Image based Wheel Detection using Random Forest Classification

Karin Hultström

Master's thesis
2013:E7

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

The aim of this master thesis is to detect and recognise wheels in images by means of image analysis. This could later on serve as a foundation for a safer vehicle counting and classification method than those currently in use that requires personnel to cross the lanes on installation.

The general layout of the classification system consists of five stages: multi-scale transformation, window extractor, pre-processing, classification and cluster analysis. In order to obtain the training and testing data for evaluation and construction of the system, images that illustrate moving cars on a road are acquired. From these, several positive and negative windows are extracted that visualizes wheels and non-wheels. For the classification stage, the learning algorithm used is Random Forest. Moreover, with the Random Forest as the foundation, two different concepts were introduced to further improve the predictions. These are referred to as bootstrap configuration and cascading classification.

The results are evaluated be means of Receiver Operating Characteristics and contingency tables. In this master thesis, the final system produces a satisfying result based on the false positive rate and true positive rate. For future development, the amount of examples in the training data could be increased in order to gain more knowledge in the teaching of the classifier. Furthermore, an optimization of the program could lead to faster execution time, which is a requirement if this system is to operate in real-time. To conclude, the system produces a satisfying result for wheel detection that can be used as a foundation when constructing a general system for vehicle counting and classification.

Acknowledgement

This master thesis was done in collaboration between COGNIMATICS and the Mathematical Imaging Group at The Faculty of Engineering at Lund University (Lunds Tekniska Högskola, LTH). Therefore, I would like to thank Rikard Berthilsson, CEO at COGNIMATICS, for giving me the opportunity of doing my master thesis with them and all the good advices I have received during this process.

Furthermore, I would like to thank my supervisors Professor Karl Åström and Postdoctoral Fellow Håkan Ardö at LTH, for their support and guidance throughout my entire master thesis. Finally, I would like to thank my family and friends for their constant support.

Contents

Chapter 1

Introduction

Counting the number of vehicles driving on a road is a task that could be evolved from its current methods by using image analysis. The Swedish Transport Administration (Trafikverket) is responsible for the construction, operation and maintenance of all state owned roads and railways in Sweden. One of their many assignments is to register the traffic flow on certain roads. When the intended traffic registration is only for a limited period, they use a traffic analyser called Metor 2000, with rubber hose sensors [1]. Otherwise, they use more sophisticated sensors submerged in the ground.

The method used by the rubber hose sensors is quite simple. The sensors are laid out across the road with a combined analyser connected to one of the ends of each rubber hose sensor. When a vehicle passes over one of these sensors, an air-impulse is created and moves through the hose until it reaches the analyser, which register the time of the impulse. By using several rubber hoses after one another the direction and speed of the vehicles can later on be calculated when the analyser is connected to a computer with the necessary software.

The device can also distinguish the vehicles in 15 predefined vehicle classes. This is done by studying, based on the centre of each wheel, every distance between two subsequent wheels belonging to a vehicle. By comparing these generated distances with a known table, a vehicle can be assigned to a corresponding vehicle class. Therefore, it is important to know the location of each wheel in order to classify a vehicle.

There are many benefits from using this method, for instance, it is only the time that are registered by the analyser, the examination of the material is done later by a software program. Therefore, it is always possible to update the software to the latest versions which contributes to flexible and more secure data analysis. Also, the rubber hose is a strong and inexpensive material and the simplicity of

the method makes it all affordable.

Although, given all these benefits, there are some disadvantages with this method that is the underlying reason for why this system could be evolved with some other technique. Mainly, it concerns the interruption of the traffic when placing out the rubber hoses and the safety of the personnel. If the examined road is highly crowded with traffic, an interruption may cause long traffic jams. Furthermore, there is always some sort of risk when personnel are working in traffic, even though the traffic may be diverted elsewhere.

A solution to these complications could be to place a camera on the side of the road, filming the traffic from the side and later on analysing the film by means of image analysis. This would mean that there is no need to interfere with the traffic and the above stated disadvantages could be eliminated. Furthermore, the benefit of updating the software to the latest versions would still be feasible. Therefore, the aim for this master thesis is to generate a system for detection and recognition of wheels that further on can be used as a foundation for vehicle counting and classification systems.

1.1 Related work

Many related works has been written within the area of vehicle detection due to its wide application area. Besides using vehicle detection for counting and classification systems, as this master thesis is based on, it can be useful in evaluating traffic parameters [26], driver assistance systems [2, 15, 21] or surveillance systems [25].

There are many different techniques used when detecting vehicles, not always by means of image analysis. As an example there is inductive loop detector [14, 5] and RF signal strength information [22]. Nevertheless, many of the related work are founded on image-based techniques for instance: background subtraction [25, 18], edge detection [31], corner detection [23] or other feature detection [35].

In the article [28], by Sun, Bebis and Miller, they use Support Vector Machines as the learning algorithm for vehicle detection. However, in this master thesis, the learning algorithm Random Forest will be used, as described in Section 3.4. Furthermore, Gabor filters are used for feature extraction while this work use normalized intensity and Local Binary Pattern, see Section 3.1.

Setchell and Dagless presents in their article [27], a method of utilizing the already existing video cameras in the road networks for vehicle detection. By a combination of these retrieved images and computer vision, they create a

model-based technique for estimating the vehicles position, orientation and size in the real world. Therefore, their work is based on recognizing vehicles by searching correspondence space. However, in this master thesis the object detection and recognition is made directly from the images. Due to the side view positioning of the camera, only the wheels of the vehicle needs to be identified instead of the entire vehicle.

In a similar manner to this work, Achler and Trivedi also considered side view images in their article [2] in order to locate the position of the vehicle. Though, the detection is made from an already moving vehicle and should function as a driving aid to avoid objects. Furthermore, they adopt another technique than the one used in this work, by processing the images using a difference of Gaussian filterbank.

1.2 Structure of the report

Chapter 2 gives an introduction to the area of object detection in image analysis. It begins by introducing the layout of the system that is constructed in this master thesis. Thereafter, it describes what binary classification means. Furthermore, some of the challenges of object detection are mentioned. On account of this, a definition is made of how a database could be constructed. Finally, the methods used for evaluation of the results in Chapter 5 are described.

Chapter 3 consist of the background theory required for the understanding and construction of the system. First, the different feature extractions are defined. Thereafter, decision trees and bootstrap aggregating are described. Both of these serve as a foundation for the learning algorithm used in this master thesis, Random Forest, which is described afterwards in this chapter. At the end of the chapter, two different methods used for improving the predictions are mentioned.

Chapter 4 describes the construction of the system generated in this work in more detail. The sections are based on the system layout description from Chapter 2. Therefore, the chapter starts with explaining how the input images are generated. From there, it moves on through the different stages until it reaches the last stage of the system, which is cluster analysis.

Chapter 5 corresponds to the results obtained when the system is evaluated through different improvement methods. Moreover, a comparison is made of these results and the main properties of the methods are summarized. Finally, a specific input image is studied which visualizes the results even further.

Chapter 6 contains the conclusions that can be drawn from this master thesis. Furthermore, it suggests some improvements than can be made on the system in order to obtain even better results in future work.

Chapter 2

Object Detection

One of the main building blocks for computer vision and image analysis is to detect objects in an image and assign them to their corresponding class label. This is referred to as *object detection and recognition*. The application of this method reaches through many fields, for instance tracking [24], face detection [20] and video surveillance [16]. In order to attain an appropriate layout of how object detection and recognition should work, similarities are drawn to the human mind. A person who is looking at its surroundings can identify different objects regardless if there are many other distractions in the view and even if some parts of the object is out of sight. Furthermore, if the object are of a different size, shape or is rotated, the human mind can usually still identify remarkably easy what kind of object it is.

The main stages of how this works in a brain can be summarized by; an image is evaluated through comparison of previous recognized objects and an appropriate classification is returned. These stages can be used as key insights to how the framework of object detection and recognition could be constructed. However, it is a demanding process to teach a computer to detect and identify an object which therefore typically requires a few more stages.

2.1 The standard architecture

The standard architecture for detection and recognition of a specified object is illustrated in Figure 2.1. All of these stages are described in more detail in Chapter 4. The main idea is to send in a complete image as input to the system, thereafter a *multi-scale transformation* is made. This alters the image by rescaling it several times by a predefined factor. Each scaled image is sent

further on to the next stage, the *window extractor*. This stage is provided with a small square that scans each scaled image. At every possible position in the scaled image, it extracts a window of the same size as the square. As a result, an enormous amount of equally sized windows are created for further evaluation in the system. The next stage concerns the *pre-processing* that is made on each window. The main purpose of the stage is to make a feature extraction from every window. This can be considered as an informative description of each window. Thereafter, the result is converted to match the requirements as an input variable to the next stage.

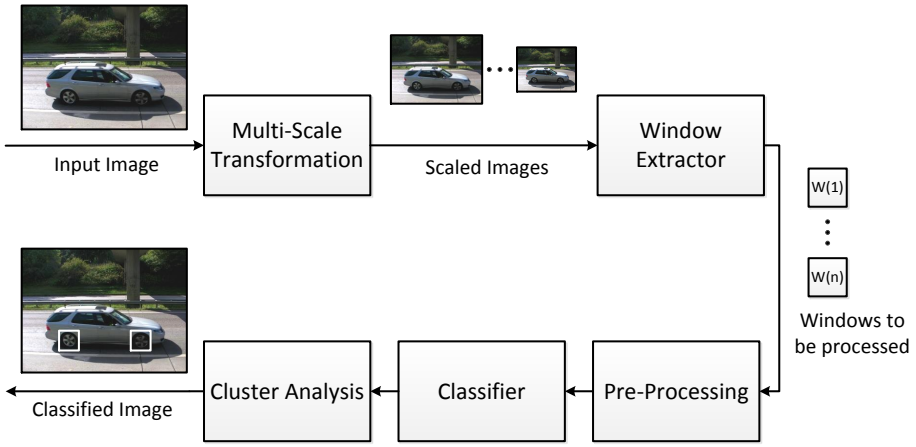

Figure 2.1: Visualization of the standard architecture.

At this moment, the image has reached the *classifier* which performs the main analysis of the object detection and recognition. By means of a given classification method, each window is assigned a class label. Thus, identifying if a window contains the specified object or not. Usually, an image contains an enormous amount of non-object windows in comparison to the desired object windows.

Afterwards, the system enters the final stage corresponding to the *cluster analysis*. Here, the extracted windows classified as the object are re-entered to their corresponding scaled image. Due to the previous window extraction, several windows may match the same object. Therefore, if several windows overlap with a certain degree, these are clustered together and thus represent the same object. When the final stage is completed, the image is returned and if the classifier has found one or several objects, these are also marked out.

2.2 Binary classification

If a classifier only have two possible categories for classification assignment, corresponding to if the extracted window has the same properties as the searched object or not, then this is referred to as a *binary classification problem*. Thus, the aim for a binary classifier is to find a decision function that separates the true objects from the non-objects as good as possible. There are a numerous different methods for training a classifier to achieve this goal, for instance support vector machines [11, 32] or decision trees [12, 33]. What they all have in common is that they require a database of both true objects and non-objects to learn from.

2.2.1 Database description

The aim of a database is to collect a set of images that reflects the properties of both the specified object and other non-objects for the classifier to train on. The challenge is that the searched object does not always look exactly the same in every image. These difficulties, due to visual variations of the searched object and its surroundings, are usually divided into two parts: *intrinsic* and *external variability*.

The intrinsic variations concern the object itself, for instance there might be large variations between different objects of the same class or variations of a particular object. The external variability describes differences in the object as a result of its environment. There are several examples of this, such as, differences in illumination, scale or occlusion. The training set needs to contain as many of these cases as possible in order to gain a functional classifier. Furthermore, the non-objects should describe as many cases as manageable that do not resemble the object.

Images representing the object are referred to as *positive windows*. In a similar manner, images representing the non-objects are denoted as *negative windows*. Moreover, these two categorise are assigned a binary value to simplify the class label representation. Thus, an image is either referred to class label one, positive window, or zero, negative window. In the creation of the database, when there is no classifier to use, these assignments are done manually. By means of the pre-processing stage, each positive and negative window receives a feature vector, see Section 3.1. In a combination of the binary class label and the feature vector, the complete database can be defined as follows.

Definition 2.1. *A **database** $\mathcal{X}$, for training a classifier is represented by*

$$\mathcal{X} = \{x_i, y_i\}^N,$$

where x_i are the features extracted from window number i, represented as a feature vector and $y_i \in \{0, 1\}$ are the corresponding binary valued class label. $N > 0$, is the number of windows in the database.

When the assembly of the database is complete, the classifier can use it during its learning process for retrieving an efficient decision function.

2.3 Evaluation of results

The results in this master thesis are evaluated using *Receiver Operating Characteristic Analysis* (ROC). This is a method used for analysing the performance of a classifier by means of graphical studies. The full complexity of this method is described in [30] and [17]; nevertheless, those definitions that will be used in the results in Chapter 5, are summarized in the next Section 2.3.1.

2.3.1 Receiver Operating Characteristic

Receiver Operating Characteristic is used when the classification problem is binary. As previously mentioned, it means that each example has two possible classes to choose between. When a positive window is sent in to the classifier it could either get correctly classified as a positive window or misclassified as a negative window. These two cases are called *true positive* (TP) and *false negative* (FN). Similar definitions are applied to a negative window. That is, if a negative window obtain a correct classification it is called a *true negative* (TN); otherwise, if it gets misclassified as a positive window it is called a *false positive* (FP). All these cases can be collected in a tabular known as a *contingency table* or a *confusion matrix*, see Table 2.1.

		PREDICTED CLASS		
		Positive	Negative	Total
ACTUAL CLASS	Positive	TP	FN	P
	Negative	FP	TN	N
	Total	P'	N'	

Table 2.1: A contingency table with the different cases entered.

In order to create an ROC graph, a pair of statistical measures of the classifiers performance needs to be calculated using the previous definitions summarized in Table 2.1. These statistics are the *true positive rate* (TPR) and the *false positive rate* (FPR). TPR is also known as *hit rate* or *sensitivity*, and describes the classifiers ability to identify positive windows. Mathematically, it is the fraction of true positive out of all the examples which actual class are positive. Analogously, FPR is the portion of false positive out of all the examples that belong to the negative actual class. Another name for FPR is *false alarm rate*. The equations for TPR and FPR can be written as follows,

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}, \tag{2.1}$$

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}. \tag{2.2}$$

The false positive rate is closely connected to the *specificity*, also known as *true negative rate* (TNR), simply by the equation:

$$TNR = 1 - FPR = 1 - \frac{FP}{N} = \frac{TN}{N}. \tag{2.3}$$

TNR outlines the ability of the classifier to identify negative windows; therefore FPR displays how many negative images that have been misclassified.

In a similar manner, the *false negative rate* (FNR) is closely related to the true positive rate by the following equation:

$$FNR = 1 - TPR = 1 - \frac{TP}{P} = \frac{FN}{P}. \tag{2.4}$$

The ROC method uses the equations 2.1 and 2.2 to create a two-dimensional graph, with the false positive rate on the x-axis and the true positive rate on the y-axis. On account of this, when a classification round has ended, a contingency table is generated and a point corresponding to this specific table is marked out in the ROC graph. This point symbolises therefore the balance between the successful classification of positive windows and the misclassification of negative windows.

By evaluating every possible threshold that separates the classification into two classes, a continuous line can be illustrated in the ROC graph. However, for this master thesis only some specific classification cases are evaluated. As a consequence, the ROC graph will only contain discrete valued points.

