THESIS TITLE

Innopolis University

Thesis submitted to The Innopolis University in conformity with the requirements for the degree of Bachelor of Science.

presented by

Your Name

supervised by

Supervisor's name

Date

THESIS TITLE

(optional) dedication.

Contents

List of Tables

List of Figures

Abstract

abstract . . .

Chapter 1

Introduction

1.1 Spacing & Type

This is a section. This is a citation without brackets **?**. and this is one with brackets [**?**]. These are multiple citations: [**?**, **?**, **?**]. Here's a reference to a subsection: 1.1.1. The body of the text and abstract must be double-spaced except for footnotes or long quotations. Fonts such as Times Roman, Bookman, New Century Schoolbook, Garamond, Palatine, and Courier are acceptable and commonly found on most computers. The same type must be used throughout the body of the text. The font size must be 10 point or larger and footnotes[1] must be two sizes smaller than the text[2] but no smaller than eight points. Chapter, section, or other headings should be of a consistent font and size throughout the ETD, as should labels for illustrations, charts, and figures.

1.1.1 Creating a Subsection

Creating a Subsubsection

This is a heading level below subsubsection And this is a quote:

> Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

[1]This is a footnote.
[2]This is another footnote.

This is a table:

Table 1.1: This is a caption.

A	B
a1	b1
a2	b2
a3	b3
a4	b4

The package "upgreek" allows us to use non-italicized lower-case greek letters. See for yourself: $\upbeta$, $\boldsymbol{\upbeta}$, β, $\boldsymbol{\beta}$. Next is a numbered equation:

$$\|\boldsymbol{X}\|_{2,1} = \underbrace{\sum_{j=1}^{n} f_j(\boldsymbol{X})}_{\text{convex}} = \sum_{j=1}^{n} \|\boldsymbol{X}_{\cdot,j}\|_2 \tag{1.1}$$

The reference to equation (1.1) is clickable.

1.2 Theorems, Corollaries, Lemmas, Proofs, Remarks, Definitions, and Examples

Theorem 1. *Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.*

Proof. I'm a (very short) proof. □

Lemma 1. *I'm a lemma.*

Corollary 1. *I include a reference to Thm. 1.*

Proposition 1. *I'm a proposition.*

Remark. I'm a remark.

Definition 1. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition. I'm a definition.

Example. I'm an example.

1.3 Section with linebreaks in the name

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Chapter 2

Systematic Literature Review

2.1 Rationale

As a part of this thesis work, the systematic literature review was performed. The objectives of the work done include the following: to know what technologies, methods, and concepts we are going to work with, to get familiar with existing research on related topics and achievements in related fields, as well as to assess results of previous works and extent of their applicability to development of the system that is a subject of this dissertation. Other few goals were analysis of used tools and strategies and their effectiveness and efficiency and retrieval of information regarding reliable ways to evaluate the work of the system.

Furthermore, there is quite an amount of justification of decision to perform the systematic literature review instead of a classical one, namely that it is convenient enough to have all the scholarly sources sorted, classified and analyzed rigorously enough to exploit the accumulated information efficiently to proceed further.

2.2 SLR protocol development

2.2.1 Research questions

In this literature review, it is attempted to answer such research questions that they include maximum relevant information needed to know in order to implement the proposed system. They are formulated as follows:

- **RQ1:** What technologies, tools and methods are suitable for implementing a computer vision and machine learning based environment modelling system?

- **RQ2:** What is needed for it to be used as a part of a smart parking system?
- **RQ3:** What measures of efficiency are best to evaluate the system and what results should be achieved to compete state-of-the-art approaches?

2.2.2 Searching process

This section provides the specification of searching process, including used search resources, keywords and search queries, study inclusion and exclusion criteria, evaluation scheme and the related methodology.

Resources

The search included electronic sources only. The following sources and tools were used for the search:

- IEEExplore Digital Library
- ResearchGate
- Google Scholar
- ACM Digital Library
- Arxiv.org

This is the exhaustive list, as no more sources were used in the process.

Search queries

For the search, five key topics were identified:

- Computer vision
- Machine learning
- Software design
- Smart parking
- Data collection on vehicles

For each topic, key concepts were written down, then synonyms found. As the result, all searches were performed using OR-combinations of the following queries:

- (automatic OR smart OR IoT OR internet of things) AND (parking OR parking lot OR parking slot OR parking space) AND (utilization OR occupancy OR tracking OR detection OR monitor OR management) AND (sensor OR camera OR video)

- (trigger OR event OR conditional execution OR decision making) AND (patterns OR design principles OR implementation OR usability OR user-friendly OR gui OR user interface)
- (vehicle OR car OR moving object) AND (data OR dataset OR metrics OR information)
- (machine OR supervised OR unsupervised OR deep) AND (learning OR classification OR regression) AND (algorithm OR dataset OR evaluation OR application)
- (computer OR machine OR automated) AND (vision OR recognition OR image processing OR scene reconstruction) AND (moving object OR shape OR vehicle)
- parking AND lot AND occupancy AND detection AND computer AND vision

The queries were edited to fit in the search opportunities of each system, but overall the semantics was left the same.

2.2.3 Inclusion and exclusion criteria

To decide which papers will be included in the review the following inclusion criteria were employed:

- Found using search queries specified above
- Related to computer science or data collection
- Written in English
- Published not earlier than in 2000
- Journal and repository articles, conference proceedings, master's and doctoral degree dissertations
- Peer-reviewed
- Primary studies or systematic literature reviews

Papers that didnt fit at least one of inclusion criteria were excluded from the review, as well as duplicates.

2.2.4 Quality assessment

To assess the quality of included papers, a set of questions was developed. The yes answer yields 1 point, partially yields 0.5 points, no yields 0 points. The points obtained on every question are then added up. The questions are:

1. Are objectives clear and specific?

2. Does the research satisfy the objectives appropriately?
3. Is research process clear and reproducible?
4. Were the results assessed properly in a paper?
5. Were the results summarized to provide a clear conclusion?
6. Are there any comparisons with alternatives?

So basically the score range is 0 (the worst) to 6 (the best).

2.2.5 Search and synthesis strategies

For papers found using resources and queries specified above, papers were evaluated according to inclusion and exclusion criteria by one of researchers each. After that, from included papers, 20 were chosen at random to be assessed against quality assessment criteria by both researchers. For differences in judgement, the mean result was taken and patterns in disagreement identified to perform systematic corrections of scores of remaining papers after they are reviewed by one of researchers.

The synthesis was performed both qualitatively and quantitatively. Statistics about the papers was gathered, such as key topics, years of publication and quality assessment. Also, we classified the found information by methods and results.

2.3 Results

2.3.1 Search Sources Overview

First of all, we are going to represent some information regarding the resources used to search papers, in order to aid assessment of the work done. In Table 2.1, we discuss advantages and disadvantages of the sources.

2.3.2 Excluded Papers

Basically, there were thousands of paper found with our queries. Nevertheless, the time constraints didnt allow us to look up them all, so we opted to only look through first 100 papers in every search, throwing out everything that did not fit inclusion criteria, that was fortunately possible to do due to the nature of criteria, that are generally possible to determine by metadata. As the result, we have gathered 93 papers, that were reviewed one more time during quality assessment. After review, 22 more papers were discarded, leaving us with 71 acceptable papers. In Table 2.2, we present information on reasons the papers were excluded.

So, hereinafter we are only going to work with 71 remaining papers in this review.

Table 2.1: Sources advantages and disadvantages.

