Evolving Cascades of Voting Feature Detectors for Vehicle Detection in Satellite Imagery

Krzysztof Krawiec, *Member, IEEE* and Bartosz Kukawka and Tomasz Maciejewski

***Abstract*—We propose an evolutionary method for detection of vehicles in satellite imagery which involves a large number of simple elementary features and multiple detectors trained by genetic programming. The complete detection system is composed of several detectors that are chained into a cascade and successively filter out the negative examples. Each detector is a committee of genetic programming trees that together vote over the decision concerning vehicle presence, and is trained only on the examples classified as positive by the previous cascade node. The individual trees use typical arithmetic transformations to aggregate features selected from a very large collections of Haar-like features derived from the input image. The paper presents detailed description of the proposed algorithm and reports the results of an extensive computational experiment carried out on real-world satellite images. The evolved detection system exhibits competitive sensitivity and relatively low false positive rate for testing images, despite not making use of domain-specific knowledge.**

I. Introduction

The task of vehicle detection in visible band in aerial and satellite imagery is challenging for several reasons: low spatial resolution, heterogeneity of the positive class (various body styles and colors), uncontrolled lighting, partial occlusions (e.g., by trees), and other undesired phenomena (e.g., strong sunlight reflexes from wind shields). The presence of man-made objects that closely resemble cars (like air conditioning equipment on rooftops, cargo containers, etc.) can additionally confuse the learner and give raise to false positive detections. And, last but not least, this is a typical needle-in-a-haystack task: the *a priori* probability of vehicle presence at particular image location is typically very low, so the negative examples outnumber the positive ones by several orders of magnitude. These features make car detection a challenging and interesting testbed for pattern recognition algorithms, with numerous actual and potential real-world applications.

Vehicle detection methods typically engage some form of sliding window (window of attention, region of interest) to focus on single image fragment at a time. Considering the approach, they can be roughly divided into matching-based and feature-based. In the former category, recognition system stores a library of positive examples (models) and recognition consists in a, more or less literal, measurement of similarity of the image part visible in the window to the models. In the latter, there are no explicit models; rather than that, some features are extracted from the window and typically fed into a previously trained machine learning classifier that makes the final recognition decision.

K. Krawiec, B. Kukawka, and T. Maciejewski are with the Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60965 Poznań, Poland; email: kkrawiec@cs.put.poznan.pl.

This study falls into the latter category, and may be summarized as evolutionary learning a cascade of compound detectors based on simple, Haar-like image features. Each detector is composed of a few genetic programming (GP, [5]) expressions; each of them works by aggregating selected image features and outputs a number that is appropriately interpreted for the sake of detection. The only training information it expects are the true locations of cars in training images; no other common-sense or domain-related knowledge is used. In particular, there is no prior preselection of the most likely vehicle locations (roads, parking lots, etc.) – any coordinates in the input image are considered as potentially containing a car.

The next section reviews the related work. Its great part is devoted to the canonical object detection algorithm by Viola and Jones [12], which we consider as a blueprint for our approach, presented in Section III. In Section IV we discuss the results of experiment, concluding in Section V.

II. Related Work

This paper is partially inspired by the seminal work of Viola and Jones [12, VJ in following] on feature-based face detection, which we will now present in short. Typically for detection, one assumes there that a rectangular window of attention (field of view) slides over the input image. For each position of the window, the detector is queried and returns a discrete response, positive or negative. Detector responses for particular locations are clustered, leading to final decisions of the system. In case of training images, these decisions may be then confronted with the ground truth (the actual locations of object to be detected), leading to typical performance measures, like sensitivity and selectivity.

Viola & Jones use Haar wavelets (called *Haar Basis functions* in [12]) to generate image features. Computing each of them requires convolving a binary mask defined by a specific Haar wavelet with the window image. Viola & Jones decided to exhaustively generate all Haar features from the input window. For typical window sizes used for detection, this process results in tens of thousands of features, some of them working as edge detectors, and some others having on-center, off-surround characteristics. A clever computational trick, the so-called *integral image*, allows calculating all such features at low cost.

