates in the world and P is the projection matrix to be estimated. At least six correspondences are necessary to compute all of the unknowns in P, using the direct linear transform algorithm (SUTHERLAND, 1974).

It is also possible to first obtain the camera's intrinsic matrix K to, then, estimate its pose in the world. In this case, the amount of necessary 2D to 3D correspondences falls to three.

This intrinsic calibration can be achieved with the use of patterns with known dimensions or (in the case where we have prior knowledge of objects in the scene) the image's vanishing points.

3 THE METHOD

In sections 3.1 through 3.4, we present our method. Figure 3.1 shows resulting images of each step of the method. They were generated on the first frame of the PETS 2009 data set (chapter 4), with a 2.0 resize factor (making the image twice the size of the original).

3.1 Selecting the Scales

The Dóllar method chooses the scales to which apply the classifier using the dimensions of the source image and of the windows expected by the classifier and a parameter that determines the smallest possible image. Depending on the nature of the source image and the parameters provided, this might result in several unnecessary scales being calculated/approximated and causes many applications of the classifier on windows which would contain objects with unrealistic real world dimensions (a pedestrian standing 7m tall, for example).

To prevent this, we factor in the provided camera projection matrix in this decision. First, we assume the source image (after applying the requested resize, which is also provided as a parameter) is the biggest image we will deal with; this and the fact that the image patches provided to the classifier have fixed dimensions make it so that in the first scale we are trying to detect the smallest possible pedestrians. The next step is to decide how much we need to shrink the current image in order for the classifier windows to be able to contain the tallest possible object (in the real world) in all parts of the image. That will be the last (and smallest) scale we deem necessary for detection.

The process works by assuming the camera is in an upright position (the ground plane is in the bottom), which makes it so the tallest possible objects (in pixels) are in the bottom of the image. Next, we find the biggest necessary bounding box in each of the bottom corners of the image by starting with the smallest possible (with the dimensions of the windows that will be analyzed by the classifier later) and gradually increasing its pixel height until its real-world height is bigger or equal to the maximum object height in the world coordinate system. We, then, choose the biggest of the two resulting values and the smallest necessary scale becomes the ratio between the height of the classifier window and it.

Knowing the smallest necessary scale of the source image we will use, we calculate all octaves between it and 1.0 (the biggest scale), where the octaves are the components of a set ordered from biggest to smallest in which the images are half the size of the previous element and double the size of the next one. The scales in-between octaves are approximated and the

ratio between their dimensions and those of the source image is decided by using the detector model parameter that describes how many scales should exist between two octaves. This way, the distance between scales is the distance between the previous and next octave, divided by the number of scales between octaves plus one.

As an example, consider the following scenario:

- The maximum object height in the world coordinate system should be 2.0m;
- The classifier expects images with 50 pixels of width and 100 pixels of height (h_0);
- The detector expects 3 scales between each pair of octaves, so we have 4 scales per octave;
- We have the camera's projection matrix (P), from which we derive its homography matrix (H);
- On the bottom left corner point a bounding box with 300 pixels of height (h_n) would have 2.1m of real-world height and that bounding box is taller (in pixels) than that found using the same procedure on the right corner.

The smallest necessary scale of the source image that would make a bounding box with 100 pixels of height contain the tallest possible object is

$$h_0/h_n = 100/300 = 0.3$$

and, from that, we know that the two octaves that we will calculate are 1.0 and 0.5 of the source image dimensions. And the approximated scales will be calculated using the following formula:

$$scale_i = scale_{i-1} - (previousOctave - nextOctave)/scalesPerOctave. \quad (3.1)$$

The resulting scale list (S) is

$$S = 1.0, 0.875, 0.75, 0.625, 0.5, 0.4375, 0.375, 0.3125.$$

Note that, in S, 1.0 and 0.5 correspond to the octaves and that the final scale is 0.3125 because the next one would be 0.25 and 0.25 is smaller than our last necessary scale (0.3). Therefore, smaller scales would not be useful for us in this particular camera setup.

3.2 Reducing Classifier Applications

Once we have the calibrated image pyramid, we reduce the amount of times the classifier is used by dividing the detection phase into two steps. The first one, explained in subsection 3.2.1, is called the sparse step and consists of applying the classifier to windows that have considerably large spacing between each other with the aim of covering all the parts of the scene which might contain a detectable pedestrian while keeping the candidate patches at a minimum. The detections found in this step will not be high quality ones, but they serve as basis for the dense step. The dense step (in subsection 3.2.2) will generate more densely spaced candidates around the resulting detections from the sparse step. This last step aims to provide detections with the same quality as a generic method, while keeping the number of classifier applications low because we only expand regions where there were detections in the previous step, instead of the whole image.