Source	Advantages	Disadvantages
IEEExplore Digital Library	Published papers can mostly be assumed to be peer-reviewed	Paywall
ResearchGate	Open access; well-formatted, full-color articles	Requires registration; problems with account confirmation
Google Scholar	Diverse sources, helpful in finding open access articles	Some sources can be unreliable, and some behind the paywall
ACM Digital Library	A good source of interesting and high-quality papers in IT, convenient means of search	Paywall
Arxiv.org	Open access, diverse papers	Very likely to be not peer-reviewed, sometimes outright poor quality, non-intuitive search

Table 2.2: Reasons for exclusion

Reason to exclude	Quantity	Percentage (of 93 papers)
Not a primary study	4	4.3
It wasn't peer-reviewed/Draft	5	5.3
Duplicate	4	4.3
Not in English	1	1
Not conference proceedings or journal articles	3	3.2
Other reasons	5	5.3

2.3.3 Studies Classification

In this section, we will first present some statistics on quality and content of studied papers, and then the overview of said content.

We have a Table 2.3 that organizes information by major topics we have. Note that these topics are not exclusive, as one paper may belong to several major topics, so percentage will not add up.

Table 2.3: Key topics distribution

Topic	Years	Number of papers	Percentage
Computer vision	2003-2017	35	49.3
Neural networks	2003-2017	11	15.5
Machine learning	2006-2017	16	22.5
Scene reconstruction	2009-2017	10	14
3D	2001-2016	8	11.3

Table 2.4 describes statistics by years of publication. For the sake of simplicity, we split all papers to five-year periods, with the last one being 2016 to present time, or a shorter one.

Table 2.4: Distribution by year

Years	Quantity	Percentage
2000-2005	7	9.9
2006-2010	16	22.5
2011-2015	28	39.4
2016-now	20	28.2

As we can see, number of relevant papers grows approximately twice every five years, and since we got almost as many papers for 2016 and 2017 as for previous five years, it is safe to conclude that interest in the topic is rising rapidly (Table 2.4). Furthermore, according to our observations, the overall quality of research is steadily improving over the years. Speaking of quality, in Table 2.5 we present statistics on quality assessment.

Table 2.5: Quality assessment statistics

QA score	Quantity	Percentage
0-2.5	34	47.9
3-3.5	13	18.3
4-6	24	33.8

Table 2.6: Quality averages by year

Years	Quantity	Quality
2000-2005	7	2.2
2006-2010	16	2.7
2011-2015	28	2.9
2016-now	20	3.9

The good news are latest research papers have significantly improved in quality, so QA scores are generally 4 or above (Table 2.6). Nevertheless, even papers with poor QA score can be useful in providing valuable information, even though we would not fully trust the results regarding efficiency of applied technologies.

2.4 Discussion

We have analyzed plenty of articles to determine which aspects and methods of computer vision and machine learning are used. In early 2000s, the following approaches were used:

- With use of Active Data Repository framework: vertex caching, approximate cube projection, density-based model fitting (results not well-reported) – human silhouette recognition [1]

- Principal component analysis + Bayes-based classifier (False alarm rate 3.21%, mistake rate 1.01%) – parking cell detection [2]
- With use of MATLAB: Object placement relation + K-NN (Accuracy 67.55%) – scene construction [3]
- With use of MATLAB: Fuzzy C-means classifier (Sensitivity up to 99.92%) – vehicle detection [4]

These approaches are based on a variety of mathematical techniques and concepts. Thus, K-NN is a method that uses vectors of features by computing distances between them, and then choosing K nearest objects according to this distance and computing a sort of central value to predict classification label, while fuzzy C-means classifier implies non-strict clustering, where each object is assigned to several clusters with certain probability, in a way that minimizes sum of squares of distances to each cluster center. Later on, with development of new techniques, these approaches gave the way to newer and better ones, such as the following:

- Sift + Gist + SVM (accuracy up to 80%) – threat detection [5]
- Sift + Extreme learning machine (Accuracy 86.05%) – general scene recognition [6]
- Global features extraction + spatial transformer + CNN (accuracy 82.10%) – general scene recognition [7]
- Regions of interest + SVM – Boosting hybrid (accuracy 87%) – obstacle recognition in traffic scenes [8]
- Principal component analysis + K-means clustering + Spatial Pyramid VLAD encoding + SVM (accuracy up to 96.15%) – traffic scene recognition [9]
- Deep convolutional neural networks (AUC up to 0.9997) – Parking lot vacancy indication [10]

Overall, it is easy to notice that accuracy has undergone improvement with time, thanks to new methods usage, as well as sensible combination of existing methods. Among all these approaches, the most popular were CNN, SVM, Naive Bayes and Random Forest, that have shown decent quality measures in solving the problems related to our field.

CNN (Convolutional Neural Network) is, by definition, a class of deep, feed-forward artificial neural networks. Having been applied successfully to visual imagery analysis, this kind of neural networks is notable in the sense that it is well-suited for work with diverse variations of image recognition. CNNs have the following advantages: they are easily parallelizable, as well as relatively robust with regard to rotation and translation. The disadvantages, however, include, for instance, the excessive amounts of network parameters, that are not understandable easily in practical sense, i.e. in fine-tuning to the each task and computational resources available. These various parameters can be any of the following: number of layers, convolutional kernel dimensionality, number of kernels for each layer, kernel shift step for layer processing, necessity of subdiscretization layers, extent of dimensionality reduction, dimensionality reduction function, neuron transmission function, existence and parameters of fully connected layer before the output, etc. All these parameters influence the results significantly but are chosen by researchers empirically. There are plenty of existing and well-tested CNN configurations, but we in fact lack recommendations to build a network for our task. CNNs were used for parking lot occupancy detection [10,11] with AUC varying from 0.8826 to 0.9999 and accuracy varying from 0.398 to 0.981; and also for general scene recognition [7] with accuracy between 0.5395 and 0.8210.

SVMs (Support Vector Machines) are again, by definition, supervised learning models for regression and classification, with associated learning algorithms. Their advantages are:

- it is guaranteed, that a globally optimal solution will always be found;
- there are plenty of implementations of this learning algorithm for different programming languages and packages;
- both hard-margin and soft-margin data are susceptible to the method;
- SVMs support semi-supervised learning.

The only real serious disadvantage of the method is relatively low suitability to natural language processing tasks, which do not relate to our case in any way. Also, SVMs cannot return probabilistic values, that makes the method in some sense non-intuitive for interpretation. SVMs were used for parking space detection [12] with accuracy between 0.9151 and 0.9984; for traffic scene detection [13] with F1-score between 0.2445 and 0.5415; for traffic obstacle recognition [8] with accuracy up to 0.87; for online vehicle detection [14] with recall around 0.86 and precision around 0.94.

Naive bayes is one of simple classification method, based on Bayes theorem. The key assumption of the method is that all incoming data are strongly independent. Good examples of advantages of the method would be training set size insensitivity, fast learning, and overfitting robustness. These are reasons why Naive Bayes is used where there are not much training data available, or else when there is too much data and the learning speed is critical. Disadvantages are rather poor quality of prediction as compared to alternative methods, especially when data are in fact dependent. Naive Bayes algorithm was used for parking space detection [2] with accuracy between 0.9781 and 0.9899.

Random forest is another classification and regression method. The main idea is randomly making a set of decision trees at a training time and using them all together for prediction. The method was introduced to overcome a strong tendency of decision trees to overfit, and it has, in fact, enough advantages to consider using it. For example, it is able to process data that has a lot of features and perform extensive multiclass classification. Then, it is not sensitive to any monotonous normalization of features. Furthermore, both discrete and continuous features are processed rather efficiently, along with ability of random forests to work with data with some missing features. There are also methods to estimate importance of different features within the model. Finally, random forests are easily scalable and parallelizable. However, there is also a considerable disadvantage, namely a large size of resulting models, linearly dependent on the number of trees. Random forest was used for parking lot detection [15] with average accuracy of 0.979; for image understanding [16] with accuracy between 0.392 and 0.8529.

Along with machine learning methods described above, some auxiliary methods were used. Semi-hard clustering and fuzzy C-means were used for parking space detection [4], to yield accuracy between 0.6206 and 0.9952. Extreme learning machine, proposed in [6], gave accuracy between 0.767 and 0.878. Haar-like features gave recall of 0.7 and precision of 0.65 for online vehicle detection, while HOG features allowed to achieve recall and precision both being 0.76 for the same task [14]. These methods were not used often, so we were not able to gather enough statistics to be sure in objectivity of information we have, especially considering the fact that in many cases we didnt have enough information on datasets to evaluate objectivity of the quality measures presented.