The features calculated from training images form the training set for the cascade of compound AdaBoost classifiers, which operates as follows. The input window image

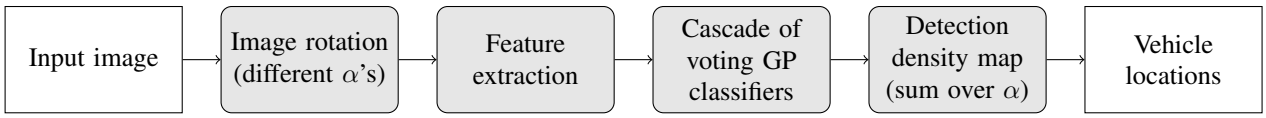

Fig. 1. The overall processing stages of the proposed vehicle detection system.

enters the first classifier C_1 of the cascade that can classify it as negative (non-face), in which case processing ends. If C_1 classifies the image as positive (face), it is passed to the next node of the cascade, C_2. There, this scheme is repeated: the image is either rejected as negative or passed to C_3. As a result of consecutive steps, the image can eventually reach the last cascade node C_n. Only if C_n classifies the image as positive, the overall decision of the entire cascade is considered as positive. The essential feature of this approach is that the subsequent classifiers make use of the decisions made at the preceding nodes. In particular, C_i is trained only on the examples that have been classified as positive by $C_1, C_2, \ldots, C_{i-1}$. Thus, the further stages of the cascade have to learn how to reject the not-yet-rejected negative examples while passing through the positive ones.

Within each cascade node, VJ employ the AdaBoost learning algorithm [2]. AdaBoost works by incrementally building a pool of simple (weak) base classifiers and combining their outputs by weighting so that their aggregated response is as consistent as possible with the original assignment of objects to decision classes. This learning process involves also weighting of training examples. Detectors trained using AdaBoost usually tend to perform very well, despite the simplicity of base classifiers, which (in the basic variant of the approach) are so-called decision stumps, i.e., degenerated decision trees that contain only root node. In other words, a base classifier makes decision by thresholding a value of a single Haar feature.

Cascaded AdaBoost proves very effective in practice and the VJ approach is currently widely cited. The algorithm has been implemented in the popular open-source computer vision library OpenCV [1]. Further extensions have been proposed, e.g., Wu *et al.* successfully reduced method's appetite for processor power and memory during training [13]. No wonder that also the EC community followed to exploit these ideas. In [8], Lichodziejewski *et al.* used a cascade of GP trees for data mining tasks. Howard *et al.* in [3] employed a cascade of GP trees for detection of vehicles in infrared imagery, using predefined, quite sophisticated features of different nature (local edge density and texture characteristics) and one GP tree per cascade node.

III. The Method

The outline of our vehicle detection method is presented in Fig. 1. After preliminary image preprocessing (*Image rotation*, described later) we extract image features and fed them into the cascade of voting GP-based classifiers. The responses of particular classifiers are aggregated over input image rotated by different angles α and form quasi-continuous detection density map (DDM), which values reflect the likelihood of vehicle presence. DDM pixels are subsequently discretized into final decision for each pixel.

In the following subsections, we focus on feature extraction and classifiers. The details on image rotation will be provided in the experimental section.

A. Elementary Features

We use quad trees to define multiscale features that on one hand are visually similar to Haar wavelets and, on the other hand, are easier to generate and index in a systematic way. Given the 32×32 pixel input window, we stack a uniform quad tree over it so that the tree nodes correspond one-to-one to rectangular image regions (referred to as *tiles* in following). The nodes at consecutive depths correspond to 16×16, 8×8, 4×4, and 2×2 tiles; there are, respectively, 4, 16, 64, and 256 of them, giving the total of 340 quad tree nodes.

Every quad tree feature is uniquely identified by *quad key* – a variable-length sequence of quaternary digits. Each digit encodes the choice of one of four tiles within specific quad tree level: 0, 1, 2, 3 for upper left, upper right, lower left, and lower right tile, respectively. For instance, quad key 30_4 encodes the upper left 8×8 tile within the lower-right 16×16 tile.

Feature value is simply the average (absolute) brightness of tile pixels. To measure the *relative* brightness differences, we enable pairing of elementary features as described in Section III-B. This gives the total of $340 \times 340 = 115,600$ features, a number comparable to that of VJ approach. Note however that our features are qualitatively different from Haar wavelets: a paired feature can refer to two non-adjacent tiles, which cannot be expressed using Haar wavelets.

Because our features are based on rectangular image regions, we can still use the integral image to compute them effectively, as it was the case with Haar wavelets used in VJ approach [12]. Given an image A with brightness values a_{xy}, its integral image B is defined as $b_{xy} = \sum_{i \le x,\, j \le y} a_{ij}$ – the value stored in B at position (x, y) is the integral of brightness over the rectangle that spans the beginning of the coordinate system and the (x, y) coordinates. Given B, a total brightness of *any* rectangle in A, like those rectangles defined by our quad tree features, can be calculated by fetching just four values from B and adding and subtracting them appropriately. As the integral image can be easily calculated incrementally in one pass over the entire image, this technique brings in essential speedup an makes the process of feature extraction almost instantaneous.