If the classifier would use random guessing for its decision making, then the points in the ROC graph will end up somewhere on a diagonal line known as *line of no-discrimination*, see Figure 2.2. This line reaches from the upper right corner down to the origin. Any point below this line would then be worse than just a simple random guess.

Consider for instance point A in the graph that is located below the line of no-discrimination. It corresponds to a false positive rate of 50% and a true positive rate of 20%. This means that in 8 out of 10 cases, when it should be classified as positive it becomes misclassified as negative. However, if a classifier with these properties inverts its result, then it would be located above the line. In the graph, the point A has been inverted and the result is illustrated by point B. This corresponds to a classifier that performs a prediction better than a random guess.

Nevertheless, any point located above the line of no-discrimination could for this reason be interpreted to be better than a random guess. However, when there are several points located above this line, for instance point B and point C, some other properties needs to be considered when evaluating which classifier that result in the best performance.

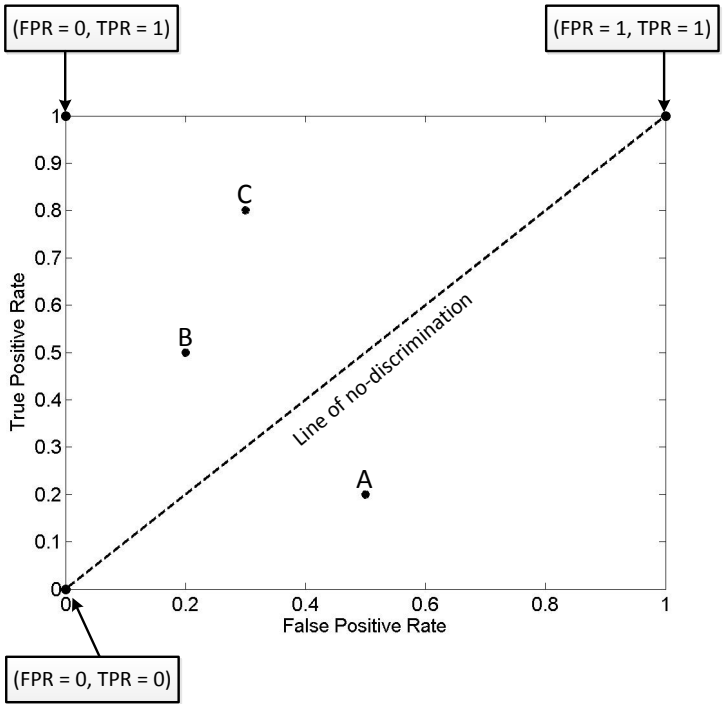

Figure 2.2: Receiver Operating Characteristic graph with examined points marked out.

There are three critical points in an ROC graph that are especially important for the understanding. In the Figure 2.2 these can be found at the corners which have a corresponding information box that contains their coordinates.

1. The first point is associated with extreme classification. This is when all windows are identified as negative regardless of their actual class. Therefore, the variables TP and FP are both zero and in the graph this corresponds to the point (FPR=0,TPR=0).

2. The second point is also an extreme classification, although this time all the windows are described as positive. In this case, the variables TN and FN are set to zero and as a result giving the point (FPR=1, TPR=1).

3. The third critical point corresponds to having an ideal classifier. That is when all the positive and negative windows are identified to their actual class. In the graph this is represented by the point (FPR=0, TPR=1), which simply mean that there are no false positive and according to equation 2.3 there is also no false negative. This corresponds to both 100% sensitivity and 100% specificity.

Considering this last critical point, if two different classifiers each produce a discrete-value point in the ROC graph, then the classifier which point is closest to the upper-left corner in the graph is the preferred one to choose. Thus, an ROC graph creates an illustrative method to analyse how the performance of a classifier may vary when the conditions changes.

Chapter 3

Background Theory

In order to follow the construction of the system that is made in Chapter 4, it is important to completely understand the background theory that is used. Thus, this chapter will describe those definitions that are used later on. It will mainly concern the theory behind the classification stage, however some other theory will also be mentioned.

3.1 Feature extraction

A commonly used method for processing images before they are sent to the classifier is to make a *feature extraction*. There are a lot of different features that can be used depending on what type of attribute that is important for the classifier to notice. For instance, the features can be used to describe different shapes of the image, such as edge detection [9] or corner detection [10].

In this master thesis, there are two different features that the classifiers use. The first feature uses more or less the raw pixel intensities of the images in order to see if the images itself could be enough information for the classifier. Though, the images are first normalized over their intensities in order to remove the lighting variations, mentioned in Section 2.2.1. Afterwards, the result for each image is collected in a vector known as a *feature vector*.

The second feature uses a bit more knowledge of how the pixel domain is structured by studying the neighbourhood relationship between the pixels. This is known as *Local Binary Pattern* (LBP) and is presented in the next Section 3.1.1. This result is added to the previously constructed feature vector for each image.

3.1.1 Local Binary Pattern

The Local Binary Pattern method is a pattern recognition that uses the relative grey levels from the examined pixel to its neighbouring pixels. Because it uses relative measurements, this feature is not affected by variation in brightness between images. Thus, the image does not have to be normalized before proceeding. LBP has been proven to be useful for face recognition [4] and background subtraction [18] which is why it is used in this master thesis.

According to [3], at each pixel position the LBP method starts by extracting the neighbouring pixels. These are found by positioning the examined pixel in the centre of a circle, which is created by a predefined radius R, and then selecting N equally spaced pixels on the edge of the circle. However, since the correlation is greater between pixels that are close to each other than far away, the radius should not be chosen too large. The neighbouring pixels are later labelled, in a clockwise manner, starting from zero.

Three different examples of how the pixels are situated depending on the circle radius and the amount of equally spaced pixels are illustrated in Figure 3.1. Also, the label of each pixel are visualized.

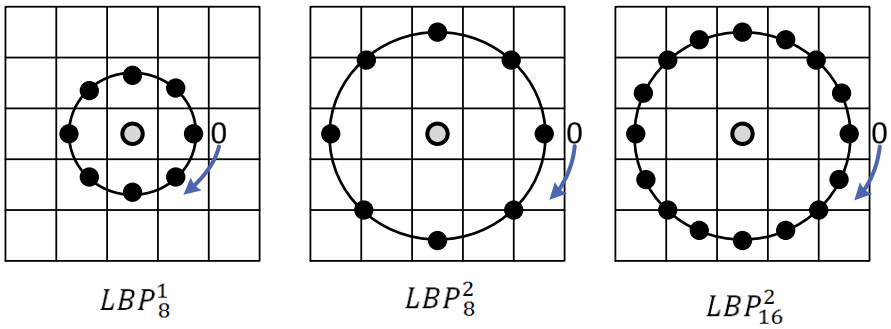

Figure 3.1: An illustration of how the LBP feature can be constructed in different ways by changing the radius and the number of equally spaced pixels on the edge of the circle.

The centre pixel is used as a threshold for the surrounding pixels intensities. If neighbouring pixels intensities are higher or equal than the centre pixels intensity, then their value is fixed to 0. Otherwise, if the intensity is lower, the value is set to 1. Combining these relative values from the neighbouring pixels produces a binary number that can be converted into a decimal number.

Depending on how many pixels there are in the neighbourhood, the number of labels that can be obtained varies. If the radius is one, then the neighbourhood consist of 8 pixels and thereby producing an 8-digit binary number. This

corresponds to a total of 256 different labels that can be obtained depending on the relative grey levels between the centre pixel and the surrounding pixels. These results are summarizes in the following general definition based from [19].

Definition 3.1. *For a pixel (x,y) the* ***Local Binary Pattern (LBP)*** *can be derived by:*

$$LBP_N^R(x,y) = \sum_{i=0}^{N-1} s(n_i - n_c)2^i. \tag{3.1}$$

$$s(t) = \begin{cases} 0, & t \geq 0 \\ 1, & otherwise \end{cases}$$

Where n_c correspond to the grey level of the centre pixel and n_i to the grey levels of N equally spaced pixels on a circular radius R.

In Figure 3.2 the procedure of collecting a LBP value for a circle of radius one is visualized. Here the number of equally spaced pixels on the edge of the circle is eight.

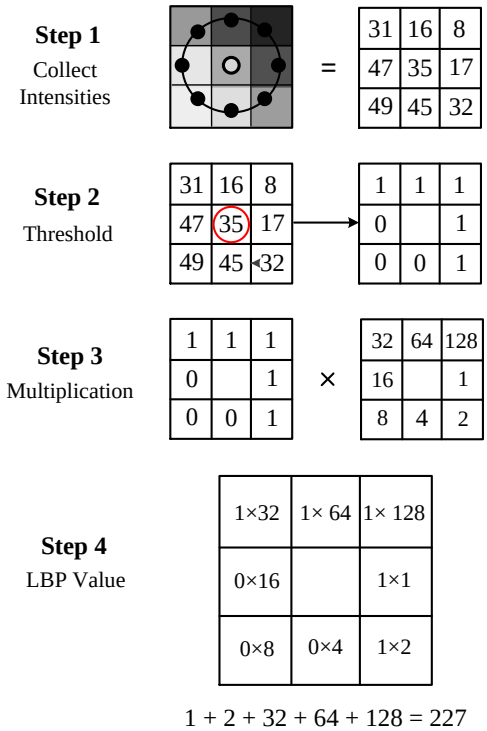

Figure 3.2: A visualization of the different procedure step of the LBP method.

3.2 Decision trees

A commonly used building block for prediction in machine learning algorithms is *decision trees*. There are two main types of decision trees, which is *classification tree* and *regression tree*. Both of them take observed data and draws conclusions based on different conditions until a decision is reached. The difference lies in how the predicted outcome is represented. For the classification tree, the outcome is mapped into predefined classes. As an example, given a person's age a classification tree can be used to predict what car type a driver usually has. The regression tree, on the other hand, produces an outcome that can be considered as a real number. For instance, a regression tree can predict the price for a certain product given its characteristics. Both types are combined under the term *Classification And Regression Tree* (CART).

According to [12], decision trees are particularly appealing because of their intuitive and simple representation. Provided that the examined data is not too large, they are also quite easy to visualize. Furthermore, they can be constructed relatively fast and thus acquiring a high execution speed, compared to other models. The disadvantage lies in the performance that might not always be the best compared to other methods, even though it is more compact.

If the name "decision tree" is studied, each term by its own, then this provides an indication of how the method works. The term "tree" originates from the methods similar construction to a natural tree. However, unlike a natural tree, the structure of a decision tree is turned upside down. This leads to, the root is found at the top of a decision tree instead of at the bottom. The technical term for this starting point of the tree is *root node*.

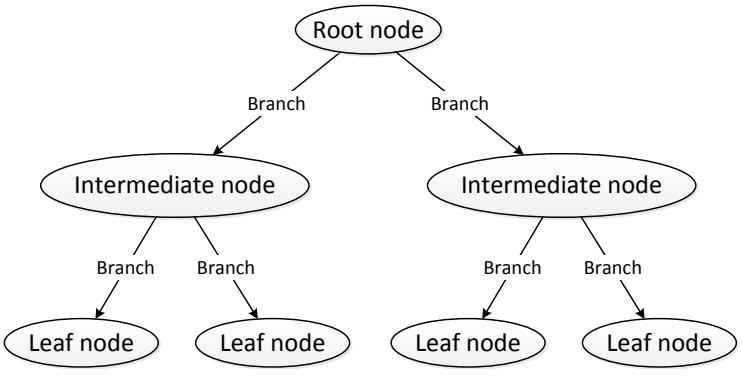

Figure 3.3: Traditional tree structure.

From the root node, the tree splits into two or more *branches* which create the next level of the tree. The end of each branch is referred to as an

intermediate node. Each intermediate node is also able to split into multiple branches, creating a new level of the tree with corresponding intermediate nodes. If an intermediate node does not split any further, then the branch terminates in a *leaf node* instead. A visualization of a traditional tree structure with its defined variables can be seen in Figure 3.3.

The term "decision" is associated with the result that should be chosen given a particular input that has been sent through the decision tree. This means that, starting at the root node, the evaluated input is confronted with either a question or some sort of a test called a *split attribute*. The outgoing branches from the root node correspond to different conditions that are associated with that specific question or test.

There are a numerous different decision trees that can be created depending on how many splits that each node makes. One decision tree that is of particular interest is the one that at each node limits the number of splits to two. This is referred to as a *binary decision tree*. The previous Figure 3.3 is also an example of a binary decision tree.

3.2.1 Deterministic split predicates

It is most common to use *deterministic split predicates*, that means given the current test and the knowledge about the input data, the different conditions attached to the branches can be determined to be true or false. For each split attribute there is exactly one condition that is true, the rest is false. By selecting the condition that is true, the input navigates to the next intermediate node, where another split attribute is considered. This procedure is repeated in a recursive manner until a leaf node is reached which corresponds to the decision or action that should be made considered that specific input.

3.2.2 Greedy induction method

The aim of decision tree classification is to have as few as possible split attributes. This would correspond to a faster method because each split separates the different class labels as much as possible from each other. This is why the choosing of the split attributes is so important for the efficiency of the method.

Decision trees use mainly *the greedy induction method*, which can be described as a recursive partitioning. By obtaining a partitioning of the data known as the training set, it forms a model that generalizes the relationship between the input data and the prediction. At each stage in the tree, it decides upon a split attribute and by means of this found classification rule, separates the data into

smaller parts. Thereafter, it repeats in a recursively manner the same process for all the newly established sets.

The more split attributes that needs to be considered, the less reliable the prediction becomes due to the tree will eventually learn noise in a phenomena known as *overfitting*. The process ends when a termination condition is fulfilled, making it a leaf node. This leaf receives its predicted label by the majority class label of the training set. However, detecting the stage where the process should stop before it starts overfitting is challenging [12].

A simple example of the greedy induction method is made in Figure 3.4. The training data is marked out in the graph as squares and circles. These correspond to a decision that has been made whether to buy an apartment or not, based on the price and the distance to work. A circle corresponds to buying and a square corresponds to not buying the apartment, given the circumstances. These set of data can at this moment be used to generate the first split attribute. This should separate the data into subsets that contains as similar data as possible. Therefore, the first split attribute correspond to a price higher or lower than $1,000,000$ sek, which is the dotted line in the graph.

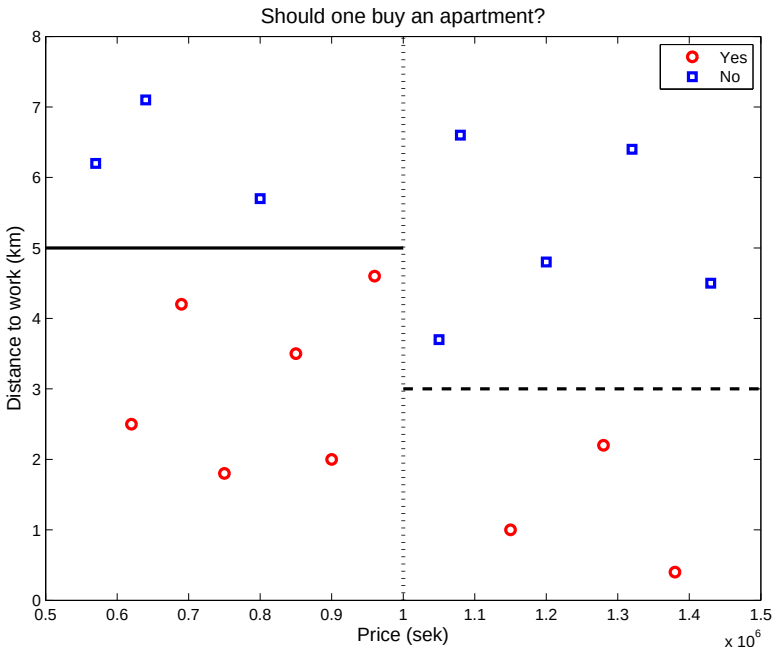

Figure 3.4: Training set used by the greedy induction method.