3.2.1 Sparse Detection

Knowing the dimensions of the smallest possible bounding box (w_0, h_0), we start the first line of new candidates at the point $(0, h_0)$ of the source image. For every new base point, we save all the candidate bounding boxes that have height h_i such that $h_i \geq h_0$ in the image coordinate system and height h_w that is smaller or equal to the maximum object world height and bigger or equal to the minimum object world height in the world coordinate system. The procedure is repeated stepping the base point $w_0/2$ to the right in each line and $h_0/2$ down at the end of a line; this is done so we guarantee that the candidates would share, at least, half of their area with adjacent candidates, which gives us good coverage of the image even with a small number of candidate patches when compared to the generic sliding window method.

Since we assume the same camera for the whole set of images, the sparse candidate generation step is done only for the first frame of the sequence and the resulting candidates are stored in order to be used in all other frames.

The classifier is, then, applied to all of those candidate patches. All detections outside an existing dense candidate region are passed to next step (on the first frame of a sequence, all detections go through).

3.2.2 Dense Detection

The detections from the first stage of our classifier are sorted by score in descending order and all detections that are near a detection with higher score are suppressed. We decide if a detection is near enough another and needs to be suppressed taking in consideration the fact that the dense region we will create around a detection has triple the base bounding box's width and double its height; so, if two bounding boxes would be inside the same region, we keep the one with the highest score. We also restrict the dimensions of the dense regions so as to avoid overlap between them.

After deciding which regions should be created, we generate candidates inside them by saving all the candidates with valid world and image coordinate systems heights using steps of one pixel (or the value of the detector model's shrink factor, which makes the images in the pyramid smaller to reduce computation cost, when used) for both the x and y coordinates of the image and apply the classifier to the resulting candidates.

3.3 Two-step Suppression

Just like in the Dóllar method, the resulting detections are then submitted to a non-maximal suppression step. It works by grouping together all detections that share a overlap value that's higher than the threshold provided as a parameter and keeping only the detection with the highest score in each of those groups.

In our method, though, we submit the results of the first suppression step through another one. This one works by comparing each detection to another one with higher score and removing all of those that have a overlap ratio that is higher than the ratio between the lower and the higher classifier score. This takes away false positives that remain after the first suppression step.

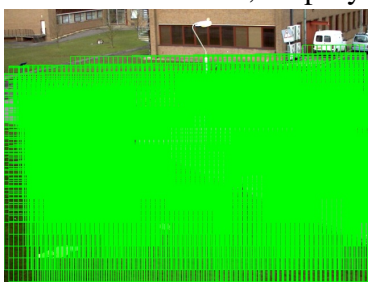
3.4 Dense Region Removal

In the last step of the method, we remove all of the dense candidate regions in which there are no detections in the current frame. This makes it so we don't keep dense regions in which we don't expect to find target objects in the future.

This step could be adjusted if we took into consideration the frame rate of the source

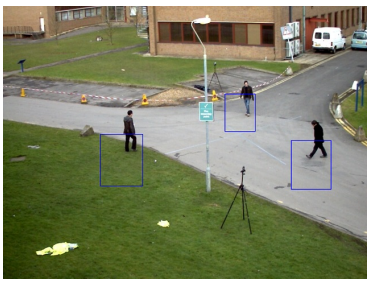
Figure 3.1: The Method, Step-by-Step

(a) Source Image

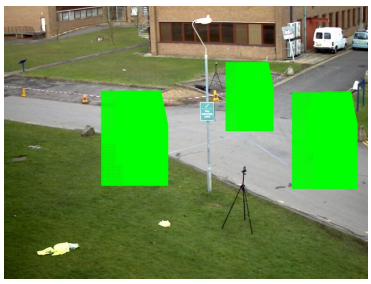


(b) Sparse candidates

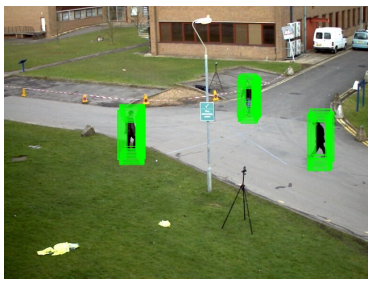
(c) Sparse step detections



(d) Dense regions



(e) Dense candidates



(f) Detections

(g) Detections after the first suppression

(h) Detections after the second suppression

footage. We, in that case, could decide a number of frames that should elapse between applications of the dense region removal. Other possibility would be to count the amount of frames where there were no detections inside a region and only remove it when that number becomes higher than a threshold. In our experience though, this step didn't cause too much overhead and, as such, we execute it on every frame.

4 RESULTS

For testing purposes, we created a generic version of our algorithm (and, thus, call the method explored in our work here "calibrated" to contrast with it). The generic version follows the same rules the Dóllar method does when selecting the feature pyramid's scales, generating candidate image patches for detection and suppressing detections. The only reason we do not call it the Dóllar method is because we couldn't replicate the results in the feature extraction portion of the algorithm. Our findings, though, are not affected since this situation happens for both the calibrated and generic methods, making comparisons fair.

The INRIA detector model, provided as part of the Dóllar Matlab toolbox, served as the classifier for both detector versions in all of the testing process. This classifier expects candidate patches of 100x41 pixels as input, we decided which "resizes" would be applied to the original images based on the fact that the classifier works best if we have pedestrians that are not smaller than that.