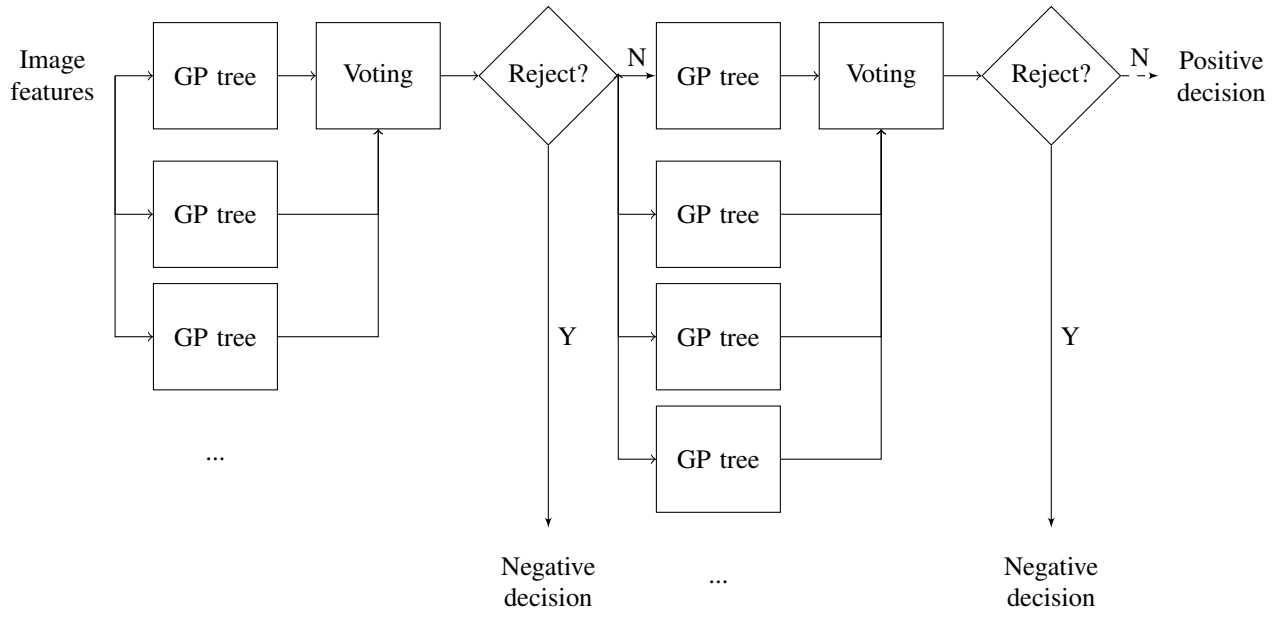

Fig. 2. The cascade of voting committees of GP classifiers. Each cascade node is a set of GP trees that make independent decisions that are subsequently aggregated via voting. In both training and testing, only the examples classified as positive by a cascade node reach the subsequent node. An example has to pass through all cascade nodes to become ultimately classified as positive.

B. Cascades of GP Base Classifiers

The essential feature of cascade architecture is sequential processing: a compound classifier in particular node learns only from examples classified as positive by its predecessors, and its training and decision making is otherwise completely independent from the other nodes. This inclined us to devote a separate Genetic Programming (GP) training process (evolutionary run) to each of n cascade nodes. In a node, we implement an individual as a vector of k GP trees, each of them serving as a base classifier. The final result of learning is the cascade assembled from the n best-of-run individuals of consecutive runs (see Fig. 2).

As in our former work in evolutionary image analysis and evolutionary feature synthesis [7], [6], we evolve both the *choice* of elementary features as well as their *aggregation*: a base classifier fetches the values of elementary features using leaves (terminal nodes), processes them by means of elementary functions, and returns the overall output at the root node of the tree. We employ strongly typed GP [10] with three data types: constants ($\mathcal{C}$), variables (feature values, $\mathcal{V}$) and image-dependent values ($\mathcal{N}$). $\mathcal{N}$ is a set type and embraces type $\mathcal{V}$. Trees are initialized by means of standard ramped-half-and-half method using the provided set of functions and terminals (leaves). The function set embraces arithmetics ($\mathcal{N} \times (\mathcal{N} \cup \mathcal{C}) \rightarrow \mathcal{N}$: $+$, $-$, $\times$, $/$) and $\tanh()$ function ($\mathcal{N} \rightarrow \mathcal{N}$) to provide simple form of nonlinearity. The terminals include ephemeral random constants (ERCs, type $\mathcal{C}$) and inputs (feature values, type $\mathcal{V}$). ERCs return floating-point constants drawn randomly from interval $[-3, 3]$.