For the second split attribute, the data corresponding to a price lower than $1,000,000$ sek are evaluated. By introducing a split attribute at the distance of 5 km from work, the solid line in the graph, the circles and squares are completely

separated from each other. Thereafter, it is only the data corresponding to a price higher than $1,000,000$ sek that needs to be divided further. On account of this, a dashed line at the distance of 3 km from work will separate these data completely.

At this moment, four homogenous areas have been constructed using three split attributes. The two lowest areas correspond to the case of buying the apartment conditional upon the distance and price. In a similar manner, the two upper areas correspond to not buying the apartment. Thus, a decision tree can be constructed, see Figure 3.5. This can be used for future predictions of when to buy an apartment based on these conditions.

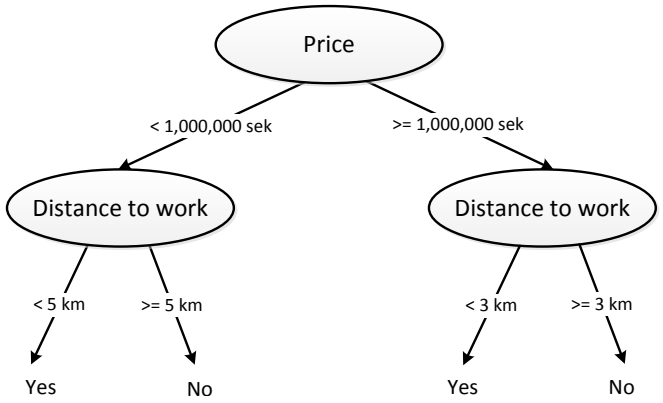

Figure 3.5: A simple illustration of a decision tree that predicts when one should buy an apartment given the price and the distance to work.

As illustrated, the training set is the foundation to how the resulting decision tree will operate. This implies that decision trees are extremely sensitive to variations of this set. A mere change by including or excluding some data might lead to a completely different decision tree and thus a different model. This is due to the changing of the split attributes and therefore the final class label is affected. As a consequence, decision trees are considered to be unstable if this is not taken into consideration when generating the decision tree.

3.3 Bootstrap Aggregating

Leo Breiman proposes in [7] a solution to unstable decision trees that was of a concern in Section 3.2.2. He claims that the variations caused by single training set can be overcome using a procedure that is referred to as *bootstrap aggregating* or simply *bagging*. The concept combines two techniques, *bootstrap sample* and *majority voting*.

3.3.1 Bootstrap sample

The procedure starts by replacing the single training set by an equally large randomly collected set using the statistical method known as bootstrap sample. Here, the multiple tuples generated by bootstrap sample are selected independently based upon the method of *uniformly sampling with replacement*. This means that after a tuple has been selected from the entire trainings set, it is added once again to the training set. Thus, every tuple is given the opportunity to be selected into the same set of bootstrap sample multiple times with equal probability.

On account of the bootstrap sample having the same size as the training set, there is one particular interesting property that occurs. Assume that the training set is relatively large N and the sampling process to create the bootstrap sample is repeated equally many times, thus making it the same large size. Each tuple from the training set has a probability of $^1/_N$ to be selected according to the uniform probability distribution of the set. The absolute complement for this occurs when a tuple is not selected, which corresponds to the probability $(1 - 1/N)$. Because the bootstrap sample should be of size N, the probability that a tuple will not be chosen during the entire process is equal to

$$(1 - 1/N)^N. \tag{3.2}$$

When N is large, this probability approaches

$$e^{-1} = 0.368, \tag{3.3}$$

according to [34]. This means that, on an average, 63.2% of the tuples from the original training set will end up in the bootstrap sample. However, in order to reach the predefined size N, the bootstrap sample will consist of a combination of 63.2% unique tuple and 36.8% duplicates. Together they represent the new training set used for building the model. Consequently, there are 36.8% of the original tuples that are left out of the newly created training set. These are thus

able to work as independent sample for performance evaluation on the acquired model and are referred to as *out-of-bag* data.

3.3.2 Majority voting

By the use of only one bootstrap sample to create a model, this could either be too simple or too specific in order to work well enough for different observations. One solution to this problem is to create multiple bootstrap sample and multiple models in a combined ensemble.

Given the knowledge of how to create a model by means of bootstrap sample, several different models can be generated from the same training set. This is where the second technique in the bagging method takes place, the majority voting. Usually, at least 50 different bootstrap samples are used to create equally many models [13]. All of these will produce its own output prediction given a certain input data. The final prediction from the bagging method is evaluated by considering which prediction receives the most votes from the models, thus the name majority voting.

This resembles the procedure when a jury combine their expertise to make a final judgement in a specific case. Depending on how large of majority, the certainty of the decision varies. For example, if 90 out of 100 models predict that the sun will shine tomorrow, then the algorithm will follow that decision and present a quite high probability for that result. Though, if only 51 out of 100 models predict sunshine, then the bootstrap aggregation method will still present the result as sunshine however, with a smaller certainty.

Because bagging uses multiple models that all are based on a randomly collected set, due to bootstrap sample, from the same training data, the original properties of the main concept to be studied are preserved. Furthermore, the statistical variance is also decreased due to the fact that each bootstrap sample is different and they in a combination compensates for each model's distinctiveness.

As previously mentioned, any bootstrap sample has on average 63.2% unique examples and the remaining 36.8% examples are duplicates. This might lead to that a tuple from the training set is overrepresented in a specific bootstrap sample or that it does not belong in some of the bootstrap sample at all. As a result of this, a model generated by a bootstrap sample could, if evaluated on its own, lead to a reduced learning accuracy. However, bagging uses majority voting where the predicted result always is constructed by evaluating all the votes from every model. Therefore, any effect of overfitting or reduced learning accuracy caused by a single model is resolved. Given these points, bagging is usually a more accurate method to use then only to consider a single model.

3.4 Random Forest

The *Random Forest* (RF) algorithm is an extension of bootstrap aggregating from Section 3.3 in combination with *random variable selection*, which will be described in Section 3.4.1. In a similar manner to bootstrap aggregating, the aim is to generate an ensemble method for classification. That means to build a classification method by combining multiple models in order to obtain better predictive performance compared to a single model. However, unlike bagging, the Random Forest algorithm introduces even more randomization principles in the creation of the resulting model.

The algorithm was developed by Leo Breiman, the founder of bootstrap aggregating, and Adele Cutler in 2001 [8]. According to Breiman, there are two reasons why Random Forest is expanded on the previous method of bootstrap aggregating instead of other methods. Firstly, used in a combination with random variable selection, the accuracy of bootstrap aggregating is enhanced. Secondly, the error rate and other estimation of the prediction performance can be valuated as the algorithm proceeds.

The algorithm uses CART:s (Classification And Regression Trees) as a key building block for its partitioning of input data. The amount of decision trees that it uses differs, however it is most common to build from 100 to 500 trees according to [33]. Because of this, the algorithm has received the symbolic term "forest" in its name.

3.4.1 One classification tree

In order to understand how the construction of each tree is made (see [6]), assume that the number of objects in the original training set is N. Furthermore, assume that the number of features in the feature vector is M. Thereafter, specify a number m that should be much smaller than the actual number of features M, that is $m \ll M$. This number should be held constant during the entire forest development.

The first step that the algorithm takes is to create a bootstrap sample of equal size N as the original training set. As described in Section 3.3.1, these selections are done with replacement. This bootstrap sample is then utilized as the foundation for growing the classification tree.

The second step introduces the method of random variable selection. As the name indicates, it randomly selects a subset of features from the feature vector. This method is often used when the length of the feature vector exceeds the number of objects in the investigated dataset, that otherwise might be difficult

to handle. For this tree construction, there is a total amount of M features in the feature vector. From those, the number of randomly chosen features is according to the predefined size m, which is kept constant during the entire forest construction. In a contrast to bootstrap sample, the collection of features is made without replacement.

This randomization technique is useful because the combination that produces the best result is not obvious to find, especially not when the number of ways to precede is overwhelming. Nevertheless, only considering a subset should not be looked upon as a guessing technique to find the best result. Actually, more information is able to impact the result of the constructed model in a contrast to other applications that uses the entire feature vector. This is due to features that normally should have been excluded by dominant features are now able to contribute. On the contrary, if there are only a few useful features among many non-informative features, this technique might lower the accuracy of the final prediction. Furthermore, by only considering a subset instead of the entire feature vector, the faster the training will be.

The third step involves the construction of the node for the first CART partition of the data. The aim is to use the m previously collected features from the feature vector in order to determine the best possible split for that node. Only these features are considered when choosing the split, not the entire feature vector. This significantly decreases the computational requirement. The algorithm then returns to step two for each subsequent split until the decision tree reaches the largest possible extent and is fully grown. The algorithm does not prune the tree, because all the trees combined in the final result will limit the risk for overfitting.

Since the algorithm uses bootstrap sample, not all of the objects from the original training set are included in the newly created training set. There are approximately 36.8% of the objects that are left out of the sample and consequently not used in constructing that classification tree. In Section 3.3.1 these are referred to as out-of-bag data. This means that the out-of-bag data are independent of the bootstrap sample and can thus be used as a test set to examine the result of the classification tree. On account of this, the corresponding out-of-bag data is sent through its decision tree when the construction of the tree is complete. The class assigned to each out-of-bag sample along with the m corresponding feature values are stored for later analyses.

In Figure 3.6 the partitioning into a bootstrap sample and the out-of-bag sample is illustrated. After the bootstrap sample has generated a decision tree based on its collected sample set, the out-of-bag sample can be sent down the tree.

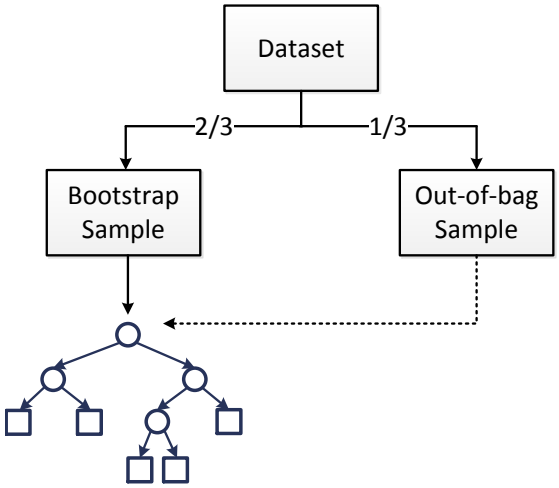

Figure 3.6: When the bootstrap sample has constructed a decision tree, the samples that are left out of the training of the tree are sent down the decision tree in order to create an independent estimate of the predictions.

Moreover, after each tree is fully constructed, the entire input data is sent in to the classification tree. The reason is to calculate the *proximity* between every pair of input objects for this specific decision tree. This can be explained as to which degree different observations tend to be classified alike. If two objects are in the same leaf node when the classification stage is over, then their proximity is set to one. Otherwise, it is assigned with a zero.

3.4.2 Constructing the Forest and the representing model

The stages described in the previous Section 3.4.1 is repeated a large number of times in order to create multiple trees for the final Random Forest model. At this point, the total proximity for the model can be derived. It starts by summarizing the proximity value across all the trees for each pair of input objects. Then the answers are normalized by dividing them with the total number of trees in the Random Forest model. This leads to a value for each pair between 0 and 1. A zero represents that they are never in the same leaf node in any of the decision trees. In a similar manner, a one corresponds that the two objects are always in the same leaf node in every decision tree. Hence, the higher the value of proximity, the more alike those observations are in how the decision trees places them. It is difficult to draw any conclusions only by studying the proximities alone. However, other information can be extracted by using the proximity in its calculations, for instance clustering and missing value imputation, see [6] for further information.

The classification made of the out-of-bag samples by each tree can now be evaluated, as the entire forest is complete. These samples will prove to be of use for evaluating an *estimate of error* to give an indication of the performance of the resulting model when it is applied to new observations. For each tree, approximately $^1/_3$ of the samples from the original training set are left out and placed in the out-of-bag set, see Section 3.3.1. This corresponds to that each sample in the original training set has, in average, been classified by one-third of the trees. For each sample, the number of times it becomes assigned to each class is counted. Then by considering the class that received the most number of votes, if this is not the true label for that sample, then this vote, averaged over all cases, is the out-of-bag error estimate.

RANDOM FOREST AS AN ALGORITHM

Assume there are N observations in the original training set. Furthermore, assume that there are M features in the feature vector and that the number $m << M$, is held constant during the entire forest growing.

1. Take a bootstrap sample of size N from the original training set to obtain a new dataset.

2. Take a random sample of size $m << M$, without replacement, from the feature vector.

3. Use the randomly sampled features m to construct the best partition of the dataset.

4. Return to step 2 for each subsequent partition until the decision tree is grown to the largest extent possible. Do not prune.

5. Drop the out-of-bag data down the decision tree. For each observation, store their assigned class label along with their m feature values.

6. Drop the original training set down the decision tree and calculate the proximity between each pair of objects. Store the results.

7. Repeat the steps $1 - 6$ a large number of times until the Random Forest model is completely assembled.

8. Calculate the total proximity for the Random Forest model.

9. For each sample from the original training set, go through all the decision trees in the model. If the sample belongs to the out-of-bag data of the tree, store the class assignment of the sample for that tree. When every tree has been considered, count the number of times that observation is classified in one category and the number of times classified in the other category.

10. Assign each sample from the original training set to a category by a majority vote over the set of trees.

3.5 Improving the classification

As mentioned in Section 2.1, when searching through a complete image for squares that corresponds to positive windows, these are highly outnumbered by the amount of negative windows found in the same complete image. Assume that many of these negative windows are significantly different from the positives. As a result, when a complete image is processed by means of a classifier, only a fraction of the negative windows would be confused as positive and labelled thereafter. However, because the proportion of negative windows is so vastly in numbers compared to the positives, even a false positive rate of 10% would still correspond to an enormously amount of windows being misclassified.

In an attempt to solve this issue, two different concepts are introduced. The first is referred to as a *bootstrap configuration* and the second is *cascading classifiers*. The main procedures for these two concepts are quite similar. They both operate with improving the classification by trimming the training set before a new classifier use it for learning.

3.5.1 Bootstrap configuration

This method is introduced in the article [29] by Sung and Poggio. It shares a similar technique as bootstrap aggregating (bagging) from Section 3.3. However some modifications have been made compared to that algorithm. Unlike bagging, the bootstrap configuration iteratively improves the same classifier instead of generating several classifiers to function as a combination.

As previously stated, the aim is to reduce the number of misclassifications of the negative predictions. Due to computational limitations and restrictions in the memory of the computer, the training set cannot contain all possible cases of the negative examples. However, the training set can be modified to contain the cases that are of most use for the learning process when obtaining a decision function. This knowledge is the foundation of bootstrap configuration.

At the beginning, the bootstrap configuration builds a classifier using the commonly known methods. That is, to retrieve a training set and then use this for its learning process. However, in order to improve this classifier to be able to classify more difficult negative examples, a more specified training set is thereafter assembled. By means of this collected training set, the learning process starts over again. As a result, a more improved classifier is generated, which the first classifier becomes exchanged by. Iterating this steps several times will lead to a more advanced classifier in the end, which is specialised in decreasing the misclassification of the negative examples.