Methods change over the years, yet there is one certain rule that still holds: the more data we use to train learning models and the more representative it is, the better the results are in the end, which leads to an obvious conclusion that we will need really large amounts of data in order to train our system well.

So overall, a lot of work has been already done in related fields. For instance, in [10, 14] the car detection was implemented reasonably, but existence of, say, bicycles or pedestrians, was not taken into account. In [17, 18], on the other hand, researchers were implementing parking space detection, but did not take into account the fact that lighting can be lacking, as well as that vehicles can be fairly large. As a whole, a task of parking lot detection has been already solved efficiently, as well as person recognition in some implementations, but still nobody has yet implemented what we are proposing – namely, everything that they have, including parking space detection, vehicle detection, people detection, both at daytime and nighttime, along with event-based triggers that would allow to integrate similar sensor systems into IoT infrastructures. So we are going to combine all the work done in order to get a much more complex product than ever, with better accuracy and easy scalability and ease of amending the system with new functionality. Therefore, the research gap is that everything we need already exists as separate parts, but has not been combined yet.

2.5 Conclusion

In this SLR, we tried to find out what methods and techniques are the most suitable for the system we are going to implement. A thorough analysis has shown that apparently, CNN and SVM and other methods are quite popular for this group of tasks, that must be for a reason, since they seem to be the most suitable for the field, so we must conclude that it is justified their further use, along with certain preprocessing techniques such as principal component analysis or K-means clustering.

Considering datasets to use for training and testing, it would come in handy to collect our own custom dataset in addition to dozens of relevant open source datasets available online. As for measures of efficiency, accuracy seems to be the most appropriate since it is easily extendible to multiclass classification case and shows relevant statistics when datasets are not strongly skewed. The aim for efficiency should be set as high as 95% accuracy or higher, to beat state-of-the-art results.

Finally, these considerations are subject to further refinement, since a certain amount of small changes in methodology can basically lead to radical change in results. Nevertheless, the chosen methods form a good starting point we are going to use.

Bibliography

[1] E. Borovikov and A. Sussman, "A high performance multi-perspective vision studio," *Proceedings of the 17th annual international conference on Supercomputing - ICS '03*, p. 348, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=782814.782862

[2] H. Deng, D. Jiang, and Y. Wei, "Parking cell detection of multiple video features with PCA-and- bayes-based classifier," *Proceedings of IEEE ICIA 2006 - 2006 IEEE International Conference on Information Acquisition*, pp. 655–659, 2006.

[3] A. Subpa-asa, N. Futragoon, and P. Kanongchaiyos, "Adaptive 3-D scene construction from single image using extended object placement relation," *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry - VRCAI '09*, vol. 1, no. 212, p. 221, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1670252.1670299

[4] H. Ichihashi, A. Notsu, K. Honda, T. Katada, and M. Fujiyoshi, "Vacant parking space detector for outdoor parking lot by using surveillance camera and FCM classifier," *IEEE International Conference on Fuzzy Systems*, pp. 127–134, 2009.

[5] G. Madikenova, A. Galimuratova, and M. Lukac, "Threat detection in episodic images," *IDT 2016 - Proceedings of the International Conference on Information and Digital Technologies 2016*, pp. 180–185, 2016.

[6] L. Wu, Y. Yu, and J. Gu, "A Scene Recognition Method using Sparse Features with Layout-sensitive Pooling and Extreme Learning Machine," no. August, pp. 178–183, 2016.

[7] S. Guo, L. Liu, W. Wang, S. Lao, and L. Wang, "An Attention Model Based on Spatial Transformers for Scene Recognition," pp. 3757–3762, 2016.

[8] R. Mocan and L. Dios, "Multiclass classification based on clustering approaches for obstacle recognition in traffic scenes," pp. 1–5, 2016.

[9] F.-Y. Wu, S.-Y. Yan, J. S. Smith, and B.-L. Zhang, "Traffic scene recognition based on deep cnn and vlad spatial pyramids," 2017. [Online]. Available: http://arxiv.org/abs/1707.07411

[10] S. Valipour, M. Siam, E. Stroulia, and M. Jagersand, "Parking-stall vacancy indicator system, based on deep convolutional neural networks," *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016*, pp. 655–660, 2017. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015181563{&}doi=10.1109{%}2FWF-IoT.2016.7845408{&}partnerID=40{&}md5=f9bd5fcabd29c0897a93b2581014e103

[11] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, "Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning," *Symposium on Computers and Communication IEEE*, no. Dl, 2016.

[12] P. Almeida, L. S. Oliveira, E. Silva, A. Britto, and A. Koerich, "Parking space detection using textural descriptors," *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, no. August 2014, pp. 3603–3608, 2013.

[13] C. Y. Chen, W. Choi, and M. Chandraker, "Atomic scenes for scalable traffic scene recognition in monocular videos," *2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016*, 2016.

[14] D. Neumann, T. Langner, F. Ulbrich, D. Spitta, and D. Goehring, "Online Vehicle Detection using Haar-like , LBP and HOG Feature based Image Classifiers with Stereo Vision Preselection," *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, 2017.

[15] R. Dube, M. Hahn, M. Schutz, J. Dickmann, and D. Gingras, "Detection of parked vehicles from a radar based occupancy grid," *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, pp. 1415–1420, 2014.

[16] W. L. Hoo, T. K. Kim, Y. Pei, and C. S. Chan, "Enhanced random forest with image/patch-level learning for image understanding," *Proceedings - International Conference on Pattern Recognition*, pp. 3434–3439, 2014.

[17] N. H. Barnouti, M. A. S. Naser, and S. S. M. Al-Dabbagh, "Automatic Iraqi license plate recognition system using back propagation neural network (BPNN)," *2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT)*, no. March, pp. 105–110, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7976099/

[18] I. Masmoudi, A. Wali, A. Jamoussi, and A. M. Alimi, "Vision based System for Vacant Parking Lot Detection : VPLD," *IEEE International Conference on Computer Vision Theory and Applications (VISAPP), Vol. 2., 2014.*, no. January 2014, pp. 1–8, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7294974/

[19] P. R. De Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich, “PKLot-A robust dataset for parking lot classification,” *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937–4949, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2015.02.009

[20] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” *arXiv preprint arXiv:1612.08242*, 2016.

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results,” http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Chapter 3

Methodology

3.1 Motivation

When you consider the idea of this thesis, it is becoming clear that the system is going to have quite a complicated structure in order to efficiently do everything it is supposed to do, and technologies that are to be combined are all nontrivial in and of themselves. Machine learning and computer vision are both highly sophisticated fields, so it is crucial to determine the methods we are going to use and ensure a good understanding of theoretical background behind them.

The facts mentioned above become the incentive to consider thoroughly describing the methodology of our work in order to, first, ensure maximal reproducibility of the results, as our final intention is to enable the system to be widely used in industry and for public good with only cosmetic changes, and overall wide adoption. Another reason is that we need to give evidence persuasive enough that usage of the proposed system can be expanded to multitude of other use cases, and in the long run, serve as another step in development and usage of IoT.