To avoid defining a separate terminal for each of 340 features, we implemented the input variables as *parametric* terminals $q(k)$ and $d(k, l)$, where k and l are quad keys (see Section III-A). The former one returns the value of quad tree feature #k, i.e., the mean brightness of the image tile identified by quad key k. The latter one is equivalent to $q(k) - q(l)$, and has been introduced to ease the evolution of differential features that mimic Haar wavelets. Figure 3 presents an exemplary GP tree and the definitions of features it fetches from the input image using its terminal nodes.

Technically, parametric input nodes are similar to ephemeral random constants (ERCs, [5]), the only difference being that the value of the constant (quad key), rather than being directly returned, is used to 'address' a specific tile in the quad tree and fetch its mean brightness level. For this reason, our input nodes undergo mutation in the same way as ERCs, i.e., by perturbing the value stored by the node.

C. Aggregation schemes

A GP tree in our setting corresponds to a base classifier in VJ and returns a continuous value reflecting its confidence in car presence at particular location. These output values have to be aggregated to form the ultimate decision made by particular cascade node. We considered a couple of aggregation schemes, including sum and majority voting, and used them in preliminary experiments. The one that worked the best was majority voting. In this scheme, only the sign of tree output matters: output greater than or equal to 0 is interpreted as positive decision, otherwise the decision is negative. The positive and negative decisions are counted and the overall decision is made on the basis of majority rule.

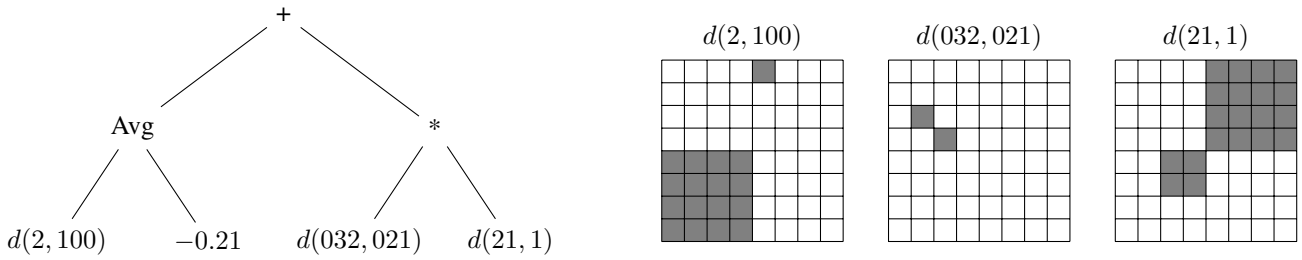

Fig. 3. An exemplary GP tree (left) and the image features fetched by its terminal nodes (right). Each feature is a difference of total brightness in two rectangular regions addressed by quad-tree codes. For clarity, only three uppermost levels are illustrated by the grid, so that it corresponds to the entire 32×32 input window and a single grid cell corresponds to a square of 4×4 pixels.

D. Method summary

The essential features of our approach are following:

1) The training process has access to a very large collection of features. There is no preliminary feature selection that could potentially harm the training process by accidentally getting rid of useful features.
2) The features used by a particular trained GP trees are dynamically adjusted by the evolutionary process. In particular, encoding image features as parameterized terminal nodes that identify the features by quad codes, makes the mutation process more natural: a small mutation of quad code is likely to lead to a similar feature.
3) The evolutionary training process is aimed at evolving a committee of relatively simple GP trees that perform well as a team, rather than trying to elaborate one sophisticated GP tree.
4) The step-wise (cascaded) reasoning allows us to cope with the extremely imbalanced characteristics of the task: 'easy' negative examples are likely to be rejected at early processing stages (cascade nodes).