3.5.2 Cascading classifier

The method of cascading classifier is described in the article [20] by Viola and Jones for their object detection framework. The aim of the method is to combine several classifiers in a cascade where only the windows predicted as positive by the previous classifiers is used as input data to the next classifier. On account of this, windows found further down in the cascade are more difficult to separate from each other and therefore more advanced classifiers needs to be constructed to achieve low false positive rates. The classifiers utilized in the cascade can be generated using a variation of techniques, although for this master thesis, they are built using the previously described Random Forest algorithm, see Section 3.4.

The general layout of cascading classifier is represented in Figure 3.7. As seen, all the examined windows are sent through a first classifier. This data usually consist of a large quantity of easy negative predictions and the aim of the first classifier is to find these. Therefore, the first classifier is usually of a simpler kind that reduces the amount of data that the second classifier needs to process.

The second classifier, being a bit more advanced than the first classifier, computes its own prediction given its input and draws conclusions of which windows that are positive and should be sent further down the cascade. In a similar manner to the first classifier, it removes windows from the data which is predicted as negative. The process continues in this order until it reaches the last classifier and produces its overall result. Thus, a window is predicted as positive by the cascading classifier if and only if each classifier assigns it as positive. As a result of the cascading method, the classification performance is increased while the computational time is reduced for every subsequent classifier.

Depending on what result is desirable and what is actually manageable to make, the number of classifiers in the cascade varies. However, it is important to note that the cascade can be terminated after any classifier as long as the classifiers in front have been used. Though, an early termination may not lead to the best result of the prediction compared to what a complete cascading classifier might produce.

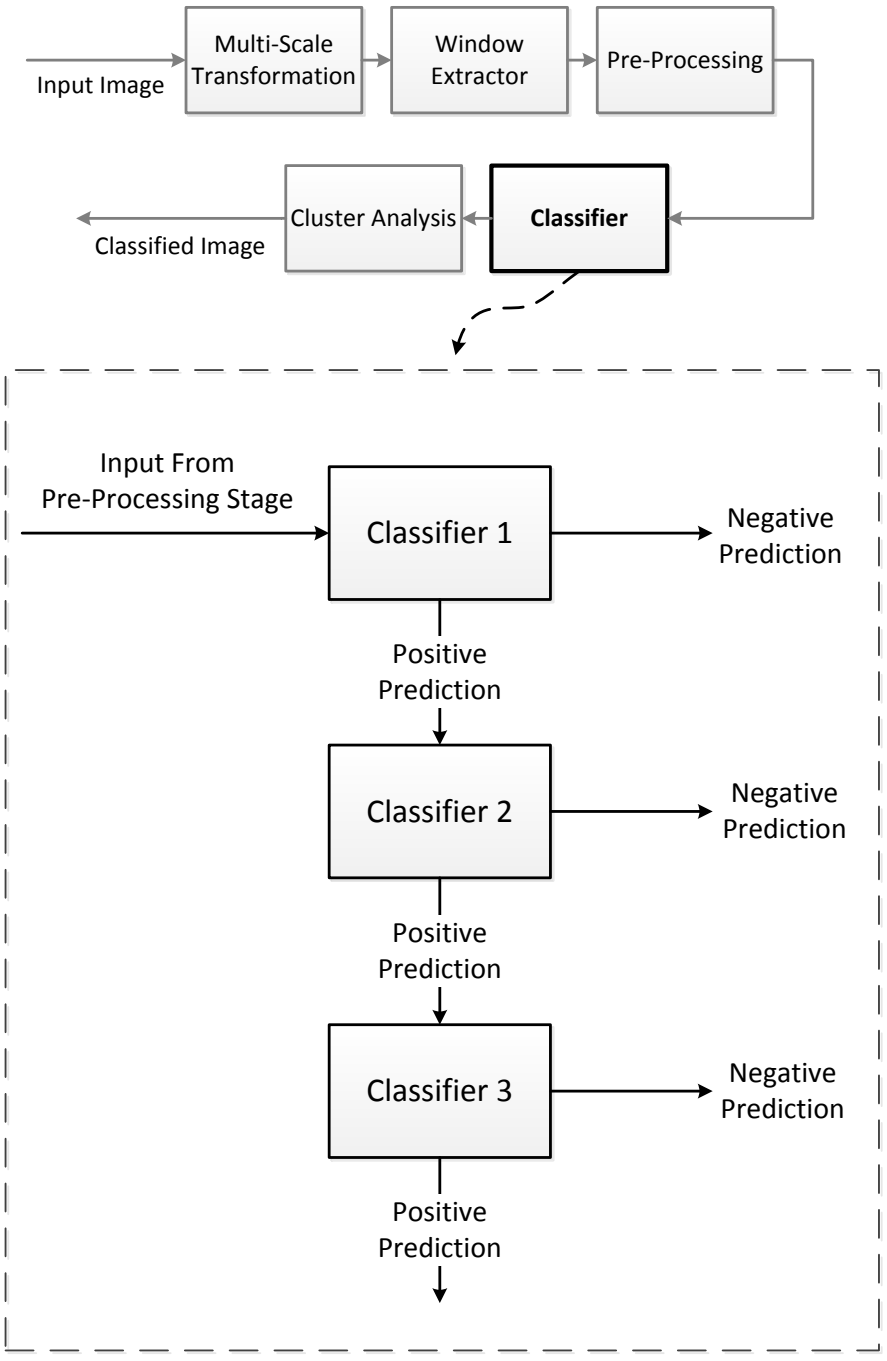

Figure 3.7: Illustration of a cascade classification layout.

Chapter 4

System Construction

The purpose of the classification system for this master thesis is to detect and recognise wheels from input images. The general layout of the classification system is illustrated once more in Figure 4.1. Here, the five most important stages are marked out: multi-scale transformation, window extractor, pre-processing, classifier and cluster analysis. In the following sections these stages are described in more detail together with some other concepts needed for the construction of the system.

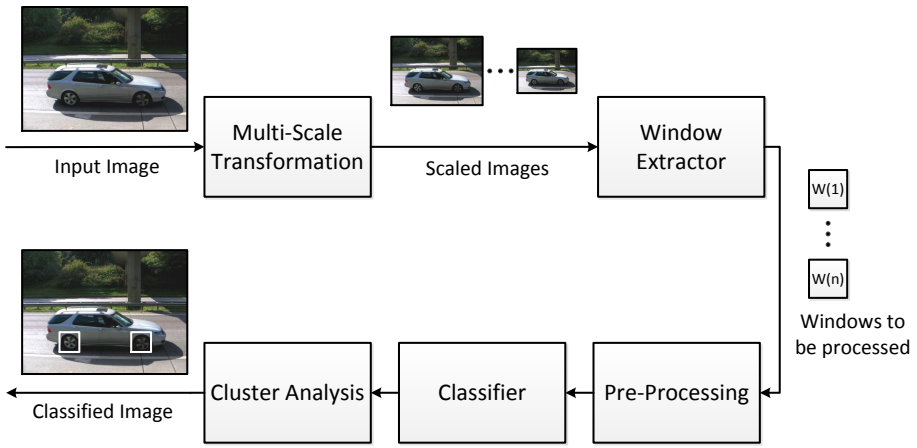

Figure 4.1: The general layout of the classification system.

4.1 Generating the input images

For this master thesis, a recording has been provided by ROADINFO in order to create the input images. They have placed a video camera at the side of a roadway which records the vehicles from the side when they move along the road, see Figure 4.2. There are four lanes in total, two in each direction, and it is only the two lanes closest to the camera that are considered in this work. From the cameras viewpoint, these two lanes move in an easterly direction. The other two lanes have too much obstacle in the way in order to work as a sufficiently good data. The road has a speed limit of 70 km/h and has no sidewalk. Therefore, there are no pedestrians, cyclists or other road users that disturbs the recording. The movie has been recorded without interruption during daylight and is almost 1 hour and 10 minutes long. There has been no rainfall or other weather disturbances; however the sunlight varies a bit during the recording.

Figure 4.2: An image from the recording.

With 30 frames per second, the entire movie consist of roughly $126,000$ images. In order to compress it to a more manageable set, only every fifth frame has been considered as illustrated in Figure 4.3. This narrows it down to $25,200$ images. Despite this, there has been no loss of information regarding a vehicle being represented successively in images, because the road being recorded is longer than it takes for a vehicle to disappear out of sight within 5 frames. As an example, if a car is driving in the lane closest to the camera with a speed of 70 km/h, then this corresponds to 19.4 m/s. Due to the cameras field of view, this lane is approximately 6.5 m long in the direction of the car. Therefore, at this speed it will be visible for around 0.33 seconds. With the camera recording 30 frames per second, it will give roughly 10 images of the car in total, and hence $2-3$ images are considered, depending on where the evaluation starts. A car that moves with a lower speed or in the lane further away from the camera will of course be visible in even more images.

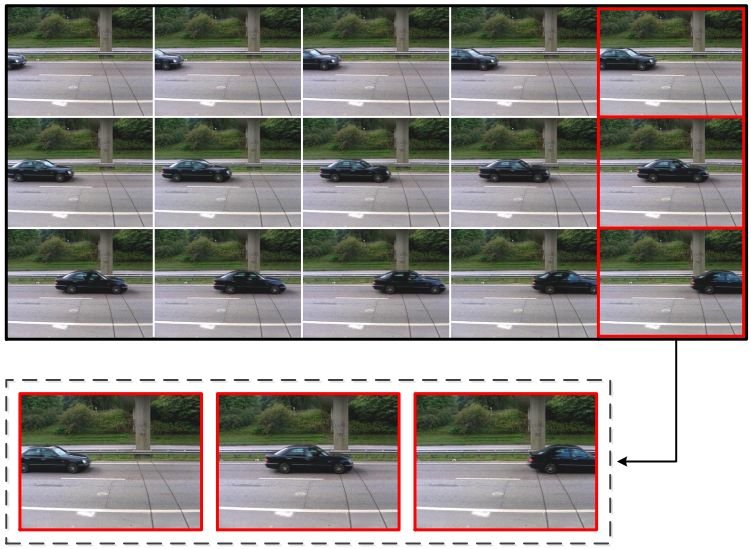

Figure 4.3: Every fifth frame from the recording is extracted.

At this moment, there is no predefined information of where the wheels are located in the image or if an image actually contains a wheel. This is crucial information that needs to be collected in order to utilize some of these images in the construction of the classifier. Also, it is important for those images used for testing the final classifier, so the classifiers prediction can be verified. Therefore, the next phase is to manually study each image in order to retrieve this information.

When a wheel is found in an image, it is marked out by a square and its location stored. Moreover, it is labelled with which vehicle type it belongs to; car, van, bus, truck, trailer or other motor vehicles. In order to attain a more uniform object to identify, only wheels from cars have been considered in this master thesis. Understandably, there is not a car in every frame recorded by the camera; therefore the number of images has been greatly reduced, resulting in a more adequate amount of images. Moreover, if a wheel is not fully visible because it is covered by another vehicle or it is located at the edge of the image, then this wheel is discarded. Consequently, the entire collection of input images consists of 828 images with a total of 1234 unique wheels. Thus, each image will contain one or several wheels from a car. Furthermore, every location of a wheel is stored for future evaluations in the system. Some of the input images will be used for the training process of the classifier and the rest will be utilized for testing the final classifier.

4.2 Multi-scale transformation

One key building block of a functional classifier is to transform the desired objects in the image so that they are as similar as possible. In these input images, the objects of interest are the wheels. However, depending on if a wheel belongs to a car in the front of the image or further back, the size of the wheel will be different. Thus, the first stage that every input image passes through in the system is the multi-scale transformation.

This generates one original and four rescaled transformations of the same image. All the input images have the original size of 480×640 pixels. In every step, this image is rescaled by a factor of 0.8 constructing a slightly more reduced image each time. As a result, the rescaled images will reach from the size 384×512 pixels down to 198×263 pixels. Consequently, the size of the wheel will be different in every image. Thus, there will be at least one scale level where the wheel corresponds sufficiently near to a predefined universal wheel size. This will prove to be of use in stages further down the classification system. Nevertheless, another effect of the rescaling is the alteration of the wheel's location. However, by utilizing the previously stored original location of the wheel, these new locations are evaluated according to which scale level it is found in and the results are stored.

The final procedure in this stage transforms the colour scheme of the image. The input image is originally in colour. However, it can be even more difficult to detect objects in those sorts of images due to larger variations. Thus, the original image and its rescaled versions are transformed into greyscale before entering the next stage.

4.3 Window extractor

At this moment, the multi-scale transformation has generated five images that should be evaluated further in order to detect and classify wheels in the images. However, these images represent a car on the road with occasionally one or several other vehicles. Thus, the wheels only correspond to a fraction of the image. Therefore, some further preparations need to be done to the images.

It is of interest for the classification part to find as good sub-image as possible that mainly consist of the wheel and not so much surroundings. Though, when an image is sent in to the system for prediction, the wheels locations are unknown. This should not be mistaken for the already stored locations that are utilized for evaluation of the classifiers prediction. As a solution to the problem, an enormous amount of windows are extracted that covers the entire

image, some of these will be a decent cut-out of a wheel. These windows need to be of the same size as the windows the classifier was trained on in order for the prediction to work. In this master thesis 32×32 pixels was chosen as the universal wheel size.

Before the stage of the window extractor begins, a property of the images is utilized in order to decrease the computational time and lowering the amount of windows extracted for prediction. As mentioned in Section 4.1, it is only the two lanes closest to the video camera that are used in this master thesis. Based on the fact that the camera is never moved during the recording, the positions of the lanes in the image are constant. Thus, it is not a requirement to search through the entire image in order to find a wheel. Actually, more than half of the image visualizes the background, the two further lanes and the barrier separating the opposite directed lanes. In Figure 4.4, the stored values of some of the wheels locations are applied to illustrate where it is most likely to find a wheel in the image. As a result, the input images are only investigated in the area corresponding to the region between the two red lines in the same figure.

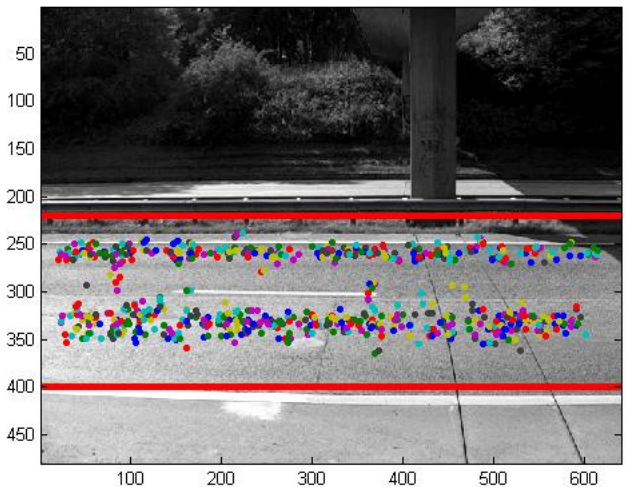

Figure 4.4: The points in the graph correspond to different wheels locations in the images.

Thereafter, the window extractor stage proceeds as follows. For each image corresponding to a certain scale level, a square of 32×32 pixels scans that image, starting at the previous designated height at the pixel furthest to the left. Then it moves from left to right, row by row until it reaches the end of the search region. The extracted window is of equal size regardless of which scale level that is considered. As a consequence of this, a window from a scaled image cover a different amount of area than a window in the original scale does.