The other columns displayed in our Method Comparison tables are:

- *Total Scales (Octaves):* The amount of scales of the original image (after resizing) in which features are extracted to form the feature pyramid and the total octaves contained between those scales. The octaves are the scales for which the features are actually calculated, making those scales much more computationally expensive.
- *CApF:* Classifier Applications per Frame; represents the amount of candidate patches evaluated per image in the tested sequence. For the generic method, this number is constant throughout all sequences and exactly the same as the Dóllar method produces. For our method, the calibrated one, this number is an average since the classifier applications vary between frames.
- *TPR:* True Positive Rate; using ground truth data, we calculate the ratio between detected pedestrians and the total amount of pedestrians in the sequence. We deem a pedestrian to be detected if there is one detection that has Jaccard Coeficient (JACCARD, 1901) over 0.5 when compared with the bounding box for that pedestrian in the ground truth. The Jaccard Coeficient is the ratio between the size of the intersection and the size of the union of two sample sets (S_1 and S_2, which, in this case, are rectangles in the image):

$$J = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \tag{4.1}$$

- *FPpF:* False Positives per Frame; serves as complement to the True Positive Rate metric

since it would be easy to cover all pedestrians if we allow there to be massive amounts of false positives. The perfect pedestrian detector would have TPR 1 and 0 FPpF and a good result for a pedestrian is achieved by finding a satisfactory compromise between these two metrics.

In table 4.1, we document the results found for the PETS 2009 data set (NEVATIA, 2012a) out of which we used a sample of 300 images. We also used a sample of 300 images of the Town Centre data set (BENFOLD; REID, 2009) to create table 4.3. On the TUD Stadtmitte (NEVATIA, 2012b) (results on table 4.2), we were only able to find 179 images for testing. Figures 4.1, 4.2 and 4.3 show graphic versions of the data contained in those tables. Table 4.4 documents the time it takes for each method to process one frame, figure 4.4 has plots of the results contained in this table. We also provide graphics containing the execution time of our algorithm split between each of its parts in figure 4.5.

Note that we were not able to locate the actual ground truth data for the PETS 2009 data set. The data we use as ground truth for the experiments in the data set is actually the result of manual annotations in key frames with trajectory interpolation for all the other frames (found in (MILAN; SCHINDLER; ROTH, 2013)). It is not a perfect method and, because of that, some correct detections are documented as false positives.

In figures 4.6, 4.8 and 4.10, we display comparisons between the detections in our method (top images) and those found while using the Dóllar detector's original code (bottom images). We did not capture numeric comparisons because the classifier in our code performs worse due to the anomalies found in the feature extraction functions that were adapted to our code from Dóllar's to reduce the development time. Also, to be able to make such comparisons, it would be necessary to make slight modifications to that code. So we decided to make all comparisons inside our code and with a classifier that performs equally for both methods. The images provided, nevertheless, give us an idea of how our method is capable of improving the results of a detector.

Table 4.1: Method Comparison in the PETS 2009 Data Set

Method	*Resize*	*Total Scales (Octaves)*	*CApF*	*TPR*	*FPpF*
Calibrated	1.5	11(2)	24527	0.68	0.40
Generic	1.5	25(4)	299973	0.85	5.9
Calibrated	2.0	14(2)	65404	0.84	0.54
Generic	2.0	29(4)	570135	0.89	7.2
Calibrated	2.5	16(2)	98573	0.86	0.71
Generic	2.5	31(4)	927349	0.88	8.4

Table 4.2: Method Comparison in the TUD Stadtmitte Data Set

Method	*Resize*	*Total Scales (Octaves)*	*CApF*	*TPR*	*FPpF*
Calibrated	1.0	16(2)	60405	0.72	1.37
Generic	1.0	19(3)	73888	0.71	4.55
Calibrated	1.5	21(3)	165793	0.74	1.11
Generic	1.5	23(3)	196871	0.73	5.28
Calibrated	2.0	24(3)	283121	0.74	1.92
Generic	2.0	27(4)	380700	0.74	5.34

Table 4.3: Method Comparison in the Town Centre Data Set

Method	*Resize*	*Total Scales (Octaves)*	*CApF*	*TPR*	*FPpF*
Calibrated	0.5	5(1)	7739	0.13	0.62
Generic	0.5	20(3)	134982	0.30	5.87
Calibrated	0.75	9(2)	37444	0.39	4.47
Generic	0.75	25(3)	349444	0.58	13.01
Calibrated	1.0	13(2)	96323	0.47	5.23
Generic	1.0	28(4)	666870	0.59	16.68