Though the overall working principle is similar here to the VJ method, there are several substantial differences. Let us outline the most important of them:

1) We use systematically generated quad-tree-based features rather than Haar features.
2) We allow more sophisticated base classifiers, defining them as GP expressions that can aggregate *many* features and allow non-linear mapping from the feature space into decision space.
3) There is no explicit weighting of base classifiers. Rather than that, we use simple non-parametric aggregation schemes to merge the decisions of base classifiers into the overall decision made by cascade node.
4) AdaBoost performs an incremental greedy local search in the space of base classifiers, always choosing as next the locally best base classifier *given* the subset of already selected classifiers. We perform evolutionary search, in which any component of solution may be revised at any step of search.

IV. The Experiment

A. The Data

Our image database comprises 33 true-color satellite images of spatial resolution 0.2m/pixel, representing different environments: urban, rural, parking lots, bridges, etc. The bottom part of Fig. 5 presents a representative image from that collection. We converted these images to grayscale and manually marked contours/silhouettes of cars of different types (sedans, station wagons, pickups, convertibles). Trucks and other non-typical cars (e.g., articulated vehicles) are not marked and considered as negative examples. The number of positive examples varied from 4 to 378 per image. The acquired contours constitute our ground truth and are subsequently used to generate the training images, evolutionary training (fitness function), and post-evolution testing.

Each training example is a 32×32 fragment of the original satellite image, either containing a vehicle (positive example) or not (negative example, 'clutter'). Due to particular feature definition used here, our detector assumes that the cars are aligned vertically within the field of view. Because in practice the orientation (azimuth) of the car may be arbitrary (as opposed to face detection, where vertical orientation is typically implicit), we rotate the positive examples so that the long axis of the car is vertical (the front and the rear of the car are not distinguished).

The 33 images have been divided into training part (24 images) and testing part (9 images). This has been done semi-randomly, i.e., while trying to keep the training set representative in terms of environments. The training set contains 659 positive examples extracted from training images, and the testing set includes 635 cars taken from the testing part. Let us emphasize how very unbalanced are the decision classes in this task: given that an average car is 9 pixels wide and 22 pixels long, the total area occupied by all cars in all 33 images accounts for less than 1.5 percent of total area of all images, and a hypothetical ideal detector would give positive response for less than 0.01 percent of all possible locations of detection window.

B. The Training Process

The training process consists in a series of $n = 5$ evolutionary runs (see Section III-B), each dedicated to one cascade node. Each individual comprises 5 GP trees that

TABLE I
STATISTICS OF CLASSIFICATION IN THE TRAINING PHASE. THE FIGURES REFLECT THE OUTCOMES OF CLASSIFICATION ATTAINED BY CONSECUTIVE NODES OF THE CASCADE.

Cascade node	TP	TN	FP	FN
1	374	11865	135	285
2	368	87	48	6
3	367	18	30	1
4	367	5	25	0
5	367	8	17	0
Total	367	11983	17	292

vote over the decision concerning each training example in evaluation phase of the evolutionary run (cf. Fig. 2). When crossing over individuals, we allow for exchange of genetic material between any of their constituent trees.

Concerning GP-specifics, we use the Koza-I-style setup [5] with some modifications. The most important settings include: population of 1024 individuals initialized using the standard ramped half-and-half method, tournament selection with tournament of size 7, tree-swap crossover engaged with probability 0.9, subtree-replacing mutation applied with probability 0.1, no elitism. The tree depth limit has been set to 10. Evolution lasts for 100 generations. We use the function set defined in Section III-B. Other parameters of evolution use the Koza-I [5] defaults as specified in ECJ [9], the software package that served as basic framework for our implementation.

Because of high imbalance of positive and negative classes, rather than relying on accuracy of classification, we use F-measure to estimate the fitness of an individual (detector), i.e., the harmonic mean of *precision* p and *recall* r (sensitivity)

$$fitness = F_{measure} = \frac{2pr}{p+r},$$

where

$$p = \frac{TP}{TP+FP}, \quad r = \frac{TP}{TP+FN}$$

Table I presents the statistics of classification collected during the training process. We observe there how the consecutive cascade nodes successfully filter out the false positives, while the number of true positives is only slightly affected and does not decrease after reaching the third node of the cascade. Unfortunately, we witness here also the weakness of the cascaded model of processing: the false negative errors, once committed, cannot be corrected by subsequent nodes. On the training data, this detector attains precision $p = 0.956$, recall $r = 0.557$, and F-measure of 0.704.