Thus, a window extracted from the smallest scale corresponds to a square of size 78×78 pixels in the original scale image. On account of this, it is possible to find a window in the image that completely covers the searched wheel, regardless of its size.

4.3.1 Positive and negative windows

For each of these windows, a corresponding class label describing if it is a wheel or not, is stored for future verification of the classifiers prediction. This is a usual procedure in the construction and evaluation of a classifier. As the number of possible class assignment is limited to two, this method is known as a binary classification problem and this has previously been described in Section 2.2.

From Definition 2.1 it is known that an object in a binary classification problem receives a binary valued class label. That means it either becomes assigned with one or zero. On account of this, if a window is labelled with one, then it is referred to as a positive window. In a similar manner, if a window is assigned zero, then it is known as a negative window.

It is important to thoroughly define what class label a certain window should be assigned to, so there will be no misunderstandings. In order to construct these definitions, the previous registration of the location of a wheel in an image is utilized in combination with information about the window, such as its size and position. These values make it possible to calculate if there is any intersection in the image between the window and a wheel. Depending on the result, it gives rise to the definitions for positive and negative windows. In this master thesis a positive window is assigned the following definition.

Definition 4.1. *Assume there are N wheels in an input image.*
Let P_{window} be the set of pixels in a window of size 32×32 pixels.
Let P^i_{wheel} be the set of pixels covering wheel i, where $i = 1, \ldots, N$.
Thus,

$$P_{window} \bigcap P^i_{wheel}$$

is the set of overlapping pixels between P_{window} and P^i_{wheel}.

If $|P|$ is defined as the number of pixels in P and

$$\frac{|P_{window} \bigcap P^i_{wheel}|}{|P_{window}| + |P^i_{wheel}| - |P_{window} \bigcap P^i_{wheel}|} \geq 80\% \qquad \text{for any } i, \tag{4.1}$$

*then P_{window} is called a **positive window**.*

This means that if the intersection between the window and any wheel in the image is great enough, then this window should be classified as a positive window. The window is always constant in size; however the wheel changes size at each scale level. The aim of this definition is to find a window that as close as possible covers the entire wheel without too much of its surroundings. Thus, this definition limits which windows that may be assigned a positive window label.

Because the window extractor collects windows by moving one pixel at a time, each area in the image will consist of many windows with similar attributes. Consequently, one wheel in an image will be represented by several windows, some with a greater overlap than others. A negative window is defined in a similar way.

Definition 4.2. *Assume there are N wheels in an input image.*
Let P_{window} be the set of pixels in a window of size 32×32 pixels.
Let P^i_{wheel} be the set of pixels covering wheel i, where $i = 1, \ldots, N$.
Thus,

$$P_{window} \bigcap P^i_{wheel}$$

is the set of overlapping pixels between P_{window} and P^i_{wheel}.

If $|P|$ is defined as the number of pixels in P and

$$\frac{|P_{window} \bigcap P^i_{wheel}|}{|P_{window}| + |P^i_{wheel}| - |P_{window} \bigcap P^i_{wheel}|} \leq 60\% \qquad \forall i, \tag{4.2}$$

*then P_{window} is called a **negative window**.*

As a result, everything in the image that is not a wheel is defined as a negative window. This could be; the road, a part of a car or another vehicle. That is, if the intersection between the window and every wheel in the image is small enough, then it is a negative window. Consequently, the negative windows involve some difficult cases that resemble the positive windows. However, they do not describe the final object as much as desired and is because of that not assigned to the positive class.

There is an enormous amount of negative windows that can be created from each image. In the case of only one car in the image and nothing else, many of the negative windows will represent the road. These images will produce similar characteristics and does not bring any new knowledge to the classifier about the differences between a negative and positive window.

Even in the case of many vehicles on the road, many windows will still produce similar attributes, just as described for the positive windows. Thus, it can be considered unprofitable for computational time to manage this large amount of negative windows when many of them do not bring any new information.

As a method used for decreasing the amount of negative windows to a more manageable set, the window extractor moves three pixels at a time before collecting the next window. Although, when the collected window comes across an intersection, regardless of its size, the distance for its next position movement is decreased to one pixel again until there is no more overlap, and then it is increased again. This is done with the purpose of not losing the amount of positive windows while reducing the amount of similar negative windows.

The gap between the different percentages in the definitions is selected to increase the boundary between when a window should be regarded as positive or negative. Otherwise, a negative and positive window could look quite similar which makes it harder when creating the classifier.

4.4 Pre-processing

The next stage in the system concerns the pre-processing of the windows before they are sent in to the classifier. It starts by feature extraction of each window. The features used in this master thesis are those described in Section 3.1, that is normal intensity and Local Binary Pattern. For the normal intensity, this is extracted by dividing each pixel in the window by the mean value of the intensity in the same window. With the sole exception when the mean value is zero, then the division is undefined. The normalization is of interest because it removes unwanted properties created for instance by different illuminations.

At every pixel location in the window, its calculated normal intensity is considered a feature value. In other words, each feature value describes what the normalized intensity is at that location in the window. These values are then stored in a feature vector describing that particular window.

The next feature is Local Binary Pattern. Unlike the normal intensity, the Local Binary Pattern cannot directly be evaluated from the window. This is due to its requirement of collecting the neighbouring pixels grey levels. The pixels situated in the outer part of the window do not have neighbouring pixels in each direction. A solution to this is to collect a slightly larger window thus acquiring all the pixels needed for the calculations.

In a similar manner to the previous feature, at each pixel location in the window there will be a Local Binary Pattern value calculated describing the window at

that specific location. These results are also entered in the feature vector after the earlier entered feature values. Since the size of the window is 32×32 pixels, the feature vector will have a length of 2048 if both features are used.

4.5 Classification

There are two main parts required for constructing a functional classifier. The first is to create an appropriate database containing both the searched object and other non-object images. The second part is based on utilizing a learning algorithm that analyses the data and recognizes patterns. In this master thesis the Random Forest algorithm from Section 3.4 has been used as a base to generate a classifier. Thereafter, two different approaches have been examined in order to improve the predictions; bootstrap configuration and cascading classifiers.

4.5.1 Generating a database

Based on the original input images before any exclusions or pre-processing is made, some of the wheels found in these images are illustrated in Figure 4.5 and Figure 4.6. They visualizes the two main components of difficulty in object recognition, the intrinsic and external variability. These were described in Section 2.2.1 and are the reasons why object recognition is so difficult to achieve. Thus, when creating the training set which the classifier should learn from, it is vital to cover as many variations of the object and non-object as possible. For this work, half of the input images generated in Section 4.1 have been made available during the construction of the database, this correspond to 414 images. As a consequence, these input images cannot be used for testing the classifier because it would produce a deceptive result. However, there are still equally many images preserved to use in the testing stage.

Figure 4.5: Two different examples of intrinsic variability; variations of the same wheel and variations between different wheels.

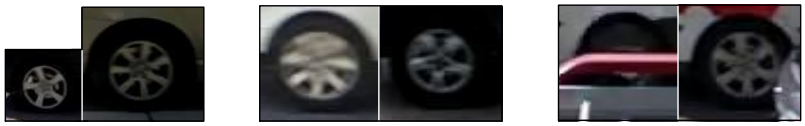

Figure 4.6: Three different examples of external variability: scale, intensity and occlusion.

Starting from the 414 collected images with a total of 620 unique wheels, a database can be assembled so it serves as a learning data for the classifier. The procedure is a combination of the previously described stages; multi-scale transformation, window extractor and pre-processing as visualized in Figure 4.7.

Each image is multiplied in the multi-scale transformation and the window extractor generates an enormous amount of windows from each of those and labels these to be positive and negative according to Definition 4.1 and Definition 4.2. On account of this, the window extractor finds approximately $30,000$ positive windows and $45,000,000$ negative windows. However, due to the computer's memory limitations, not all of these windows can be used in the database simultaneously. Hence, a predefined number is used to decide the amount of positive and negative windows that should be included in the training set. After a random sample is made of the windows to collect the desired amount, these continue to the pre-processing stage. There, the feature extraction is made on each window and the final database for training can be represented according to Definition 2.1.

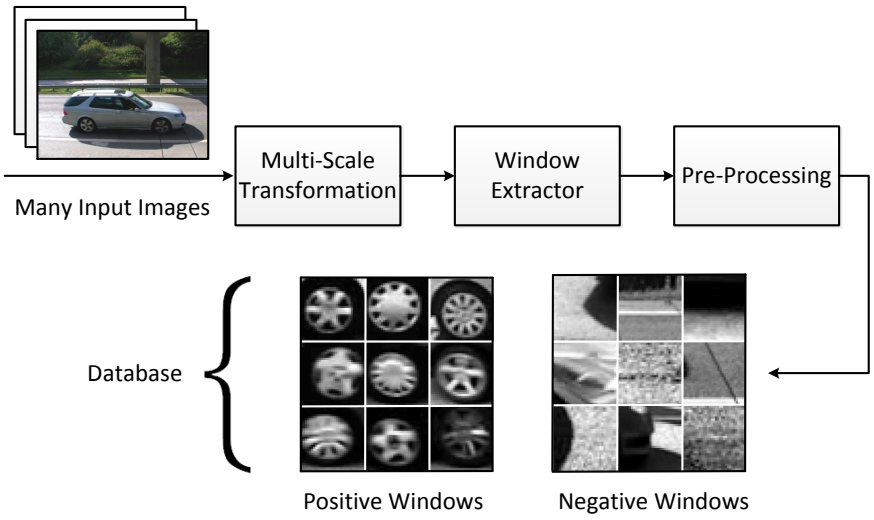

Figure 4.7: Visualization of how a database is generated.

4.5.2 Learning and improvement

A classifier can at this point be constructed by utilizing the available database which contains the training set. The learning algorithm used during this building is previously described Random Forest from Section 3.4. One of the input variables that need to be defined when using this algorithm is the number of trees used for constructing the forest. In this master thesis, this value is assigned to 100 for two reasons. Firstly, it decreases the computational time in comparison to a model constructed with more decision trees. Secondly, for those classifiers tested in this work, the model usually becomes stable after 100 trees. Thus, generating additional trees would increase the computational time more than it would give any further knowledge.

Once this first classifier has been assembled, it might not produce as low false positive rate as desired for a final classifier. Thus, this classifier operates as a foundation for further classification improvement using two different methods; bootstrap configuration and cascading classification. Both of these methods were described in Section 3.5. However, obtaining a database for each of these methods is a rather complicated task and is therefore explained in the following sections.

Generating a database to a cascading classifier

A cascading classifier consists of several classifiers in a row where each classifier should be more advanced than the previous. Therefore, they all need different databases to learn from. Moreover, the further down the cascade, the more difficult database it should learn from. The first classifier in the cascade is allowed to be of a simpler nature, provided it reduces the amount of data. On account of this, the first classifier is the already assembled classifier from the previous Section.

The input data to the second classifier is depending on what data the first classifier assigns to the positive class label. As a more advanced classifier, it should be able to draw further conclusions on this already processed data. In order to achieve this, the database for the second classifier should be collected in an expanded way in comparison to previous procedures. It is not important for this classifier to train on those examples that the first classifier already discards as negative windows.

The standard procedure is to collect a database according to previous theory in Section 4.5.1 and to let the first classifier make a prediction and present the result in a contingency table, known from Section 2.3.1. Every window predicted as negative is excluded from the second classifiers database, regardless

if the prediction was correctly made or not. As a result, some actual positive windows might be lost after the first classification round. The database consist therefore of only those windows the first classifier assigns to the positive class label. However, some of these are in fact negative windows of a more difficult classification and it is these examples the second classifier should learn to recognize.

As always, the database should consist of both positive and negative windows. By using the knowledge from the contingency table produced from the first classifier, the database of the second classifier can be correctly assembled. Those windows assigned to the true positive class (TP) are correct classifications made by the first classifier. These should still operate as examples of the searched object, which are the positive windows. The negative examples are those the first classifier misclassified as positive, corresponding to the false positive class (FP). The database is now completely obtained for the second classifier to learn from.

The procedure of constructing a database is similar for any subsequent classifier. They use those parts of the generated database that pass through all the previous classifiers. Therefore, for each classifier further down the cascade they will have more difficult data to train on than the previous. As a consequence, they can classify more difficult windows than the previous classifiers can. This gives lower false positive rate and as a result, the Receiver Operating Characteristic curve will be moved to the left.

Generating a database to a bootstrap configuration

Bootstrap configuration involves the same classifier that gradually becomes more improved in its predictions by learning from more specified databases each time. The aim is to incrementally enhance the amount of false positive (FP) in the database for each new training round and thus be able to reduce the misclassification of negative windows.

The method of collecting a database for bootstrap configuration can be divided into three main steps. Firstly, a database is constructed according to Section 4.5.1. This should consist of a small portion of negative windows in comparison to the amount of positive. Secondly, a classifier is trained of these windows using the assigned learning algorithm, Random Forest. Thirdly, a new database, only consisting of negative windows, is collected and sent through this classifier. As the purpose is to decrease the misclassification of negative windows as positive, only the cases assigned as positive after the classification is evaluated further. However, on account of the database completely consist of negative windows; the true positive class (TP) will be empty. This means, the

only class label evaluated further corresponds to the false positive (FP) that is when a negative window is assigned a positive label. A new enhanced database is obtained by combining the false positive windows from the classification, with the negative windows in the current database; though, the positive windows remain the same. Thereafter, the procedure returns to the second step where a new classifier is trained.

4.6 Cluster analysis

The final stage of the classification system corresponds to the cluster analysis. When an input image has passed through the previous stages, there will usually be many windows assigned a positive label although there is not that many wheels in the image. This is due to the window extractor, which produces many windows that overlap. As a consequence of this, when a window is classified as a wheel, there is a very high probability that the windows nearby will produce the same outcome. Thus, many windows might correspond to the same wheel.

The aim of this stage is to combine two windows to a single window if they have a sufficiently large intersection. By iterating this procedure multiple times, the amount of windows labelled as positive will decrease and give a more representative result of the final classification.

This stage also provides the system with a final opportunity to exclude misclassified predictions. The amount of windows combined together as one should give an indication of how possible it is to find a wheel there. If only a few windows have been clustered together, this might depend on misclassified windows and should for that reason not be presented in the final result. Consequently, if there has been a clustering where only three or less windows have been combined, these clustered windows are excluded. This means that windows that never became clustered at all also are discarded. Given these points, the clustering has been made on each scale level in order to exclude misclassified windows before the result is transformed into the original scale level where a final clustering is made.

The clustering procedure begins by calculating the intersection between two nearby windows. If this is greater than a predefined value, then a new representative window is created and is used as a substitute for the two discarded windows. The location of this window is calculated as the mean value of the previous windows and it receives the same size as these as well.

Depending on how large the intersection must be in order for a clustering to take place, the amount of clustered windows differs. However, for a clustering method to be useful it should decrease the amount of windows without too much

loss in the accuracy of the wheel's position. If for instance the intersection is assigned with a low value, this would correspond to windows that are located on larger distances are clustered together. As a worst case scenario, a true positive window might be combined with a false positive window on a distance far away from the actual position of the wheel. On the other hand, if the intersection is assigned a too high value, this might lead to no clustering at all. As a result, the amount of windows is not decreased. After these different aspects have been taken into consideration, an intersection of 40% has been used in this work for clustering.

Afterwards, the clustered windows from each scale are transformed into the original scale image. Here, another procedure of clustering is made and windows are discarded if previously described conditions are not fulfilled. The final predictions are presented as output from the classification system.