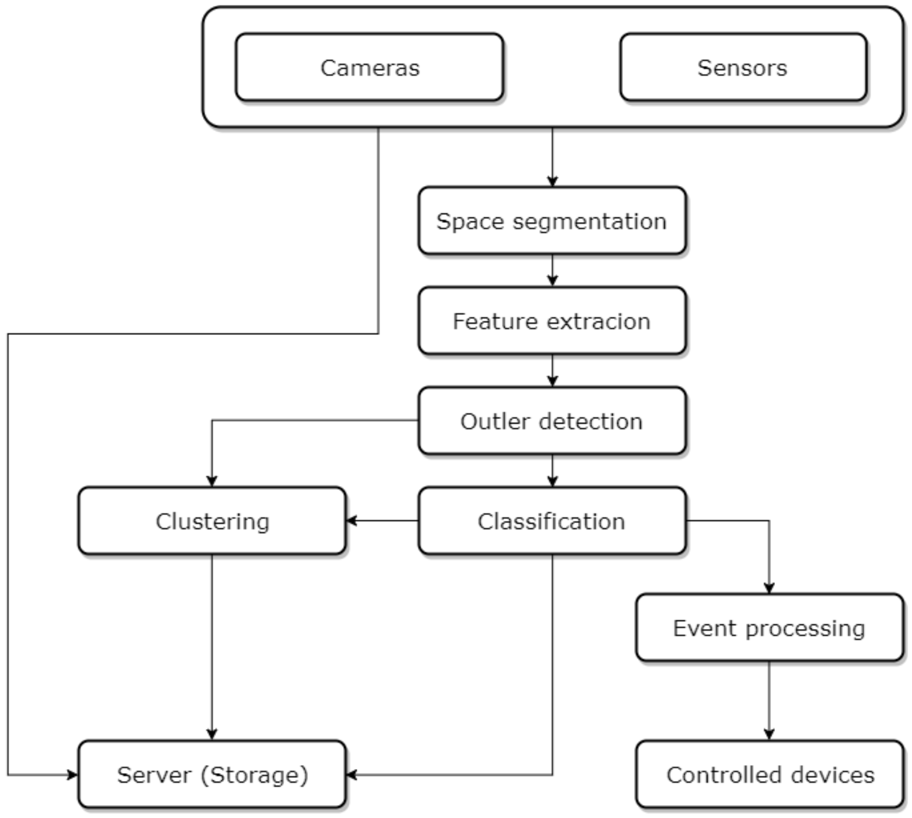
Thus, in this chapter we are going to consider:

- The overall high-level architecture of the proposed system
- Overall workflow plan
- Suitable computer vision methods for image segmentation and feature extraction
- Suitable machine learning models
- Possible datasets to use for training
- General strategies of performance evaluation

3.2 High-level architecture

Let's take a look at overall system architecture (Figure 3.1). By design, at first the information is collected from different sources, such as cameras or, for instance, infrared sensors. The number and kinds of these sensors does not really matter, and should not really. The collected data then goes in parallel to the storage and into the data processing stage.

Figure 3.1: High level architecture



What is happening in data processing stage, depends on the data, but in case of camera usage, like in our work, we are going to extract one to five frames each second, then perform segmentation on them. The segmentation of pictures from each camera is going to be determined manually on each deployment of the system. After that step, all the further work is going to be done on segments we obtain. To be specific, feature extraction is performed on each segment, with further normalization where it is needed. After this step, data is going to flow into the classification module, where a machine learning model gives out appropriate labels for each segment. After classification, if the segment was unrecognized, it proceeds to the clustering stage to be subsequently stored in structured fashion. In all other cases, labels are saved and sent to the event processing module, where triggers are going to be activated based on labels and locations of segments. For this task, any subscription-based system is suitable, along the lines of Apache Kafka. One of subscribers might be a visual module that outputs the video along with labels on a separate layer. Also, we are going to have GUI for segmentation tasks, for clustered segments lookup, and for overall configuration of the system. The described architecture was created with extensibility in mind.

3.3 Development process

Overall, the research process is most likely going to be performed the following way (Figure 3.2). First, we will need to find or collect datasets suitable for our task, as they are going to be needed not only for the sake of testing, but also in order to perform proper learning of components. After that, we are planning to implement and test most of described computer vision and machine learning algorithms in certain combinations, during the process of the search of an optimal combination of methods and optimal parameters for that combination. Only after that, when a more or less optimal solution is found, we are going to deploy the system as a whole, that is, with real-time streaming from cameras, settings, event processing module, unknown data clustering module and everything else that is planned regarding the architecture of the solution.

Although the workflow described above might seem linear, in fact this is not the case. The workflow is going to be cyclic, or iterative, where on each iteration a new module is deployed, and already existing ones are improved and optimized. So in the beginning, more focus will be placed onto computer vision and later machine learning, while in the end, it will be more about architecture and integration. Consult iteration plan below.

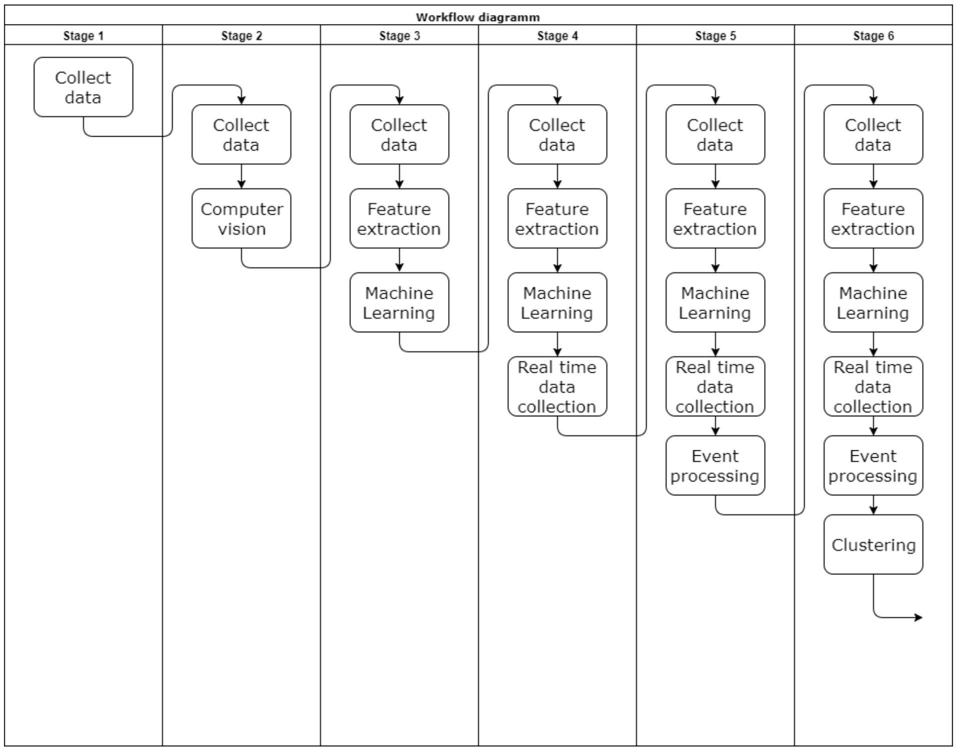
In next two sections, we are going to consider CV and ML methods we are going to use with detailed explanations of them and justification of the choices made.

3.4 Computer vision

3.4.1 A few notions on computer vision

In the field of computer vision, the questions we are facing are numerous, for instance, how to retrieve features from images and what methods exist for this task, how many features are enough and how it is justified. Another issue is that the quality of the picture is not going to stay the same throughout the cycle of system usage, as darkness, winter and bad weather conditions ensue. Thus, it is important to take into consideration results we are going to get in the sense of robustness and different kinds of invariance. Let us now get familiar with a couple of basic concepts.

Figure 3.2: Workflow diagramm



Computer vision is a specialized approach to creation of machines that aim to detect, track and classify objects on pictures or videos. They are able to do it based on obtained experience, based on machine learning, through extracting special points or places of interest, called features, from the image.

For images, features are abstractions of what is pictured on the image, and level of these abstraction can vary widely. They may be curves, points or probably characteristics of the points. Basically, they can represent any remotely interesting fragments of the image, while also throwing away redundant information. This is normally done on the low level.

As mentioned above, image features come in four different flavors, namely edges, corners (interest points), blobs (regions of interest points), and ridges.

Edges are usually one-dimensional, and represent the boundaries between objects or just image regions. They can generally have any shape and are defined as sets of points with high gradient magnitude. There are plenty of algorithms that determine edges with different constraints, such as smoothness or color difference.