C. Detection Results for Test Images

In the post-evolution test, an evolved detector is applied to testing images. This process requires scanning the image and analyzing detector's responses at particular locations. Because our Haar-like features are based lack rotational

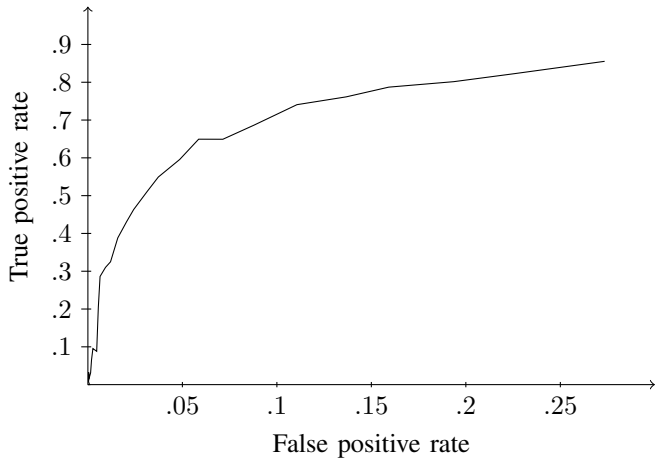

Fig. 4. ROC curve for the evolved cascade of detectors.

invariance, for each test image I, we create its $l = 8$ versions I_i, $i = 1, \ldots, l-1$ rotated by $180i/l$ degrees (i.e., 22.5 degrees for $l = 8$ used here). For each rotated image I_i, the detector scans all locations and returns a binary (positive/negative) response for each pixel. The coordinates of positive responses are mapped back (by reverse rotation) onto the coordinate system of the original image I. Based on these co-registered coordinates of positive responses, we build the detection density map (DDM).

DDM is a raster image, with all pixels initially zeroed. We iterate over all positive detector responses and, assuming that each of them increases the local likelihood of vehicle presence, we increment the corresponding pixel in DDM and its neighborhood pixels according to two-dimensional Gaussian distribution with small standard deviation $\sigma = 2.6$ pixels. As a result, close positives tend to reinforce each other. Next, we detect local maxima in DDM; if the DDM value at given location (x, y) exceeds a predefined threshold t, we issue positive decision for (x, y).

Figure 5 demonstrates the outcome of this process for an exemplary image of dimensions 884×584 pixels. The original input image, shown in the bottom part of the Figure, has been subject to detection process that produced the DDM shown in the top part. Darker pixels in DDM indicate more evidence in favor of vehicle presence. One can observe how, for the more evident car locations, the multiple Gaussians aggregated in DDM reinforce each other.

If the detection at location (x, y) falls within the contour of a car, we count it as true positive (TP) and subsequent hits within the same contour are ignored. Otherwise, we register a false positive (FP) error. The red crosses overlaid on the input image in the lower part of Fig. 5 mark the detections derived from the DDM using this procedure. It can bee seen that the thresholding effectively removes many 'weak' detections.

The quantify the overall detection performance we assume, for both true positives and false positives, that the 'unit' of detection is an equivalent of car area (approximately 200 pixels): the true positive rate is defined as the number of