Chapter 5

Results

In order to achieve a greater variation of windows during the evaluation of the results, the classification systems in Section 5.1 to Section 5.4 are tested on a randomly sampled set of windows. On account of this, the cluster analysis stage is excluded during these sections and the result correspond to the output from the classification stage. However, in Section 5.5 the result of the complete classification system is presented in more detail when one input image is send in. Furthermore, in Section 5.6, several images are combined to give an indication of how the final system with all its stages performs. The results are evaluated using Receiver Operating Characteristics such as the ROC graph and the contingency table. Both of these methods are described in Section 2.3.

5.1 Asymmetric database

The performance of a classifier changes depending on how large the database is and how many variations it contains. This is illustrated in Figure 5.1. Here, two classifiers have been trained on different database sizes. The classifier to the left is trained on $1,000$ positive windows and $1,000$ negative windows. The classifier to the right is trained on ten-times as big database. The testing set evaluated contains 100 positive and 100 negative windows. As can be seen, the left classifier receives a true positive rate of 96% and a false positive rate of 6%. However, the classifier to the right has the possibility to train on more windows. Thus, it gains more knowledge on the difference between a positive and a negative window. As a result, this classifier produces a better result both for the true positive rate, with 99%, and the false positive rate, with 4%, compared to the other classifier.

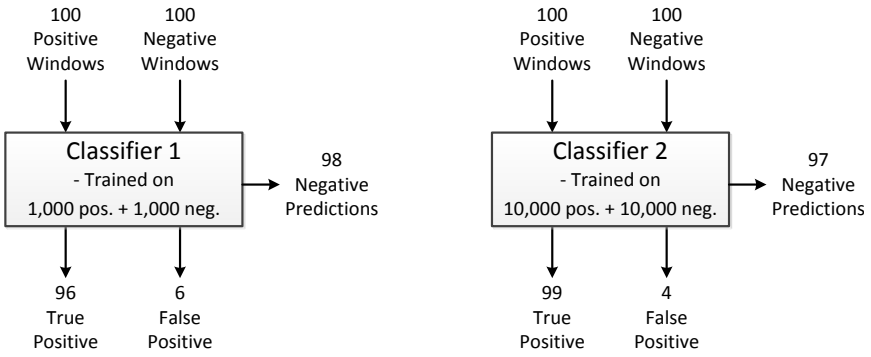

Figure 5.1: An example of classifiers trained on different database sizes.

Nevertheless, it can sometimes be a good idea to deliberately change the proportions of positive windows in relation to the negative windows to reach some desired characteristics of the classifier. For instance, if the database has an asymmetric appearance with mainly positive windows to train on, then the classifier has a higher tendency to assign a window to the positive class label. As a consequence, the true positive rate receives a high value, which is desirable, because many of the positive windows obtain a correct classification. However, more negative windows than usually becomes misclassified as a result of the asymmetric database. This leads to an increased false positive rate compared to an even training set, even though some negative windows obtains a correct classification. The purpose of utilizing an asymmetric training set is the ability to leave the amount of windows from one of the classes fairly unchanged while decreasing the amount of the other class. Two examples of asymmetric training sets is illustrated in Figure 5.2. The classifier to the left has a database consisting mainly of positive windows and the classifier to the right has a database consisting mainly of negative windows.

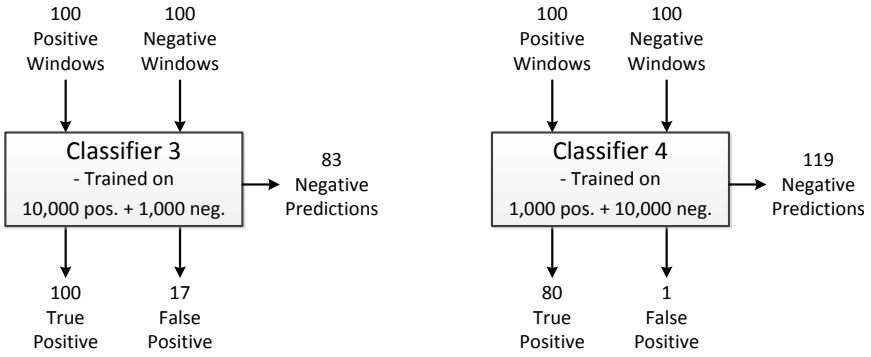

Figure 5.2: An example of classifiers trained on asymmetric databases.

5.2 Cascade classification

Five classifiers are combined in the cascade classification evaluated in this master thesis. The first classifier has an even database and is used to discard many of the easiest negative predictions. The testing set consist of 10,000 positive windows and 100,000 negative windows. These windows have been randomly sampled from all the possible windows generated by the 414 remaining input images not used in the training of the classifiers, see Section 4.5.1.

The result after the first classifier is presented in Table 5.1. It has obtained a true positive rate of 95.88%. Therefore, the amount of positive windows lost at this stage is reasonably low with 4.12% of the total positive windows. Furthermore, the false positive rate is already at 4.323%, which corresponds to the main part of the negative windows having received a correct classification. Consequently, the amount of negative windows is drastically diminished while the amount of positive windows still remains high after the first classification.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,588**	412	10,000
	NEGATIVE	**4,323**	95,677	100,000
	TOTAL	13,911	96,089	

Table 5.1: A contingency table for cascade classification stage 1.

For the second classifier, it is important to lower the misclassified negative windows, that still is quite large with 4,323 windows, while not losing too many of the positive windows. Thus, the knowledge of an asymmetric database from Section 5.1 is applied. However, in order for the second classifier to be able to distinguish the difficult negative predictions from the positives, the database needs to be further adapted.

As described in Section 4.5.2, the collection of the second database needs to take the predictions of the first classifier into consideration. Thus, the second database starts with almost thirteen-times as many negative windows as positive, before it becomes reduced by the first classifier. Thereafter, the amount of positive windows is almost double as many as the negative and an asymmetric database can be assembled.

The predictions corresponding to the true positive (TP) and false positive (FP) classes in the contingency table 5.1 of the first classifier, becomes then classified by the second classifier. The result is presented in Table 5.2. The false positive rate should now reflect the amount of false positive after the second classifier, based on the total amount of negative windows sent in to the cascade at the

beginning. A similar explanation can be provided for the true positive rate. Once more the amount of negative windows is significantly decreased, resulting in a false positive rate of 0.786%. Moreover, the amount of positive windows is kept high corresponding to a true positive rate of 93.77%.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,377**	211	9,588
	NEGATIVE	**786**	3,537	4,323
	TOTAL	10,163	3,748	

Table 5.2: A contingency table for cascade classification stage 2.

The third classifier proceeds in a similar manner as the second classifier with an asymmetric database. Though, the amount of negative windows assembled in the beginning needs to be considerably much larger due to the low false positive rate from the second classifier. As previously, the result is collected in a contingency table, see Table 5.3. At this point, the true positive rate is 90.05%, which can be considered a fairly good estimate. Also, more than half of the negative windows, that has passed through from the previous classification step, are at this stage classified correctly, making the false positive rate reach 0.351%.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,005**	372	9,377
	NEGATIVE	**351**	435	786
	TOTAL	9,356	807	

Table 5.3: A contingency table for cascade classification stage 3.

For the fourth classifier, the proportion of the asymmetric database is altered. Here, the amount of negative windows is increased from 50% of the positive examples to 70%. This will give the classifier the opportunity to train more on the difficult negative windows while still emphasizing the importance of identifying the positive windows correctly.

From Table 5.4, it starts to be more visible how difficult the classification is at this stage. The proportion of false negative (FN) is increased in comparison to previous stages. This is due to the similarities between the positive and negative windows at this stage that the classifier has been trained on. Many of the positive windows are therefore misclassified as negative and the true positive rate is therefore at 80.73%. As a result, approximately 1/5 of the original positive windows sent in to the cascade classification are at this moment absent

due to misclassifications either here or in the previous stages. Nevertheless, the false positive rate is at a remarkably low level at 0.066%. Hence, the loss in positive windows might be acceptable in order to reach this low false positive rate.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**8,073**	932	9,005
	NEGATIVE	**66**	285	351
	TOTAL	8,139	1,217	

Table 5.4: A contingency table for cascade classification stage 4.

The final stage in the cascade corresponds to the most difficult positive and negative windows to separate. Therefore, a highly advanced classifier is required. As usual, when collecting the database, the previous classifiers predictions are taken into consideration. On account of this, the database needs to contain an enormous amount of negative windows in the beginning, before downsizing. At the end, the database utilizes the same proportion of negative and positive windows as the previous classifier.

The result is presented in the contingency table 5.5. At this point, it is very obvious how much more difficult the separation has become. Almost every sixth positive window sent in for prediction from the fourth classifier is misclassified. As a result, the true positive rate ends up at 68.38%. Though, the false positive rate continues to decrease and the final value is 0.028%.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**6,838**	1,235	8,073
	NEGATIVE	**28**	38	66
	TOTAL	6,866	1,273	

Table 5.5: A contingency table for cascade classification stage 5.

When the database that the fifth classifier utilizes are visually examined, the results from Table 5.5 can be justified. There it can be seen that a majority of the negative windows does in some amount contain a car wheel. This is due to the previous description of how a negative window is defined in Section 4.3.1. Consequently, when a feature extraction is done in the pre-processing stage of the system, the negative and positive windows receives similar features. It then becomes difficult for the classifier to identify if a window actually should correspond to the negative or positive class label, because to some extent it contains a wheel.

In order to avoid future problem similar to this, the negative training windows should correspond to a much lower intersection or even not contain wheels at all. However, by training the classifier with purer negative and positive windows, it has no examples of what to classify those windows that only contain a fraction of a wheel. Therefore, it is entirely up to the classifier to assign those windows to the class label it finds suitable.

When the false positive windows from Table 5.5 are investigated, this discussed property, of the negative class containing car wheels in some amount, is striking. Actually, it is only 5 windows of the entire 28 that does not contain a car wheel in some extent. These five examples are illustrated in Figure 5.3. Two of these are pure negative windows. However, the remaining three visualizes a wheel from another vehicle group that present great similarities to car wheels and thus receives a positive prediction.

Figure 5.3: Out of 100,000 randomly sampled negative windows, only 28 false positive windows pass the cascade classification, where 23 of those are actually near hits on car wheels. The five remaining false positive windows are illustrated in the figure above. Out of these, there are three windows that belong to another vehicle type and could therefore also be considered a near hit of a wheel. As a result, there is merely two false positive windows of the total 100,000 negative windows that are truly misclassified in the cascade classification.

5.3 Bootstrap configuration

The method of bootstrap configuration has also been evaluated using the same testing windows as for cascading classification. Starting from one classifier, this has been improved four times. As the method consist of learning from the same positive windows after each improvement while updating the number of negative windows, the original database has ten-times as many positive as negative windows.

The result after the first classification stage with the original classifier is presented in the contingency table 5.6. The consequence of utilizing an asymmetric database of these proportions is clearly visible in the table. Almost every positive window obtains a correct classification while a fairly large group of the negative windows becomes misclassified. This can be compared to the previous description of Figure 5.2 from section 5.1.

The true positive rate is situated almost at the desired 100% with its 99.83%. On the other hand, the false positive rate is 17.446%. This means that almost every fifth negative window is misclassified. However, this is merely the starting classification that should discard the easiest negative predictions, which according to the results can be considered as fulfilled.

		PREDICTED CLASS		
		Positive	Negative	Total
ACTUAL CLASS	Positive	**9,983**	17	10,000
	Negative	**17,446**	82,554	100,000
	Total	27,429	82,571	

Table 5.6: A contingency table for bootstrap configuration, original classifier.

To improve the result, the database is adapted so that the negative windows are doubled before it is used for learning again. These newly added negative windows correspond to those windows that are difficult to classify. After training the classifier with this database, its result is entered in Table 5.7.

This time, the amount of lost positive windows is reasonably low in comparison with how many negative windows that are correctly classified. The true positive rate still corresponds to a high value at 98.95%. Moreover, the false positive rate is lowered to 4.533% from the previous 17.446%.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,895**	105	10,000
	NEGATIVE	**4,533**	95,467	100,000
	TOTAL	14,428	95,572	

Table 5.7: A contingency table for bootstrap configuration, original classifier improved once.

A second improvement is made by adding equally many difficult negative windows to the database as in the previous step. The amount of negative windows is now 1/3 of the positive windows. Once more, the values are presented in a contingency table, see Table 5.8. The prediction rates still correspond to satisfying results, with a true positive rate of 97.96% and a false positive rate at 1.54%. After two iterations, the classifier has gone from misclassifying almost every 5:th negative window to only misclassifying every 65:th negative window.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,796**	204	10,000
	NEGATIVE	**1,540**	98,460	100,000
	TOTAL	11,336	98,664	

Table 5.8: A contingency table for bootstrap configuration, original classifier improved twice.

For the third improvement, the database consists of 40% negative windows compared to the positive windows. This is achieved by expanding the database with even more difficult negative predictions. Here, the results are entered in Table 5.9. The true positive rate is still considerably near the desired value with its rate of 96.09%. The main part of the negative windows is also assigned to the correct class label. As a result, the false positive rate is merely 0.679%.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,609**	391	10,000
	NEGATIVE	**679**	99,321	100,000
	TOTAL	10,288	99,712	

Table 5.9: A contingency table for bootstrap configuration, original classifier improved three times.

The classifier is improved one last time before the final result is evaluated. At this moment, there are twice as many positive windows as negative in the database. Thus, the negative windows mainly correspond to windows that each improved classifier has had a problem of predicting correct. The predictions are entered in the contingency table 5.10. From there, the true positive rate can be calculated and it corresponds to 94.14%. The false positive rate ends up at 0.309%. To conclude, the false positive rate of the classifier is kept quite high while the false positive rate is reasonably low.

		PREDICTED CLASS		
		POSITIVE	NEGATIVE	TOTAL
ACTUAL CLASS	POSITIVE	**9,414**	586	10,000
	NEGATIVE	**309**	99,691	100,000
	TOTAL	9,723	100,277	

Table 5.10: A contingency table for bootstrap configuration, original classifier improved four times.

This classifier could be improved furthermore. However, it would involve a great amount of negative windows that needs to be collected in the beginning in order for the database to receive a reasonably amount of new negative windows to learn from. Due to shortage of free disk space and computation resource constraints, this size of a database has not been assembled for this work. Though, if the size of the original database is lowered, then the number of improvements might be increased. This is because every improvement would then not require an equally large contribution of negative examples as studied in the steps above. However, by starting from a larger database of positive examples, more variations are learned thus, the higher the true positive rate will be. Nevertheless, the original classifier and its four improvements can give a tendency of how the bootstrap configuration method works.

5.4 Comparison of improvement methods

One way of visualizing the true positive rate in comparison to the false positive rate is done by a Receiver Operating Characteristic graph. In Figure 5.4, the values from the two previous Sections 5.2 and 5.3, describing the different improvement methods are illustrated. In order to achieve an easier examination of the result, the false positive rate is in a logarithmic scale while the true positive rate remains the same.

In the figure, the boundaries of the ROC graph is visualized by the green lines. The different classifiers from the cascading classification are marked out

by the blue circles with lines drawn between each subsequent classifier. The first classifier in the cascade corresponds to the blue circle furthest to the right. The improvements made by the bootstrap configuration, including the original classifier, are the red triangles. Here, each subsequent improvement is connected with a line starting from the original classifier furthest to the right.

As can be seen, the bootstrap configuration method always stays above the cascading classifiers. This is probably the result of the classifiers in the bootstrap configuration having trained on a larger database, with more asymmetry, than the classifiers in the cascade.