Table 4.4: Execution Time Comparison Between Methods

Dataset	*Resize*	*Frame Time(Calibrated)*	*Frame Time(Generic)*
PETS	1.5	0.167 s	0.232 s
PETS	2.0	0.315 s	0.432 s
PETS	2.5	0.499 s	0.673 s
TUD	1.0	0.118 s	0.086 s
TUD	1.5	0.269 s	0.176 s
TUD	2.0	0.514 s	0.347 s
Town Centre	0.5	0.071 s	0.153 s
Town Centre	0.75	0.217 s	0.365 s
Town Centre	1.0	0.340 s	0.609 s

Figure 4.1: PETS 2009 Data Set Comparison Plots

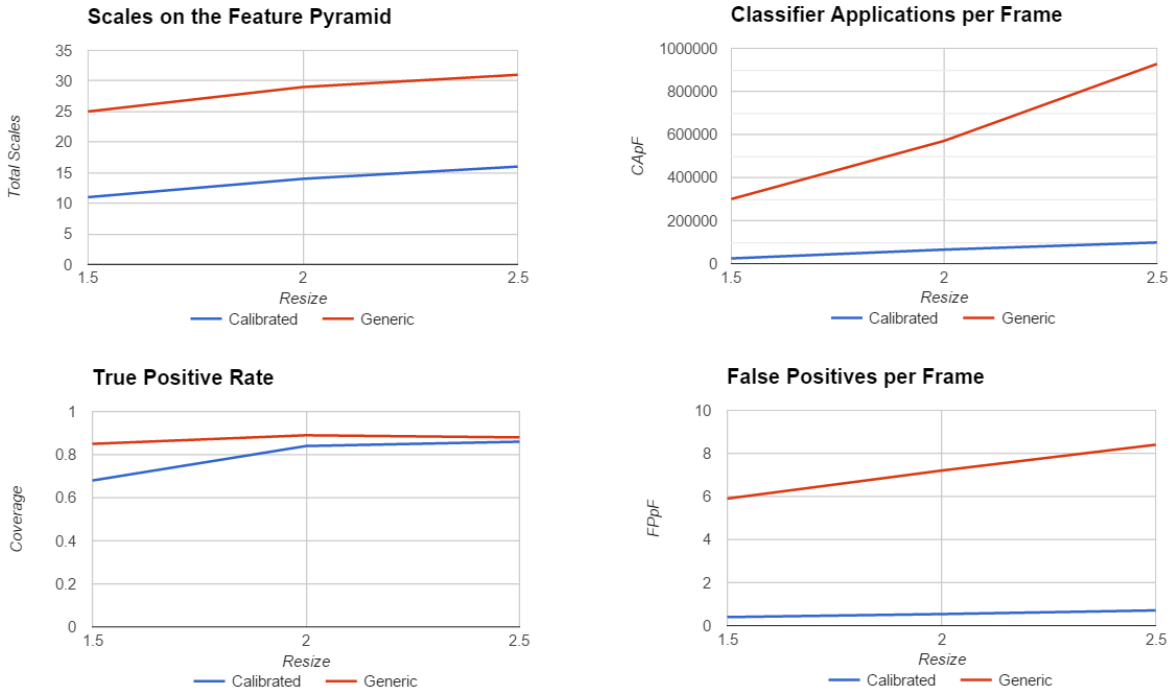


Figure 4.2: TUD Stadtmitte Data Set Comparison Plots