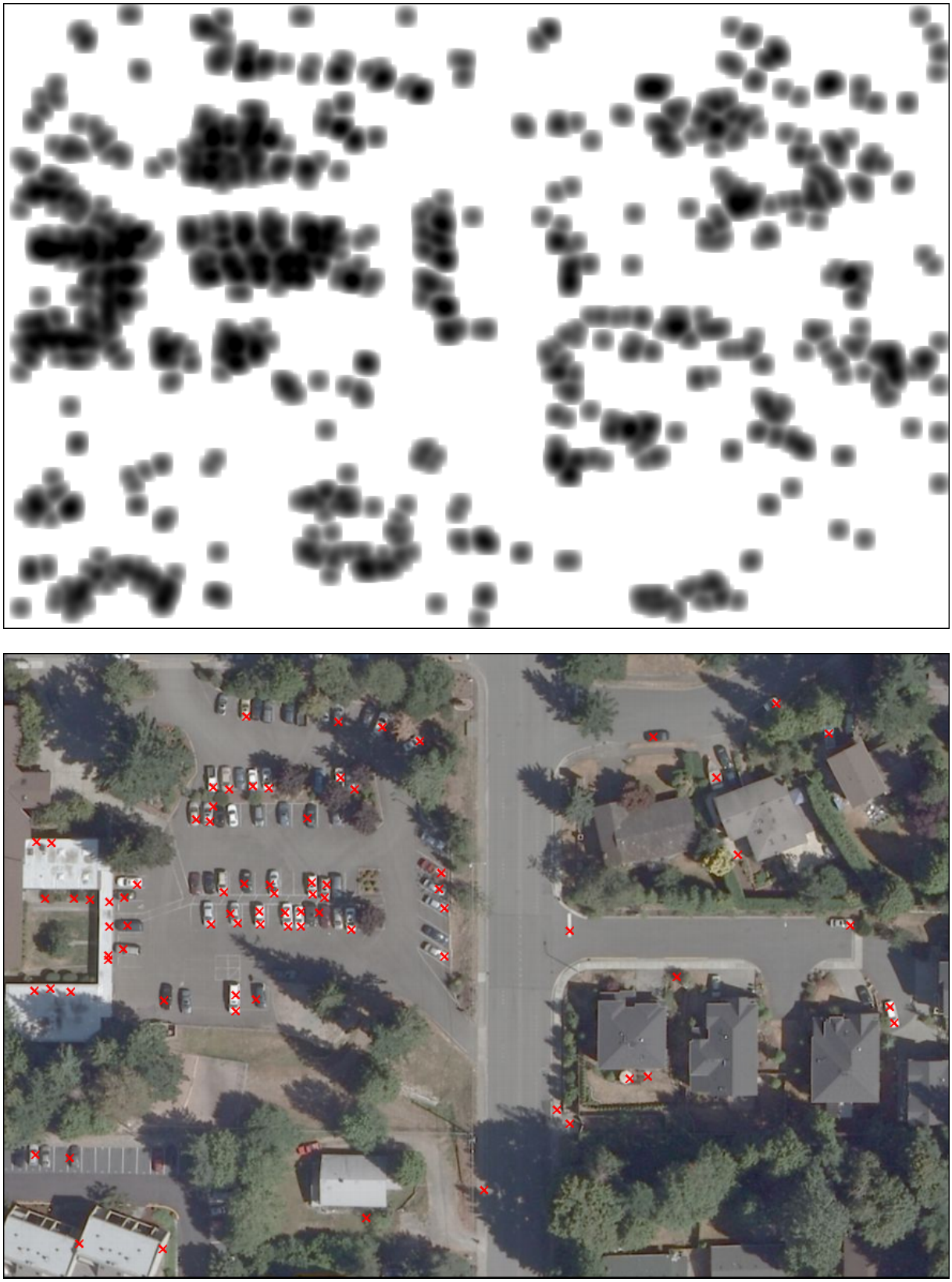

Fig. 5. Detection density map DDM (top) and the corresponding final detection result (bottom). For the DDM, the darker the pixel, the more evidence has been collected for the presence of vehicle at a particular location. In the bottom image, red crosses indicate the positive decisions of the system.

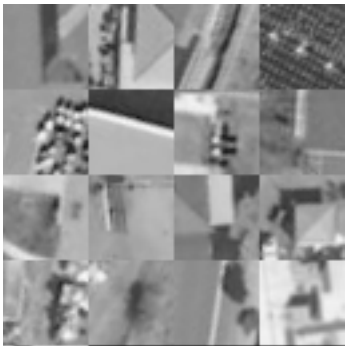

Fig. 6. Exemplary false-positive detections.

Fig. 7. Exemplary false-negative errors.

true positives divided by the actual number of cars in an image, while the false positive rate is the number of false positive detections divided by the number of cars that could be squeezed into the negative region (i.e., the number of non-car image pixels divided by the average car area in pixels).

Figure 4 presents the ROC curve spanned on these ratios. It has been obtained from the best evolved detector by varying the DDM threshold t. The smaller t, the easier it is for the evidence gathered in DDM to yield positive decision. The area under the curve (AUC) amounts in this case to 0.8666. Using ROC curve and tuning the value of t, one can realize different trade-offs between recall and selectivity. E.g., for $t = 0.35$, sensitivity amounts to 0.65 and false positive rate to 0.07.

Figures 6 and 7 present exemplary errors committed by our evolved cascade. While the reasons of false positives are rather obscure and call for in-depth analysis of the evolved programs, the causes for false negatives are more clear. They can typically result from low contrast within the input window, presence of other cars in immediate proximity, occlusion by trees, or long shadow cast by the car.

In its current shape, the detection process is quite time-consuming, which is mostly due to the fact that the image has to be scanned multiple times for different rotation angles. To partially remedy this problem and reduce the processing time, we employed the following technical solution. After training the complete cascade of voting committees of GP trees we automatically convert them into an independent class of the Java programming language. That class is subsequently used for testing. Because it comprises mostly arithmetic expressions and conditions that encode the particular GP trees, it runs much faster than the internal representation of GP trees used in the ECJ software package.