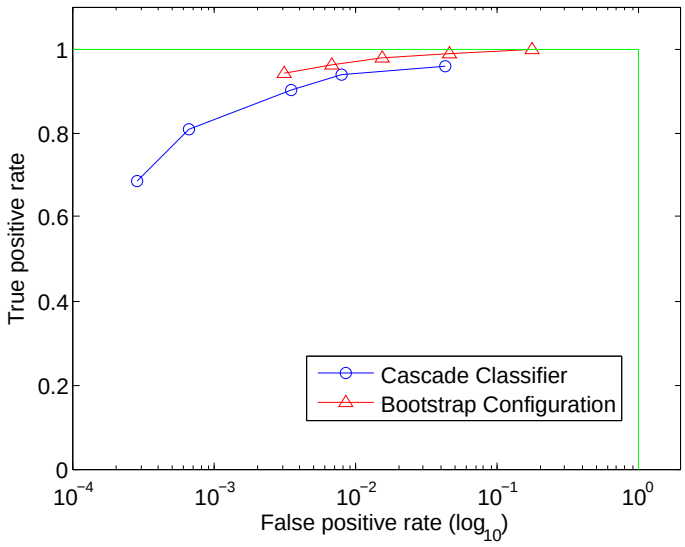

Figure 5.4: Receiver Operating Characteristic graph of the bootstrap configuration and cascading classification results.

Both methods use the same principle in generating the first classifier, the only difference lies in the structure of these first databases. The cascade classification starts its first classifier with an even database in this work. As a result, it receives a satisfying result both in the predictions of positive and negative windows for future classifiers to build on. On the other hand, the bootstrap configuration begins with a highly asymmetric database with ten-times as many positive windows as negative. Therefore, it is located closer to the horizontal green boundary line, representing the desired true positive rate. However, it also obtains more misclassifications of the negative windows and therefore it is located further to the right compared to the first classifier of the cascade

The second classifier in the cascade classification has a database with twice as many positive as negative windows. This is the same asymmetry as the database

to the final improvement stage in the bootstrap configuration, corresponding to the red triangle furthest to the left in the graph. Though, this database is ten-times as large as the database of the cascading classifier. Therefore, the bootstrap configuration produces a lower false positive rate with 0.309%, compared to the cascade classifier with its 0.786%. Furthermore, it achieves a slightly better true positive rate of 94.14% compared to the cascade, which results in 93.77%.

Given the graph and these points, the main characteristics from the two improvement methods are quite similar and can be summarized as followed.

- In the first stages, it is easier to reduce the false positive rate without too much loss in the true positive rate. This is because there are many easy negative predictions in the beginning compared to the later stages.
- The bootstrap configuration method for this work is trained on an asymmetric database that always has twice as many positive windows as negative or more. Therefore, they are located closer to the desired true positive rate than the cascading classifiers, which are created with less asymmetry in their databases.
- The original classifier from the bootstrap configuration illustrates the tendency a classifier has to assign windows according to the class most trained on. The advantages it gains for preserving the amount of one class, the more disadvantages it receives by an increase of misclassification of the other class.
- On account of a careful selection of negative windows to be added to the database, the false positive rate is decreased for each newly generated classifier.
- When more difficult negative predictions are reached, the true positive rate is more affected then earlier. Thus, in order to reduce the false positive rate further, the larger the loss is in the true positive rate.

An ideal classifier corresponds to the point in the ROC graph where the false positive rate is equal to zero and the true positive rate is one, as described in Section 2.3.1. Given the results presented in Figure 5.4, the classifier in this work that is located closest to this point is the bootstrap configuration after the fourth improvement. Then the Euclidean distance between this classifier and the ideal classification point is 0.0587. However, depending on what properties that is most important for the classifier, the choice of which classifier to use may vary. If the aim is to obtain a general classifier that produces a satisfying result both from the false positive rate as well as the true positive rate, then

the bootstrap configuration classifier mentioned should be used. Nevertheless, if it is more important to decrease the amount of false positive rate, then the cascade classification with all its classifiers should be used.

5.5 Result from one input image

As an illustration of the result when the cluster analysis is used, an input image containing two cars is sent through the classification system. The image has four unique wheels, all fully visible, as can be seen in Figure 5.5. After the window extractor stage, the amount of positive windows is 150 and the negative windows are 135,033. Thus, there are several windows corresponding to the same wheel. Therefore, there is a large margin of error which makes it possible to lose several windows in the prediction without actually losing any vital information. For this reason, the cascading classification is utilized because it performs the lowest false positive rate while still receiving an acceptable true positive rate.

Figure 5.5: The input image sent in to the classification system.

The output values of the true positive and the false positive after each classification stage is illustrated in Figure 5.6. As previously described, these output values are used as input values to the subsequent classification stage. The first classifier in the cascade reduces the amount of negative windows from 135,033 to 10,916. This massive decrease is due to the image containing many easy negative predictions. Furthermore, the loss of positive windows on account of misclassification is at this stage reasonably low. The second classifier also performs an enormous decrease in the amount of negative predictions while there is only one window loss of the positives.

The third classifier performs a smaller identification of negative predictions compared to the fourth classifier. This is a result of the asymmetric databases that these classifiers were trained on. As described in Section 5.2, the proportion of the fourth classifiers database is altered compared to the third classifier. It decreases the asymmetry by adjusting the amount of negative windows in the database from 50% of the positive windows to 70%. Therefore, the fourth classifier has more knowledge of the negative windows properties than the third classifier has.

When the final classifier is reached in the cascade, the predictions correspond to 118 true positive and 80 false positive. On account of this, the true positive rate is 78.67% and the false positive rate is 0.059%.

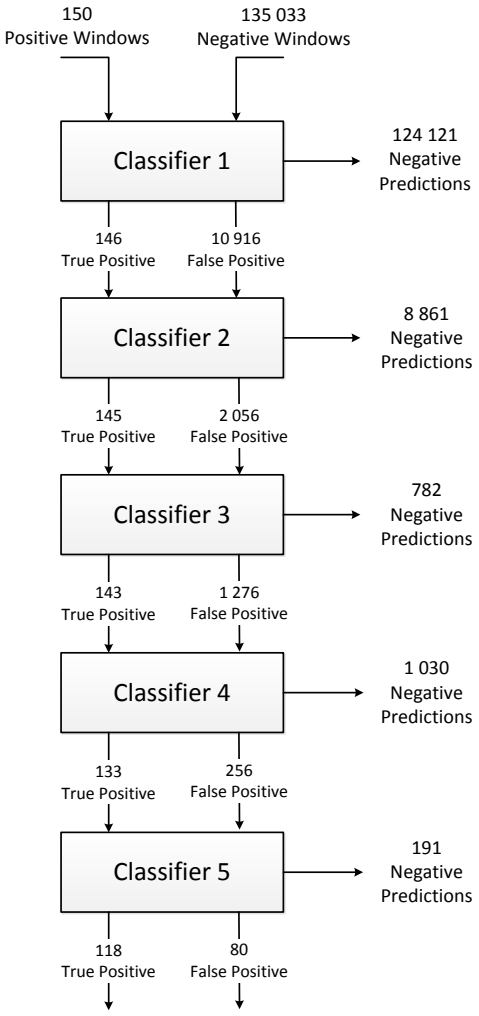

Figure 5.6: It illustrates the output classes after each classification stage in the cascade when one input image is sent through.

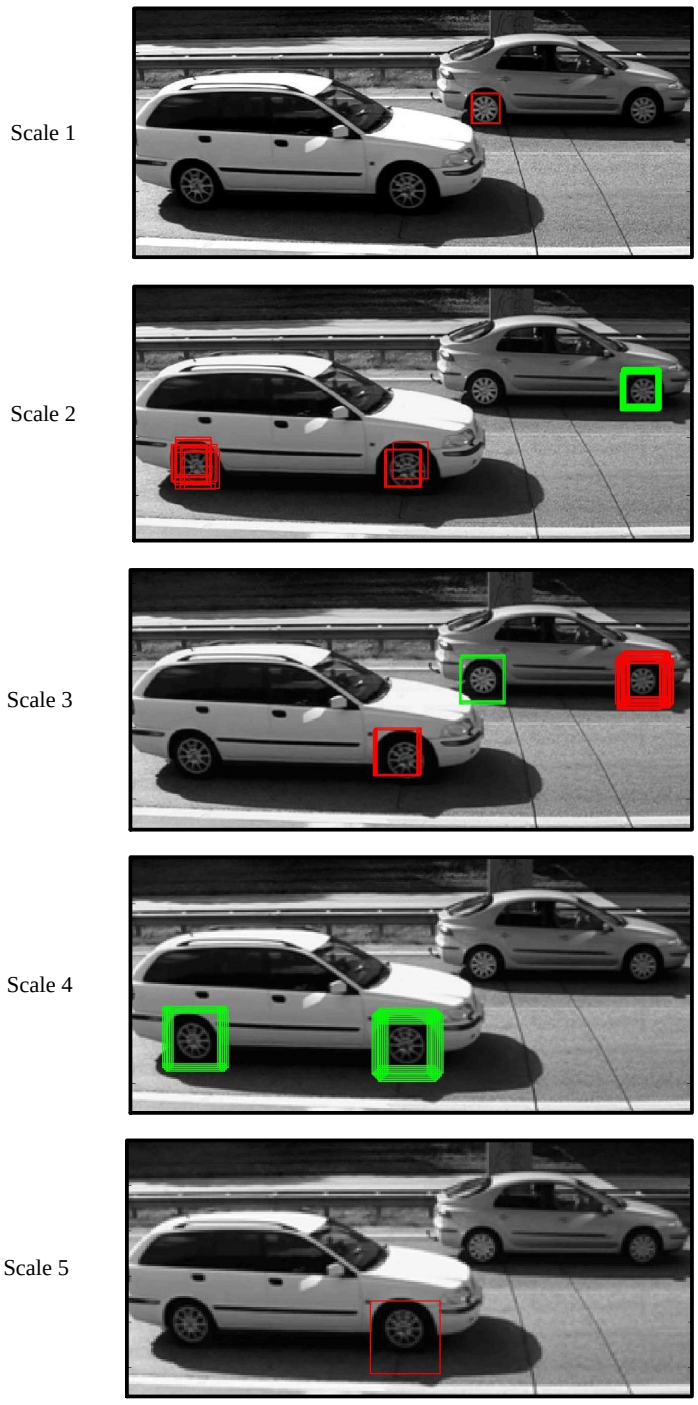

Figure 5.7: When the image has passed through the entire cascade classification, the positive predictions are marked out in their corresponding scaled image. Windows marked out with green are true positive windows and the red are false positive windows.

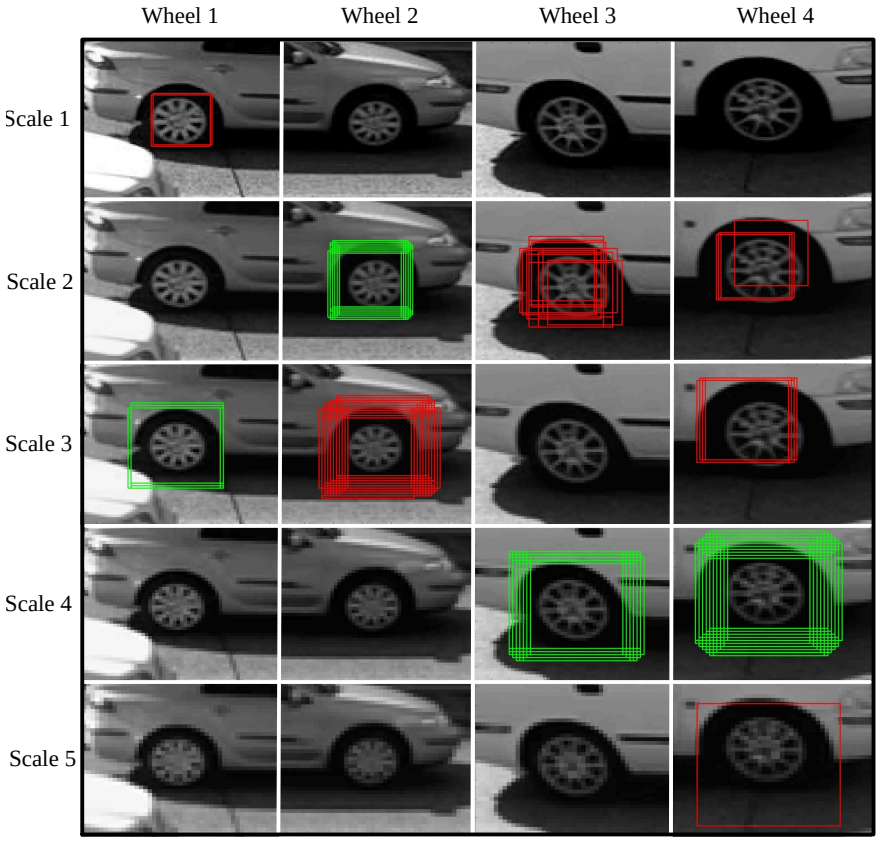

Figure 5.8: Given a certain wheel and scale image, the corresponding predictions made by the cascade classification are marked out.

Each of the true positive and false positive windows from the final classification stage are marked out in their corresponding scaled image and illustrated in Figure 5.7. Windows marked out with green are true positive windows and the red are false positive windows. As can be seen, the algorithm only finds wheels and never divert from that with other window representations. On account of this, the false positive windows do in fact surround a car wheel in some extent although not with the desired proportions.

In Figure 5.8, each wheel with its corresponding window prediction are visualized in all the scales. Here, it is easier to compare the difference between a true positive and a false positive. If for instance wheel 1 is examined, the classification system finds a wheel in the first and third scale. However, in the first scale the entire wheel is not contained within the window, therefore it should be considered a false prediction according to the definitions. In the third scale, the entire wheel is covered by the window which results in a true positive prediction.

The last stage in the classification system is to make the cluster analysis. At this moment, the windows are clustered together. Those combinations that correspond to three or fewer windows are excluded from the output, see Section 4.6. Hence, some of the false positive windows are discarded. An example of this can be found in the fifth scale image from Figure 5.7. Here, only one window has been found and thus the clustering will consist of only one window, which therefore will be excluded. The final result is presented in Figure 5.9. Here, only four windows remains, one over each wheel. In conclusion, a satisfying detection and recognition has been made on this image be means of the cascading classification.

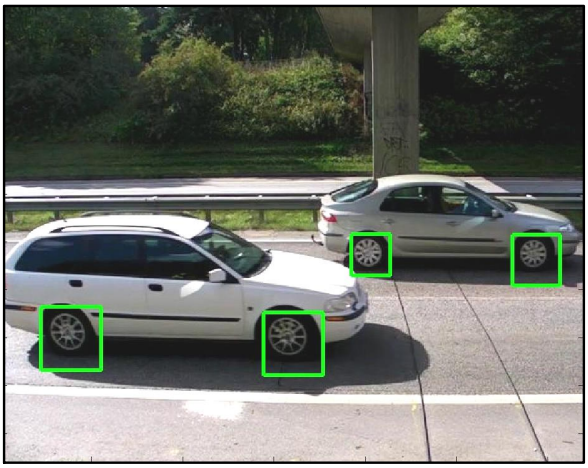

Figure 5.9: The final classification result from one image.