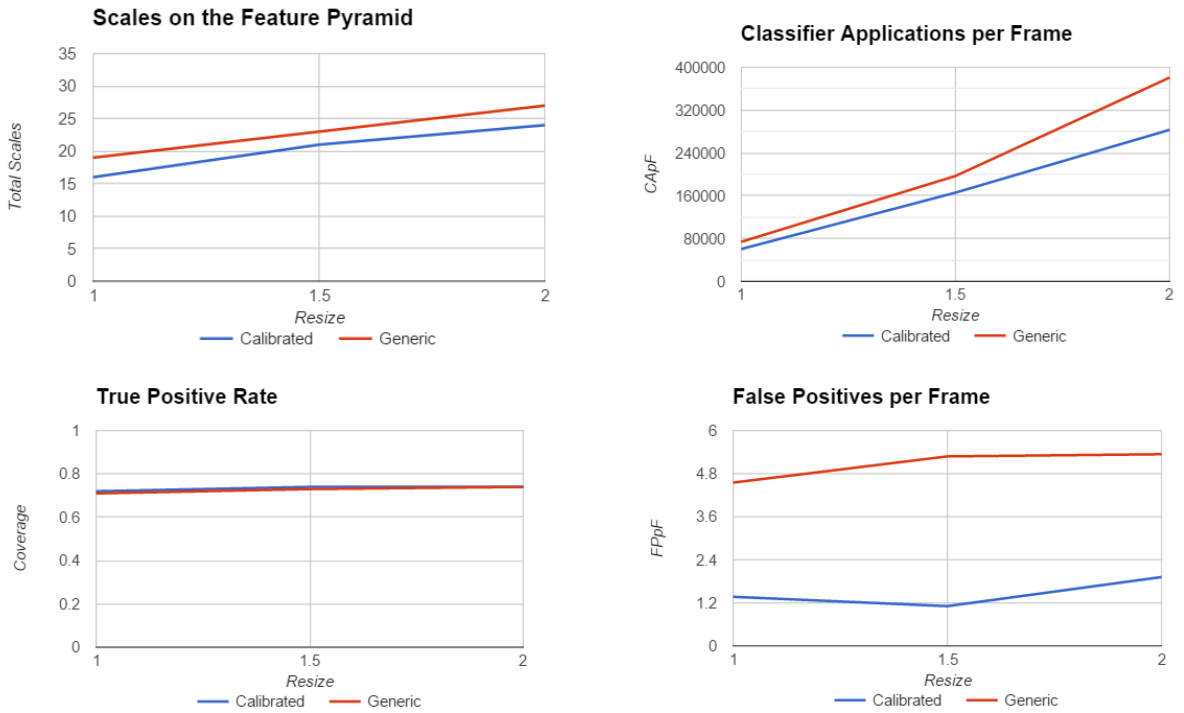


Figure 4.3: Town Centre Data Set Comparison Plots

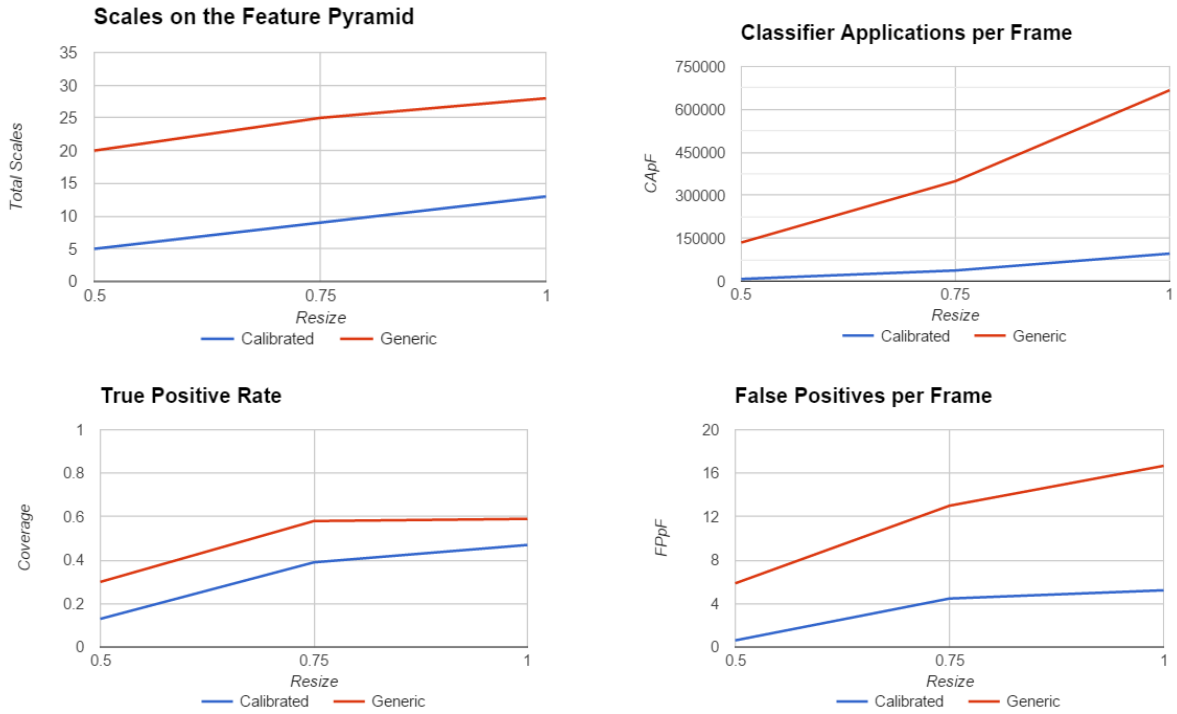


Figure 4.4: Execution Time Comparison Plots

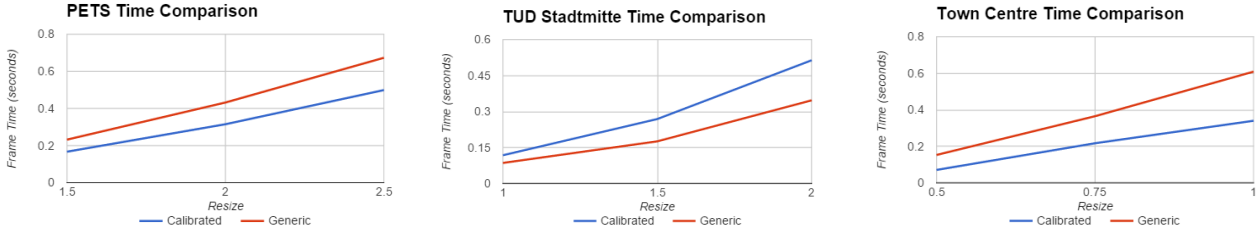
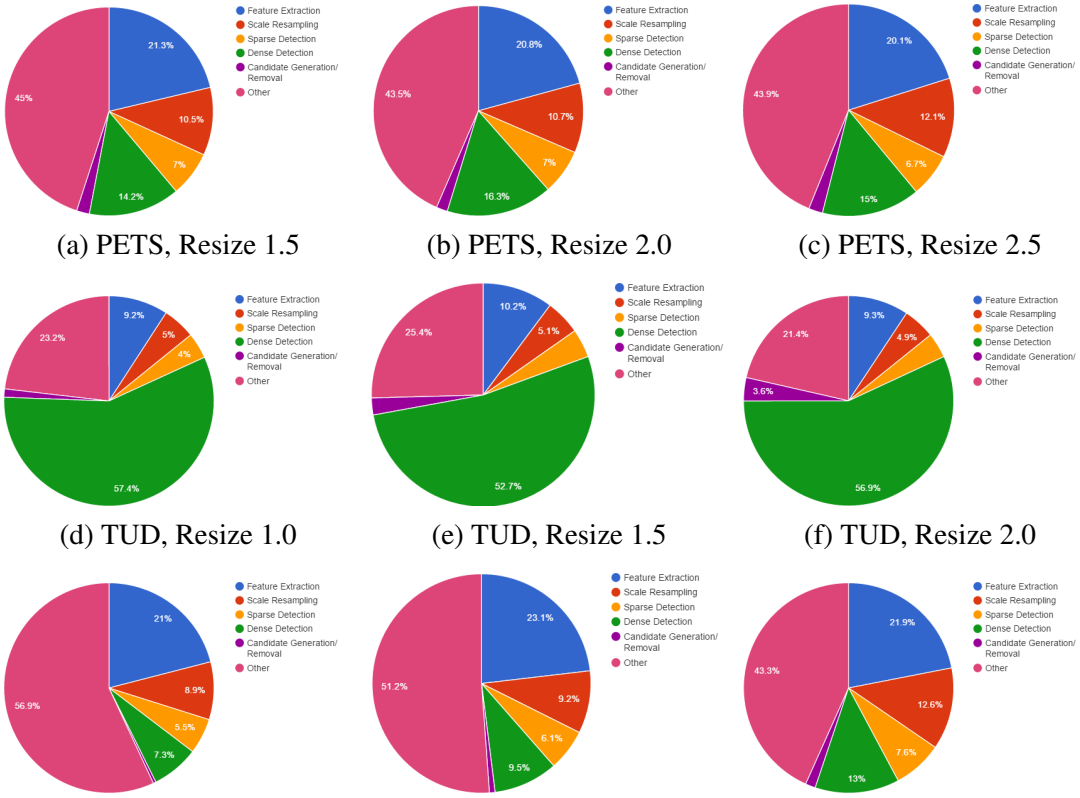


Figure 4.5: Time Portion Spent on Each Part of the Algorithm



(a) PETS, Resize 1.5 (b) PETS, Resize 2.0 (c) PETS, Resize 2.5

(d) TUD, Resize 1.0 (e) TUD, Resize 1.5 (f) TUD, Resize 2.0

(g) Town Centre, Resize 0.5 (h) Town Centre, Resize 0.75 (i) Town Centre, Resize 1.0

The feature pyramid creation is comprised of calculating scales (*Feature Extraction*) and using those results to approximate the rest of the scales (*Scale Resampling*). *Sparse Detection*, *Dense Detection* and *Candidate Generation/Removal* compose the detection step of the algorithm. The *Suppression* field was aggregated automatically into the *Other* field because not enough time was spent on this phase.

Figure 4.6: Sample images of our method on the PETS 2009 data set.