V. Discussion and Conclusions

In this paper we demonstrated that genetic programming combined with cascaded processing and using very simple elementary features can attain decent performance on a challenging task of vehicle detection in satellite imagery. We dare to claim that this result is notable, given the fact that image resolution is low (average car dimensions are here 9×22 pixels) and the method operates completely autonomously and without help of any extra domain knowledge.

Though this result is inferior when compared to human performance, one should keep in mind that humans unconsciously rely on contextual information which immensely helps detection. No human would consider a rooftop as a reasonable place to look for vehicles. However, in absence of such contextual information, human performance can dramatically decrease. Figures 8 and 9 present respectively the particularly confusing false positive detections and false negative errors. Many of these mistakes are likely to be made also by humans.

Aware of the potential advantages brought by extra knowledge, designers of some vehicle detection algorithms take it into account for the benefit of performance. For instance, a neural network-based detection method described in [4] attains sensitivity of 0.86 while maintaining very low number of false positives (0.008 percent), but does so by constraining the search to road networks only, which, by the way, are not automatically detected but retrieved from GIS. Similarly, the authors of [11] use a separate preprocessing stage that detects streets based on 2.5D elevation data and report slightly worse results (sensitivity around 0.75 at 0.05 false positive rate). Our approach does not exploit such additional sources of information.

While conducting experiments, we came up with several ideas concerning the possible extensions and improvements of this approach. Apart from the possibility of exploiting other output aggregation schemes (see Section III-C), these include: taking color information into account (many cars have colors that are infrequent in nature), delegating separate cascades for detection of bright and dark cars, and using multiobjective evolutionary search to avoid aggregating precision and recall into one scalar fitness value (as it likely leads to compensation). We plan to exploit some of these ideas in future research.

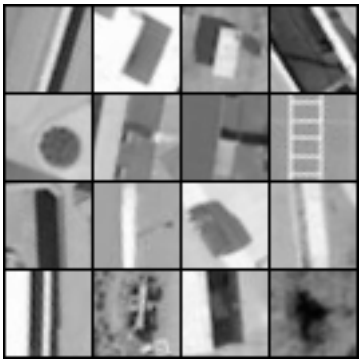

Fig. 8. Particularly confusing false positive detections. Some of these objects are likely to be erroneously classified as cars also by humans if presented without a wider context.

Fig. 9. Particularly confusing false negative errors. Some of these objects are likely to be erroneously classified as non-cars also by humans if presented without a wider context.

ACKNOWLEDGMENT

We would like to thank W. Jaśkowski and B. Wieloch for preparing the training data and labelling the images. This work was supported in part by Ministry of Science and Higher Education grant # N N519 3505 33 and 91-493/10-DS.

REFERENCES

[1] *Open Source Computer Vision Library: Reference Manual*, 2001.

[2] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In P. M. B. Vitányi, editor, *EuroCOLT*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 1995.

[3] D. Howard, S. C. Roberts, and C. Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, Aug. 2006. Evolutionary Computer Vision and Image Understanding.

[4] X. Jin and C. H. Davis. Vehicle detection from high-resolution satellite imagery using morphological shared-weight neural networks. *Image Vision Comput.*, 25(9):1422–1431, 2007.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[6] K. Krawiec. Generative learning of visual concepts using multiobjective genetic programming. *Pattern Recognition Letters*, 28:2385–2400, December 2007. DOI: 10.1016/j.patrec.2007.08.001.

[7] K. Krawiec and B. Bhanu. Coevolutionary computation for synthesis of recognition systems. In *Proceedings of Computer Vision and Pattern Recognition Conference, Workshop on Learning in Computer Vision and Pattern Recognition CVPR*, 2003.

[8] P. Lichodzijewski, N. Zincir-Heywood, and M. Heywood. Cascaded GP models for data mining. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 2258–2264, Portland, Oregon, 20-23 June 2004. IEEE Press.

[9] S. Luke. ECJ evolutionary computation system, 2002. (http://cs.gmu.edu/ eclab/projects/ecj/).

[10] D. J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993.

[11] J. Pers, editor. *An Improved Car Detection using Street Layer Extraction*, Moravske Toplice, Slovenia, 2008.

[12] P. A. Viola and M. J. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 1311–1318. MIT Press, 2001.

[13] J. Wu, S. C. Brubaker, M. D. Mullin, and J. M. Rehg. Fast asymmetric learning for cascade face detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(3):369–382, 2008.