5.6 Result from the cluster analysis stage

As illustrated in the previous Section 5.5, after the examined image has passed through the complete classification system, all of the four wheels were located. Thus, four wheels were classified out of four possible, which correspond to a true positive rate of 100%. This could be compared to the classification stage in the system, which produced a true positive rate of 78.67%. The difference lies in how these values are derived. The classification stage calculates its true positive rate based on all extracted positive windows corresponding to a car wheel in the image. On the other hand, the cluster analysis stage is only interested in the actual amount of wheels in the image. Furthermore, there were no false positive windows left in the final classification result of this specific image, which corresponds to a perfect false positive rate. Once more, this could be compared to

the previous false positive rate after the classification stage, which were 0.059%. Given these points, the cluster analysis stage seem to improve both the true positive rate and false positive rate, compared to using the classification stage alone and is therefore examined further.

At this moment, due to computational constraints and time consumption of the program, it is a rather overwhelming task to evaluate a satisfying amount of images that would be necessary to examine the cluster analysis correctly. Nevertheless, ten different images are illustrated in Figure 5.10 that can give an indication of the performance of the classification system when the cluster analysis is invovled. Green windows correspond to true positive windows and red windows correspond to false positive windows.

If the images are visually examined before starting the classification system, it is easy to see how many wheels there are and what the desired amount of detected wheels should be. For this reason, the true positive rate is fairly easy to derive. On the other hand, the false positive rate is more difficult to calculate because there is no obvious amount of false positive windows in the images. Under these circumstances, the false positive rate is based on the known amount of extracted negative windows from all the images combined.

As can be seen in Figure 5.10, there are 25 car wheels in total that are fully visible and are of interest for detection and recognition in the classification system. The system classify 23 of these correctly, therefore the true positive rate (TPR) is 92%. On account of this, two wheels in total were misclassified giving a false negative rate (FNR) of 8%, according to Equation 2.4 on page 9. There are 12 windows in the images that are misclassified as a wheel. The total number of extracted false positive windows is $1,328,400$. Therefore, the false positive rate (FPR) is 0.0009%.

In conclusion, the cluster analysis stage, based on these examined images, generates an improved true positive rate and false positive rate compared to the previous values from Section 5.4, which corresponds to the values obtained after the classification stage in the system. However, the amount of examined images needs to be increased in order for the cluster analysis stage to be completely evaluated. These presented results only give an indication of how the cluster analysis stage can improve the result.

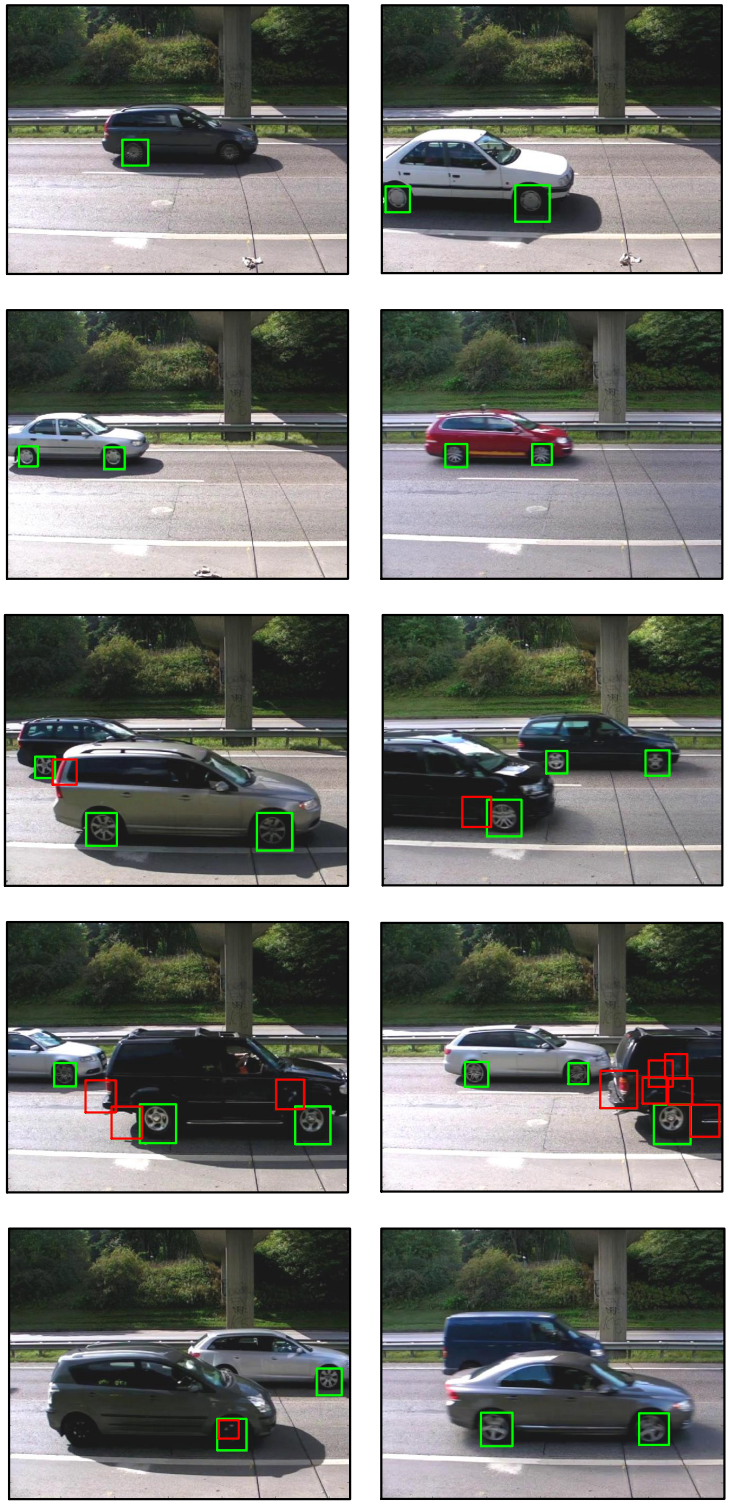

Figure 5.10: The classification result after the cluster analysis stage. The green windows correspond to true positive windows and the red windows correspond to false positive windows.

Chapter 6

Conclusion

In this master thesis, a classification system has been generated to detect and recognize wheels in images based on Random Forest classification. This could work as a foundation for a vehicle counting and classification system in the future, with image based techniques.

The classification system is based on five stages; multi-scale transformation, window extractor, pre-processing, classification and cluster analysis. These stages can be summarized as followed.

1. Multi-scale transformation
   It transforms the input image into grey scale. Thereafter it rescales the image four times by a predefined factor in order to create variations of the size of the wheels in the image.

2. Window extractor
   Because the location of a wheel in an image is unknown, this stage is used for extracting an enormous amount of windows of size 32x32 pixels from each scaled image. However, by using knowledge of the image scene, the entire images do not need to be examined, which results in a higher execution speed.

3. Pre-processing
   For each window a feature extraction is made, which results in a feature vector used as input to the classifier. The feature examined in this work corresponds to normalized intensity and Local Binary Pattern.

4. Classification
   This is the main stage of the classification system. The learning method used for generating the classifiers is Random Forest, which is constructed by several decision trees in a combination. In order to reduce the amount

of misclassifications of negative windows, this classifier becomes improved using one of the two methods: bootstrap configuration and cascade classification. Both of these methods are based on Random Forest as a learning method. However, they adapt the database in order to decrease the amount of misclassifications.

5. Cluster analysis
   The final stage of the classification system is to combine those predicted windows that match the same wheels. Moreover, it removes those predictions that are sparsely located, based on that they probably correspond to false predictions.

In this master thesis, the evaluation of the result is divided into two parts depending on where in the classification system the results are examined.

The first part contains the result after the classification stage of the system, stage $1-4$ above, and is evaluated by means of Receiver Operating Characteristics. As previously mentioned, the two improvement methods are in this work generated with different databases. Therefore, if the aim is to use a classification stage that results in the best general result, then the bootstrap configuration method should be used. It produces a true positive rate of 94.14% and a false positive rate of 0.309%. Despite this, if the aim is to reduce the number of misclassifications of negative windows as much as possible while receiving a satisfying true positive rate, then the cascade classification should be used instead. On account of this method, the true positive rate is 68.38% and the false positive rate is 0.028%.

The second part examines the result from the entire classification system, stage $1-5$ above, including the cluster analysis, when the cascade classification was used. At this stage, all of the extracted windows from each examined image need to be evaluated in the system. Due to computational constraints and time consumption of the program, only ten input images with their corresponding extracted windows are evaluated. Therefore, the presented result can only give an indication of how the cluster analysis affects the final classification system. Given these points, the true positive rate is derived to be 92% and the false positive rate is 0.0009%. In order to obtain a more certain result, the evaluation needs to contain a larger amount of images.

6.1 Future work

In order to achieve an even higher true positive rate while reducing the false positive rate even further, the database for teaching the classifier could be improved. At this moment, some of the negative windows have quite similar characteristics as a positive window. If the definitions for a negative and a positive window are reviewed, then this could be avoided by excluding those similar negative windows from the database. As a second improvement, if the amount of examples in the database is increased then the classifier would gain more knowledge during its training.

If this master thesis should operate as a foundation for a general system for vehicle counting and classification, then the wheels from other vehicle types that were excluded during this work should be included. As a result, another classifier could be generated that is able to detect and recognize these as well. Furthermore, an optimization of the program could lead to faster execution time, which is a requirement if this system is to operate in real-time. This could also be of importance for a further evaluation of the cluster analysis.

To conclude, an interesting continuation of this master thesis is to introduce some sort of tracking technique to recognize the same wheel in different images. This could be of an assistant to evaluate which wheels that belongs to the same vehicle. Based on this knowledge together with the location of each wheel, the vehicles can be assigned to their corresponding vehicle class when the distances between subsequent wheel pairs are calculated.

Bibliography

[1] Vägtrafik- och hastighetsdata. Accessed: 30/11/2012. http://www.trafikverket.se/Foretag/Trafikera-och-transportera/Trafikera-vag/Verktyg-e-tjanster-och-vagdata/Vagtrafik--och-hastighetsdata/.

[2] Ofer Achler and Mohan M. Trivedi. Vehicle wheel detector using 2d filter banks. In *IEEE Intelligent Vehicles Symposium*, pages 25–30, June 2004.

[3] Timo Ahonen, Abdenour Hadid, Matti Pietikäinen, and Guoying Zhao. *Computer Vision Using Local Binary Patterns*, volume 40 of *Computational Imaging and Vision Ser.* Springer, June 2011.

[4] Timo Ahonen and Matti Pietikäinen. Face description with local binary patterns: Application to face recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28 (12), pages 2037 – 2041. IEEE Comput. Soc, December 2006.

[5] S. Sheik Mohammed Ali, Niranjan Joshi, Boby George, and Lelitha Vanajakshi. Application of random forest algorithm to classify vehicles detected by a multiple inductive loop system. In *15th International IEEE Conference on Intelligent Transportation Systems*, pages 491–495, September 2012.

[6] Richard A. Berk. *Statistical Learning from a Regression Perspective.* Springer series in statistics. Springer New York, 2008.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 24 (2):123–140, August 1996.

[8] Leo Breiman. Random forests. *Machine Learning*, 45 (1):5–32, 2001.

[9] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8 (6):679 – 698, November 1986.

[10] Jie Chen, Li hua Dou, Juan Zhang, and Li hui Zou. The comparison of two typical corner detection algorithms. In *Second International Symposium on Intelligent Information Technology Application*, volume 2, pages 211 – 215, December 2008.

[11] Andreas Christmann and Ingo Steinwart. *Support Vector Machines*. Information Science and Statistics. Springer, 2008.

[12] Alin Dobra. Decision tree classification. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, Springer reference, pages 765–769. Springer US, 2009.

[13] Wei Fan and Kun Zhang. Bagging. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, Springer reference, pages 206–210. Springer US, 2009.

[14] Janusz Gajda, Ryszard Sroka, Marek Stencel, Andrzej Wajda, and Tadeusz Zeglen. A vehicle classification based on inductive loop detectors. In *IEEE Instrumentation and Measurement Technology Conference*, volume 1, pages 460–464, May 2001.

[15] Dariu Gavrila and Stefan Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International Journal of Computer Vision*, 73 (1):41–59, May 2007.

[16] Zhang Genshan, Han Guodong, Geng Kai, and Wu Qiuhong. Video analysis system of intelligent surveillance based on bayesian. In *International Conference on Computer Science and Network Technology*, volume 2, pages 1008 – 1011, December 2011.

[17] David J. Hand and Wojtek J. Krzanowski. *ROC Curves for Continuous Data*. Monographs on Statistics and Applied Probability 111. CRC Press, 2009.

[18] Marko Heikkilä and Matti Pietikäinen. A texture-based method for modeling the background and detecting moving objects. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28 (4), pages 657 – 662, April 2006.

[19] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid. Description of interest regions with local binary patterns. In *Pattern Recognition*, volume 42 (3), pages 425 – 436. Elsevier B.V, 2009.

[20] Michael J. Jones and Paul Viola. Robust real-time face detection. *International Journal of Computer Vision*, 57 (2):137–154, May 2004.

[21] Zhou Junjing, Duan Jianmin, and Yu Hongxiao. Machine-vision based preceding vehicle detection algorithm: A review. In *10th World Congress on Intelligent Control and Automation*, pages 4617–4622, July 2012.

[22] Nehal Kassem, Ahmed E. Kosba, and Moustafa Youssef. Rf-based vehicle detection and speed estimation. In *IEEE 75th Vehicular Technology Conference*, pages 1–5, May 2012.

[23] Jinhui Lan and Meng Zhang. A new vehicle detection algorithm for real-time image processing system. In *International Conference on Computer Application and System Modeling*, volume 10, pages 1–4, October 2010.

[24] Ruixin Niu, Pramod K. Varchney, and Min Xu. Detection and tracking of moving objects in image sequences with varying illumination. In *International Conference on Image Processing*, volume 4, pages 2595–2598, October 2004.

[25] Xinting Pan, Yunlong Guo, and Aidong Men. Traffic surveillance system for vehicle flow detection. In *Second International Conference on Computer Modeling and Simulation*, volume 1, pages 314 – 318, January 2010.

[26] Sompoch Puntavungkour. Vehicle detection from three line scanner image. In *IEEE Intelligent Transportation Systems*, volume 1, pages 785–788, October 2003.

[27] C. Setchell and E.L. Dagless. Vision-based road-traffic monitoring sensor. *IEE Proceedings - Vision, Image and Signal Processing*, 148 (1):78–84, February 2001.

[28] Zehang Sun, George Bebis, and Ronald Miller. On-road vehicle detection using gabor filters and support vector machines. In *International Conference on Digital Signal Processing*, volume 2, pages 1019–1022, 2002.

[29] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (1):39–51, January 1998.

[30] Pang-Ning Tan. Receiver operating characteristic. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, Springer reference, pages 2349–2352. Springer US, 2009.

[31] Luo-Wei Tsai, Jun-Wei Hsieh, and Kuo-Chin Fan. Vehicle detection using normalized color and edge map. *IEEE Transactions on Image Processing*, 16 (3):850–864, March 2007.

[32] Lipo Wang. *Support Vector Machines: Theory and Applications*, volume 177 of *Studies in fuzziness and soft computing*. Springer, 2005.

[33] Graham Williams. *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. Use R. Springer New York, 1 edition, 2011.

[34] Hwanjo Yu. Bootstrap. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, Springer reference, page 264. Springer US, 2009.

[35] Wei Zheng and Luhong Liang. Fast car detection using image strip features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2703–2710, June 2009.

Master's Theses in Mathematical Sciences 2013:E7
ISSN 1404-6342

LUTFMA-3240-2013

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/