Interest points, however, are points or small clusters of points, as implied from the name. They can be locally two-dimensional, and were historically called corners since many of them are located on intersections of edges.

Blobs are less point-like, and more region-like, compared with interest points. Recognizing blobs is, nevertheless, similar to recognition of interest points, as blobs often have central points that define them, and also, it is more a matter of scaling than anything, as interest points can also be more than single points.

The last kind, namely ridges, are quite specific and are more suitable for elongated objects. Ridges can be described as a weird kind of symmetry axes of these objects, or sort of medial axes, i.e. sets of points that are closest to more than one point on edges of an object.

So overall, all features extracted from images are more or less based on these four, or better say, three kinds of information (as interest points and blobs are very close conceptually and methodologically, so it is not clear whether it is even reasonable to treat them like two separate classes.

Note that from now on, we are going to use the term feature the way it used in machine learning, while in computer vision there are other terminology conventions, namely those that were used above. For instance, features in CV are edges, interest points, blobs and ridges, and when they are processed and a vector is formed as the result, it is called feature descriptor or feature vector, versus feature or feature vector/set in machine learning. From time to time, we will have to explain such subtle differences for the sake of better understanding of what we are doing here.

Features are blocks of numeric information that is, although at times looks as nonsense for humans, represents the most significant properties of an image (or whatever else) while also having significantly smaller dimensionality than the original image. This can be explained by the fact that usually images contain plenty of relatively redundant data that can be thrown away without much loss, that helps greatly considering the fact features allow us to drastically reduce image processing overhead along with noise and other undesirable effects.

There are two categories of features worth considering, namely local and global features. The distinction is made based on application. Local features are those that are used primarily for object identification or recognition [1], while global features perform better in the tasks of object classification, detection and image retrieval. Local features are used to describe key points of the image, while global features aim to generalize the entire object. You can easily say that local features describe the texture pattern of the image fragment, just like SIFT, SURF, LBP, BRISK, MSER, FREAK and many others do. As for global features, there are HOG, Co-HOG, Invariant Moments or Shape Matrices, that take context into account, unlike local features.

Combining local and global features improves the accuracy of prediction for the price of bigger overheads in computation. As has been written above, there are many different kinds of features, where every kind is extracted and subsequently used for tasks that stand quite far away from each other. Every task essentially needs its own package of features, that needs to be fine-tuned

[1]It is crucial to understand differences between detection and recognition: recognition means finding the identity of an object, and detection means finding the existence of an object.

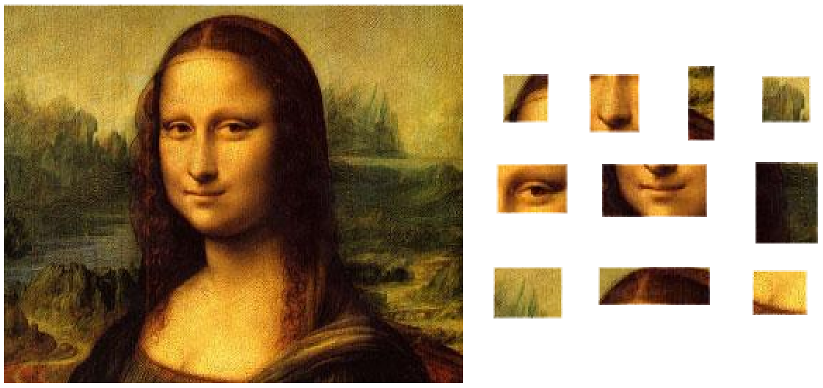
carefully, since accuracy of problem solving very much relies on the feature choice and subsequent data preprocessing.

Thus, plenty of methods have been created for the sake of feature extraction and accumulation, to satisfy needs for numerous and variable tasks. Some of them are going to be described below, along with their advantages and disadvantages as a whole and regarding the problem of this research.

3.4.2 CV methods analysis

One of the methods of feature extraction is Bag of Features, of BOF. While working with D-dimensional descriptors, it uses k-means clustering to group local descriptors into k categories, and then, using a histogram of distribution of feature vectors, produces a single k-dimensional vector, which is subsequently normalized. The choice for normalization may differ, but generally either the Manhattan distance or Euclidean distance are used. The next step is usually weighting by inverse document frequency. Multiple improvements for the scheme have been proposed, including abolishing k-means and using soft quantization techniques instead.

Figure 3.3: Example of BoF


(a) Before **(b)** after

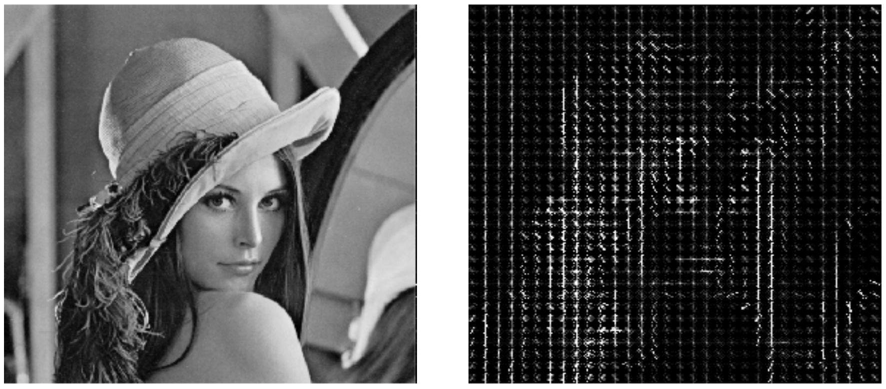
The nature of Bag of features is essentially Bag of words translated onto image feature extraction. In document classification, Bag of words is a sparse vector of number of occurrences of each word of a document set dictionary in a document, so it essentially represents frequencies of singled features in an unordered way. Even though bag of words is an efficient model that is widely used for feature extraction, it also has its own drawbacks. The most significant one is BoW ignoring spatial relationship between words, even though in image representation this information is incredibly meaningful. Several approaches to adding this information have been proposed. For different models, these approaches tend to vary. For discriminative models, a conventional method is spatial pyramid match that partitions an image into regions of decreasing size in order to compute histograms of local features inside each one. For generative models, the Constellation model is used. Its role is to capture spatial relationships between different parts of layers on which features are extracted, provided we can assume that the structure is hierarchical. For feature level improvements, correlograms were proposed to capture spatial correlations between different features. Moreover, the BoW model has not been properly and thoroughly tested with regard to performance and various kinds of invariance, nor has it been understood what exactly its role in object localization and segmentation is. At the same time, methods that considerably decrease codebook size while at the same time increasing the accuracy of classification in comparison to BoW have been proposed, such as Vector of Locally Aggregated Descriptors (VLAD) or Fisher Vector (FV), or other methods based on encoding of first and second order statistics. Furthermore, detailed comparisons of coding and pooling methods for BoW have been performed, with results disadvantaging Fisher Vectors in favor of combination of second order statistics, Sparse Coding and appropriate pooling methods that seem to be soon to start approaching the results that normally were only possible with use of simple convolutional neural networks. Overall, BoF is a method that has more advantages than disadvantages, even though it is not the most efficient one. For instance, it is especially efficient when it comes to classification of images according to objects pictured on them, while also outputting a constant length vector regardless of number of detections. It is robust to translation and rotation, but still the problems arise if you ever need to localize objects within the image, as it does not explicitly use configuration of positions of the features.

Another common feature descriptor is histogram of oriented gradients (HOG), used mainly for object detection. The method is to take small localized areas of an image and then to compute histograms of gradient orientation.

The idea behind HOG is that you can use edge directions or gradient distributions to describe local shape of an object efficiently. The image is divided into so-called cells, that are essentially small connected regions, and afterwards, the histograms of gradient directions are compiled in order to further concatenate these histograms to get the whole descriptor. After that, contrast normalization is often applied based on average color intensity across larger image blocks, so that descriptor becomes more invariant with respect to lighting conditions such as shadowing and illumination.