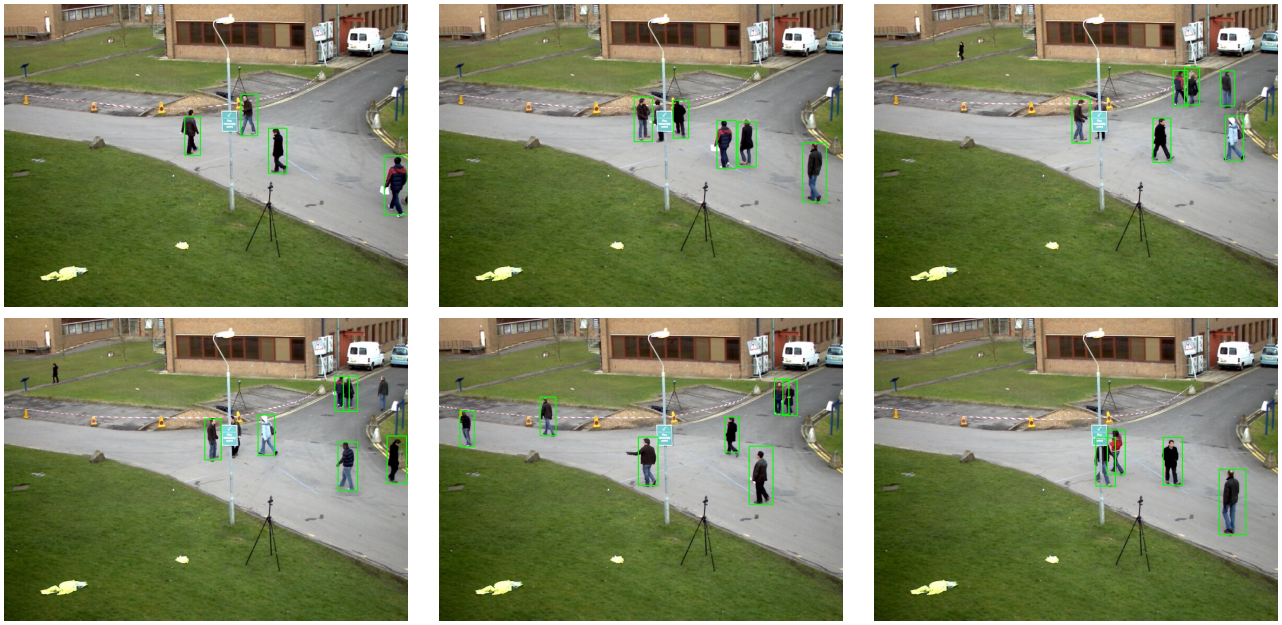


Figure 4.7: Sample images of Dóllar's method on the PETS 2009 data set.

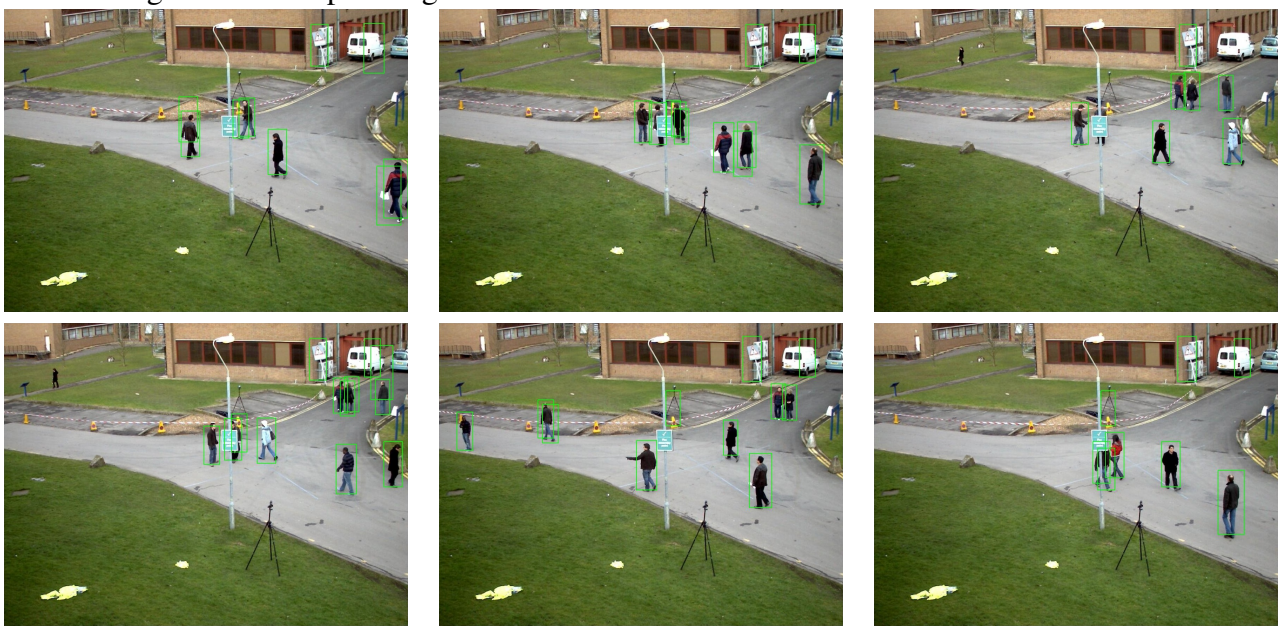
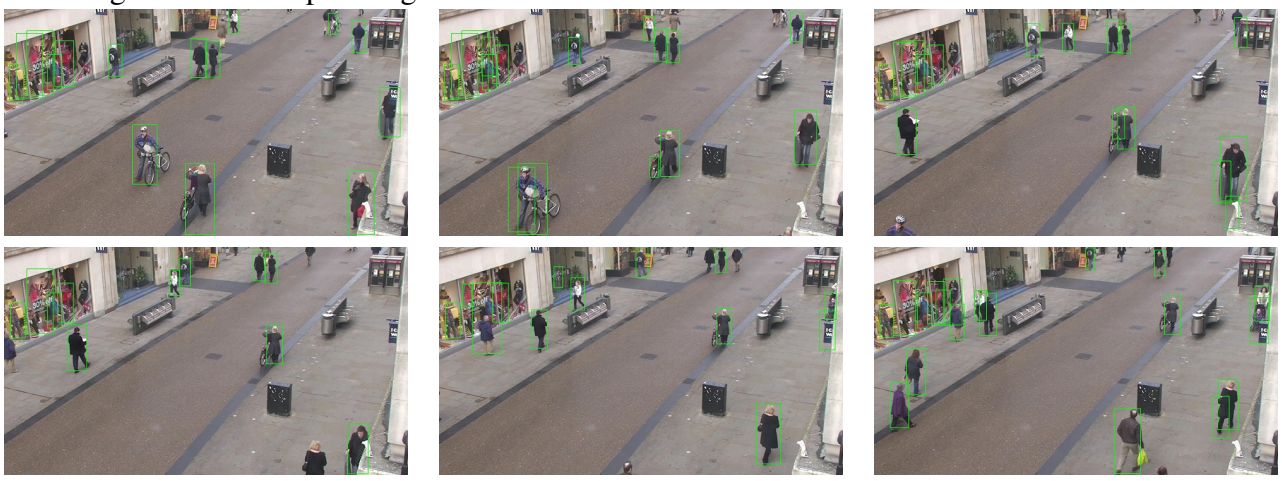