The HOG descriptor is basically a very strong feature, as it has a number of undisputable advantages that make it especially suitable for the task of human detection. For instance, it is invariant to photometric and geometric transformations due to relative locality of the feature, since this kind of changes only affects regions starting from certain size. Likewise, it was shown that with coarse spatial sampling, orientation sampling and strong local photometric normalization, individual movements can be safely ignored provided that overall orientation remains roughly the same. As was already said above, HOG is a decent feature representation for human detection and specifically, face detection, mainly by the means of binary classification. What is even better, it still works well for

Figure 3.4: Example of HOG features



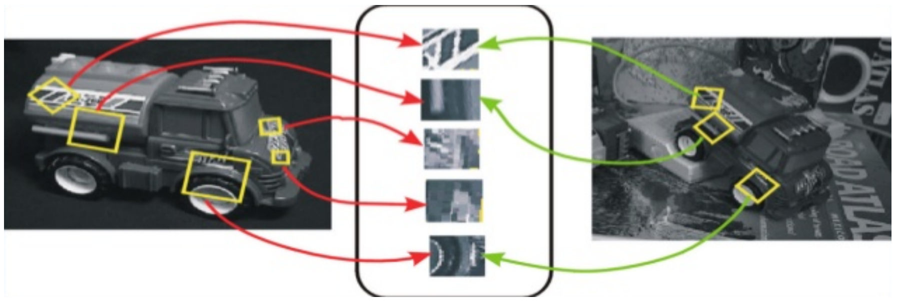
(a) Before **(b)** after

even for small resolutions, while also requiring very few additional actions in order to ensure efficient learning. Furthermore, if we want to do more localized and part-based prediction it is still possible, and allows to deal with more complex things such as occlusions, overlaps or moving body parts. However, that would require more complex reasoning, that implies more elaborate models to build.

More features to go, and now let us consider a classical algorithm of feature extraction called SIFT (scale-invariant feature transform) that is used in junction with local features. There are points on an image, that can be considered especially descriptive (such as points of interest), that will provide a feature description of an object after being extracted. This might help in recognition of a target object on a test image of many diverse objects, once a training image was fed to an algorithm to extract the needed features. For this, the features must be robust to illumination, noise and scaling, so that recognition remains reliable at all times. The most crucial factor for these features is unchangeability of relative positions between features comparative to one in the original scene. That means that only those points that will characterize an object at all times, regardless of the position, can be chosen. Same thing can be mentioned regarding flexible objects whose internal geometry can change from picture to picture. However, in practice SIFT uses many more features than a minimal necessary number, that allows for better stability of results by reduction of contribution of errors occurring because of presence of local variation. Overall, SIFT is quite a reliable descriptor, invariant even to affine distortion, not even mentioning illumination and orientation changes as well as uniform scaling, that makes SIFT able to perform under rough conditions like partial occlusion and clutter and still give robust performance results.

Nevertheless, SIFT is a rather old approach, that has become classical but is now overwhelmed with drawbacks. To start with, it would not work effectively

Figure 3.5: Example of SIFT



on low-powered devices, as it is quite demanding in terms of resources. This is easily explainable considering the fact that the method actually requires a lot of CPU time since we need gradients of each pixel to be computed in order to obtain gradient histograms and subsequently the result, not to mention that the algorithm is also mathematically complicated. Still, it is worth remembering that it was one of the earliest to be proposed, and that it remains being one of the most accurate and reliable ways to describe the pictures even today, regardless of scaling and rotation. Now, considering everything written above along with the fact that SIFT is patent-protected, we suppose that probably it is not the descriptor we ought to use.

Shape context is a feature descriptor proposed mainly on order to recognize objects. It is based on the idea that a good and discriminative descriptor, that will be compact, robust, and accurate, should take into account the distribution over relative positions. In order to achieve this, n points are picked on the contour of the shape, in order to construct n-1 vectors for each point by connecting it pairwise to all remaining points. The set of all this descriptions gives the possibility to measure shape similarity and find point correspondences, but still, it is way too detailed at the moment.

Now we need to ensure a number of invariances of a descriptor in order for it to be useful, namely small perturbations, scale and translation, as well as rotation in certain cases. This all is possible here, as it was empirically shown that shape contests are invariant to outliers, noise and deformations. As for scale invariance, it can be achieved by normalization of all vectors either by mean or by median norm, and lastly, translational invariance comes naturally, as only vectors are recorded, and not the origin.

Also shape contexts allow you to provide full rotational invariants in case it is necessary. It is not always necessary and in some cases even forbidden, when it stops you from recognition of different objects as different, since they come out to be actually similar with respect to rotation, so some local features, if not measured relative to the same frame. Still, if it is needed, you can make it a completely rotationally invariant descriptor by measuring angles of vectors at each point relative to the direction of the tangent to the contour on that point.

Now, let us take a look at the GIST descriptor, which was initially proposed in 2001. The idea is to have a descriptor that does not require segmentation, and which is in fact a low-dimensional representation. For this, five parameters are proposed to measure in complex ways to represent the dominant spatial structure of a scene, namely ruggedness, expansion, roughness, openness and naturalness, that are to be estimated with coarsely localized and spectral information, which is possible to do reliably.

The computation goes as follows. The image is first processed with 32 Gabon filters with 8 scales and 4 orientation, giving us 32 equally sized feature maps, each of them is then divided into a grid, dimensioned 4x4, in order to further average feature values inside each grid cell . All averages are then concatenated across every parameters, given a vector of 512 values for each parameter. Since this is actually a lot, different compression strategies are used.

Another kind of digital image features designed for object recognition is haar-like features. Historically, it was quite computationally expensive to calculate features based on image intensities, that is, RGB pixel values for every single pixel of the image, that led to consideration of so-called Haar wavelets. Haar-like features consider pairs of adjacent rectangular regions, which are assigned values based on sums of all pixel intensities, in order to further substract one region value from another. The regions are chosen at specific locations in an image, and the obtained difference is further used to categorize subsections of the image, as there are common almost-always-true observations regarding certain regions of the picture having a specific Haar pattern, and that might help to recognize an object provided a bounding box is given.

One good advantage of Haar-like features is speed, namely that due to specifics of the method, any Haar-like feature can be computed in constant time, regardless of its size.

3.4.3 CV methods conclusion

There is a notably big amount of methods for extracting features from images for different purposes, and in this chapter, we only covered a tiny fraction of them, since they were ones that were mentioned most frequently in articles related to our cause, so other features are probably much less relevant to us, while we only need something that is likely to work well in our case, and trying out all possible features would not be possible within a given timescale. Some information on methods robustness in respect do some kinds of transformations is presented in the Table 3.1. Furthermore, there is much more space for improvements than just choosing features already as it is, since different classifiers will need different feature sets with varying normalization methodology. Therefore, we are going to try described methods feature extraction, test them and find out which of them are the most suitable to classifiers of our choice, searching for a perfect methodical combination. We are not going to evaluate them separately, since features might have completely incompatible relative evaluation scores on different models, so the plan is to evaluate the system as a whole.

Table 3.1: CV methods summary table

Method	Scaling	Noise	Rotation & Translation
BOF	If scaling applied	Sensitive	Robust
HOG	Robust	More or less	Robust
SIFT	Robust	Robust	Robust
Shape context	Robust	Robust	Rotation – on demand, Translation – robust
Haar-like	Robust	Robust	Rotation – sensitive, Translation – robust

3.5 Machine learning

3.5.1 A few notions on machine learning

Machine learning is a class of artificial intelligence methods that are notable because of the fact that instead of straightforward problem solving, learning by observing solutions of multitude of similar problems is exploited. In other words, the algorithm learns ways to solve problems in practice, hands-on, just like a human would, only much faster, say, hours instead of years. After the learning stage has passed, the algorithm becomes able to solve similar tasks, based on the learning experience. This fact also saves a programmer from the need to actually know how to solve a given task, as they leave the task of finding dependencies and patterns to an algorithm, and the only things they need to do is properly prepare data and adjust the model, that saves the time spent coding.

There are numerous machine learning algorithms, which can be classified in a variety of ways, like by availability of data, type of predicted data, of principle of work of the algorithm. Say, there are clustering algorithms for unsupervised learning, when we dont have labels for datasets, or prediction algorithms for supervised or semi-supervised learning, where at least part of a dataset available is labeled properly. On the other hand, there is regression, that outputs numbers on a real scale, or classification, that assigns a label from a finite set of possible discrete labels. Then there is binary classification, where only two classes are considered, multiclass classification, where there are more than two classes, and multilabel classification, where each object can be assigned more than one label. Finally, algorithms can be based on linear equations (linear predictors), the concept of distance (metric predictors), trees (such as decision tree or boosting algorithms), or neurons (neural networks).

In this work, we are going to consider only linear models, due to their simplicity, speed and relatively good prediction quality, and neural networks, due to their ability to efficiently handle even really complex patterns, such as those that you normally have in computer vision tasks. Furthermore, in the course of the literature survey we completed, the absolute majority of models used in the relevant research were either linear (e.g. SVM) or neural network based (namely CNNs), that may have something to do with applicability of different machine learning models to computer vision. Also, we are only going to consider classification models, as we are going to sort the data we get into a finite number of classes.

Let us now consider differences between linear classifiers and neural network. We are going to start with general definitions and descriptions, and then move to their comparative analysis.

Linear classifier is essentially a classification algorithm based on calculation of linear hyperplanes to divide options. In case of binary classification, only one hyperplane is built, to divide the hyperspace into two parts. If there are more classes, the discriminating surface is piecewise linear. Overall, linear classification task is essentially a classification task, where the function 3.1 is minimized.

$$\arg\min_{w} R(w) + C \sum_{i=1}^{N} L(y_i, w^T x_i) \tag{3.1}$$

Where w is a vector of classifier parameters, R is a regularization function for these parameters, that is needed in order to prevent overfitting, L is a loss function that measures cumulative differences between predicted labels and actual labels, and C is the coefficient that exists in order to balance regularization component and loss function component, that is essentially the balance between model complexity and overfitting prevention.

Depending on different regularization functions and loss functions, we get different classifiers, such as hinge loss for linear support vector machine, or log loss for linear logistic regression. Also, implementations may vary depending on the method used for optimization, whether it is gradient descent or something else.

Finally, what is best about linear models is that most of them are easily generalizable to linearly non-separable cases, using a kernel trick. In many linear models, scalar products are used in some way. We can replace scalar product with another function with a number of specific properties, which is called kernel function, that implicitly performs feature transformation, and by the means of this, space transformation, so that in the transformed space, the boundary is linear. There are about half a dozen of functions commonly used for this trick.

Artificial neural network is a mathematical model, along with its implementation, inspired by biological neural network that we might find in living organisms. Internally, it is a system of simple processors (artificial neurons) that are connected to interact with each other. Such processors are usually quite simple, especially while compared to processors we might find in relatively modern computing devices. In such a network, every processor only works with signals it periodically receives and signals it sends to other processors. Nevertheless, when connected into a network big and sophisticated enough, these processors, although quite primitive one-by-one, can give impressive results at solving rather complicated problems. Neural networks are most often used in image recognition, clustering and discriminant function analysis.

Let us here note that in our research, we are going to use both clustering and classification on different stages of the system (Figure 3.1). Clustering is going to be used, first, in computer vision part, during feature extraction, and second, after classification, in order to sort unrecognized entities to ease the further manual problem solving.

By definition, classification is systematic distribution of researched objects, phenomena, or processes, by sort, kind, type, any significant features, for the sake of research convenience; grouping of primary concepts and a certain sort of ordering, based on an extent of similarity. Or, on the other hand, a set of objects ordered based on some principle, where objects have some classificational properties that are chosen in order to determine how similar or different they are.

Clustering, on the other hand, is designed to split a set of objects into monolithic groups that are called clusters or classes. The aim of clustering is to find already existing structures. It is a descriptive procedure that does not offer any statistical conclusions, but nevertheless, gives opportunities to perform primary analysis and learn more about data hierarchy.

So the main difference, in terms of machine learning, is that classification implies assigning existing class labels to objects, while in clustering we do not know any labels in advance and just try to split the set into a number of groups.

3.5.2 ML methods analysis

Let us now consider a variety of clustering methods, such as k-means, k-median and k-center, as well as the concept of fuzzy clustering.

In fact, k-means, k-median and k-center are all very similar and follow the same pattern, to an extent that it is safe to say that they are all the same algorithm, only with slightly different functions to optimize. In each of them, k starting points are chosen as cluster centers, then all other points are assigned to the cluster with the closest center. Then, centers are rearranged to minimize a certain function, that will be considered later. After that, a new check performed on all objects to reassign them to clusters they are closest to at the moment. The process is then continued until no object is assigned a different cluster compared to previous iteration.

Now considering the function that we need to minimize, it is the very difference between these three algorithms. For k-means, it is sum of squares of distances between a center and objects of a corresponding cluster, or L2 distance. For k-medians, it is sum of distances between a center and objects of a corresponding cluster, or L1 distance. And finally for k-center, it is the radius of every cluster, or distance between the center and the cluster member located furthest from the center.

Let us now take a look at fuzzy clustering. This is a kind of clustering where an object can belong to more than one cluster simultaneously, to an extent. That is, instead of cluster labels, objects are assigned coefficients for every cluster, such that sum of all coefficients is 1 for every object. For instance, fuzzy c-means is a fuzzy version of k-means.

What is good or bad about this method? Let us first start with disadvantages. The first issue is that euclidean distance measures or any similar metrics will most probably weight underlying factors not in accordance with their importance, so at the very least, it is useful to use weighted versions of these metrics. Then, you will need to go through significantly more iterations if you want to achieve a slightly better result, that is sometimes disappointing. Lastly, as with every clustering method described here, you need to specify the number of clusters in advance (that is inconvenient and will probably later make us abolish these lousy methods and find something that fits our needs better).

Despite everything mentioned above, certain advantages still hold. For example, sometimes the possibility to assign more than one cluster to an object is a very useful and desired property. Also for overlapped datasets this algorithm gives much better results as compared to k-means, that defines the area of application of the method and basically implies that probably we ought to use fuzzy clustering methods in our work (since we might have several unrecognized events in one frame).

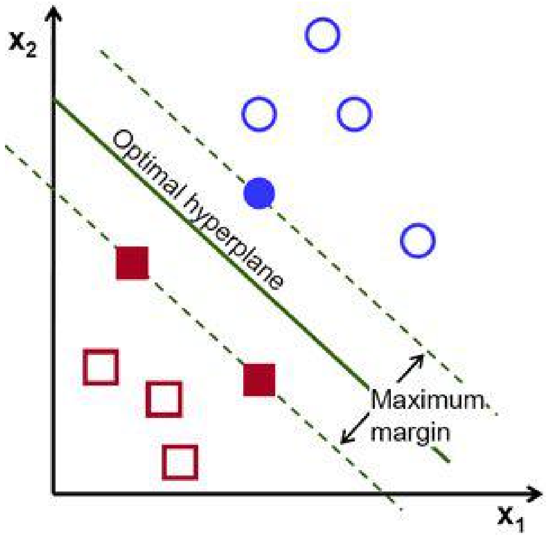
Now that we are finished with the overview of the clustering methods, let us have a look at several classification methods we are going to try out.

The first model for us to consider is Support Vector Machine (SVM). It is essentially a linear model with hinge loss and L2 regularization. The aim of this method is to find such a class border hyperplane that it is maximally far from both classes. Usually it is done by setting a condition that margin should always be no less than one, or alternatively, in case of nonseparable classes, less than one minus certain constant that is also optimized for every object. In addition to linear classification, SVM can also perform non-linearly with use of alternative kernel functions instead of classic scalar product, implicitly mapping input features to more dimensions.