Figure 4.8: Sample images of our method on the TUD Stadtmitte data set.

Figure 4.9: Sample images of Dóllar's method on the TUD Stadtmitte data set.

Figure 4.10: Sample images of our method on the TUD Stadtmitte data set.

Figure 4.11: Sample images of Dóllar's method on the TUD Stadtmitte data set.



5 CONCLUSIONS

We were able to dramatically reduce the amount of classifier applications per frame in both data sets were the camera is positioned well above the ground plane (PETS 2009 and Town Centre). In the TUD Stadtmitte data set, though, we saw just a small reduction to this number because of the camera's angle in relation to the ground plane, which makes it so more areas of the image are plausible to contain pedestrians and, as a result, increases the number of candidate patches generated by our method.

Again, the angle of the camera was important to determine the reduction in the amount of scales in the image features pyramid with the two data sets with the higher camera seeing the biggest reduction.

The number of false detections dramatically decreased across the board, which compensates for the reduction in detection coverage seen in two out of the three tested sequences. The biggest loss in detection coverage when compared to the generic algorithm happened in the Town Centre data set, but the generic method also generated the biggest amount of false positives when applied to it.

The gains in execution time were not compatible with the reduction in scales of the feature pyramid and in classifier applications due to the more complex data structure we use to keep the candidate image patches and the overhead caused by adding, removing and accessing its information. It is possible that more work towards the overall performance of the code could provide a more favorable comparison towards the generic method in this area. These higher-level data structures where used in order to have more legibility in the code during development.

We believe adjustments could be made to increase the amount of candidates in the sparse phase and to reduce the density of the dense candidate regions. This change could improve the detection coverage while keeping the gains in the total number of classifier applications and could also solve the problem with cameras that are parallel to the ground.

Overall, though, the results were satisfactory based on the reduction in computational costs and the better detection quality caused by the cut in false positives.

REFERENCES

BENENSON, R. et al. Ten years of pedestrian detection, what have we learned? In: **ECCV**. [S.l.: s.n.], 2014.

BENFOLD, B.; REID, I. Guiding visual surveillance by tracking human attention. In: **Proceedings of the 20th British Machine Vision Conference**. [S.l.: s.n.], 2009.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: SCHMID, C.; SOATTO, S.; TOMASI, C. (Ed.). **International Conference on Computer Vision & Pattern Recognition**. INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334: [s.n.], 2005. v. 2, p. 886–893.

DOLLÁR, P.; BELONGIE, S.; PERONA, P. The fastest pedestrian detector in the west. In: **BMVC**. [S.l.: s.n.], 2010.

JACCARD, P. Étude comparative de la distribution florale dans une portion des alpes et des jura. **Bulletin del la Société Vaudoise des Sciences Naturelles**, v. 37, p. 547–579, 1901.

MILAN, A.; SCHINDLER, K.; ROTH, S. Challenges of ground truth evaluation of multi-target tracking. In: **Proc. of the CVPR 2013 Workshop on Ground Truth - What is a good dataset?** [S.l.: s.n.], 2013.

NEVATIA, R. Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. In: **Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Washington, DC, USA: IEEE Computer Society, 2012. (CVPR '12), p. 1918–1925. ISBN 978-1-4673-1226-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2354409.2354940>.

NEVATIA, R. An online learned crf model for multi-target tracking. In: **Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Washington, DC, USA: IEEE Computer Society, 2012. (CVPR '12), p. 2034–2041. ISBN 978-1-4673-1226-4. Disponível em: <http://dl.acm.org/citation.cfm?id=2354409.2355020>.

SUTHERLAND, I. Three-dimensional data input by tablet. **Proceedings of the IEEE**, v. 62, n. 4, p. 453–461, abr. 1974.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 1st. ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN 1848829345, 9781848829343.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: . [S.l.: s.n.], 2001. p. 511–518.