SVM is pretty widely used everywhere where machine learning is in use at all, and for a number of good reasons. First and foremost, it is the fastest existing method to find discriminative functions. Also, the method is essentially based on solving a quadratic programming problem in a convex area that implies there is always the only solution. And what is best, the method performs the search for the widest possible band, that allows for straightforward and sure classification in the long run. On the other hand, there are certain drawbacks to consider, namely noise sensitivity, as well as strong dependence of results on data standardization and normalization. Furthermore, there is no general approach to choosing a kernel function in cases where classes cannot be linearly discriminated, so whenever you face such a situation, you need to manually choose a kernel and hope for the best.

Let us now consider another classification method, called logistic regression. It is especially suitable when classification is binary, and is a type of regression analysis. It is a predictive method, which means it outputs probabilities of labels, not labels themselves, that has a number of certain advantages over discriminant analysis. In fact, there are so many advantages that they deserve

Figure 3.6: Example of SVM



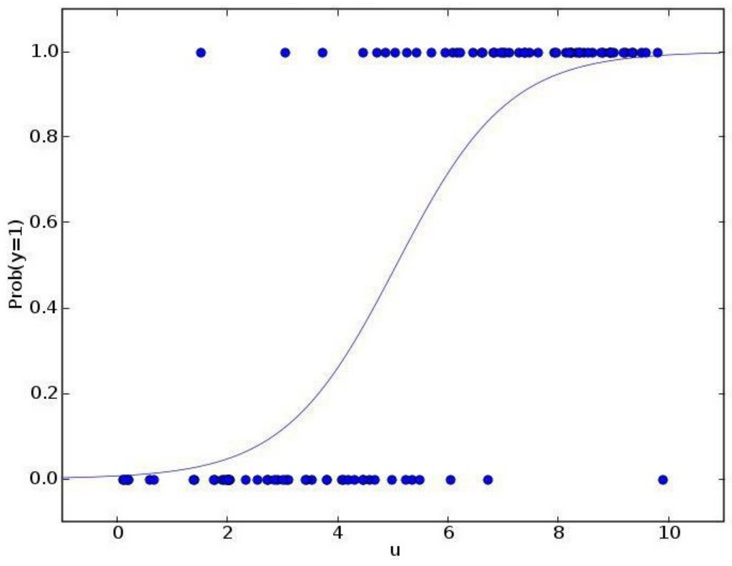
a separate paragraph.

To start with, feature values can freely be bounded or non-interval. Basically, no assumptions about features are done. As such, there is no assumption that error terms are normally distributed, or that variance is homogenous. Moreover, the outcomes do not need to be normally distributed. Furthermore, it is absolutely possible to add power terms or alternatively, explicit interaction, and, as it could have already become obvious, nonlinear effects are also handled quite simply. In fact, linear relationship between features and outcomes is not assumed at all, and as a result, the method is strongly robust in the sense that equal variance for each group is not needed, as well as normal distribution for features does not have to be assumed.

With so many advantages, it might seem surprising that any other methods are used at all, but still, the main drawback for logistic regression is that it needs way more training data than discriminative methods even just in order to get halfway stable, let alone marginally meaningful, results. The lowest amount of data in a training set ought to be several times higher, and that is the cost of such a flexible instrument to use.

Now let us proceed further to neural networks. For a long time, up until mid-nineties, fully connected neural networks basically ruled the world, and even today they are considered classic. Their main disadvantages are poor scaling, along with insufferably large number of parameters. This is why convolutional neural networks (CNN), which contain much less interconnections and share weights between neurons, finally came to life. They were proven incredibly successful in tasks of natural language processing and computer vision, giving completely record-breaking results.

Figure 3.7: Example of logistic regression



Considering specific properties of CNNs, the first thing you would want to mention is that they use such a variation of multilayer perceptrons that allows for as little processing as possible. Their structure was inspired by real biological patterns in visual cortex of animals, and the network is actually able to learn the filters that would otherwise have to be hard engineered. All said above means that relatively little dependence on human effort and prior knowledge is left intact, and thus CNNs are pretty much self-sufficient. They can also be used in recommender systems.

Although CNNs are impressively accurate when it comes to image recognition problems, this accuracy certainly comes at a cost, primarily literal high computational cost, down to the fact that without a GPU powerful enough, you will spend ages training a network, as they require massive amounts of training data.

So to sum up, advantages of CNNs are not many, but still quite impressive. For instance, the task of adapting a network to new problems is relatively easy, while at the same time it saves time spent feature-engineering, as it requires virtually no preprocessing, well, small amount of preprocessing. What is more, performance of CNNs is significantly higher than of other methods, especially in tasks that are generally considered hard to solve even with impressive machine learning machinery, and by a high margin.

To calm the fire of approaching instant success, let us reconsider disadvantages of the method. First, large amounts of data means really large amounts, as tens of thousands of sample will most likely be not enough. Second, CNNs are extremely computationally expensive when it comes to training, that would mean loads of powerful GPUs and yet still weeks until training is finished. Finally, since there is no strong theoretical foundation for convolutional networks, any fine-tuning is black art, be it a training method, hyperparameters, topology or anything else.

Now that we have a bit of general information, let us look at several popular and well-known models of convolutional neural networks. The first one is VGGNet, proposed in 2014, and utilized best with error rate of 7.3%. Due to convolutional and pooling layers, the spatial size of input is decreased on every layer, yet depth is increased, along with number of filter, in respect to downward direction in the network. Actually, after each pooling layer, the number of filters doubles. VGGNet is good both in classification and localization tasks, but training it takes months. Overall, the main principle was – keep it deep, keep it simple, which is about all you need to know in the beginning.

Unlike VGGNet, Region-based convolutional neural network (RCNN) is not even borderline suitable for our needs. It is hard to train, as it is a pipeline process, and needs terabytes of training data, but what is the most important drawback is the fact it is extremely slow, taking order-of-minute time to process a single image, that makes it completely unsuitable for the goal of real-time tracking.

3.5.3 ML methods conclusion

While analyzing different approaches (Table 3.2) to machine learning, we found out that there are numerous kind of machine learning models. However, we thoroughly describe only those two that we are going to test in terms of applicability to our task. The first kind is linear models, that are relatively fast to train, quite good in terms of accuracy and other quality metrics, and not strongly prone to overfitting. The second one is neural networks, that allow to drastically increase accuracy (and sometimes decrease it because of inappropriate usage), but at the price of slow learning speed, need for incredibly large amount of samples in a training set, and much stronger risks to overfit. Thus, neural networks are actually useful but require more skills and determination to handle.

3.6 Data preparation

Over the course of discussion, we defined the following labels for classification:

- People
- Car
- Empty parking lot
- Animal
- Garbage bin
- Motorcycle

Table 3.2: ML methods summary table

	SVM	Logistic regression	CNN: VGGNet	CNN: Faster RCNN
Model type	linear	linear	neural network	
Loss function	hinge loss	log loss	n/a	n/a
Sensitivity to outliers	moderate	moderate	NoDataYet	NoDataYet
Tendency to overfitting	moderate	moderate	strong	strong
Training speed	one of the fastest	fast	weeks or months	
Accuracy	good	good	depends on tuning	
Training data size	thousands samples		hundreds thousands samples	
Use cases	long numeric vectors		various	

- Big vehicle
- Unrecognized/everything else

The classification is going to be multilabel, as it is fully possible that there are going to be two or more kinds of objects in each image segment. All items labeled as unrecognized will be sent for further clustering, in order to ease human-driven monitoring.

3.7 General strategies of performance evaluation

todo

3.8 Conclusion

Chapter 4

Implementation

4.1 High-level architecture. Key points specification

As it was stated above, we used to have some really big plans for this project, because if not aim for great, why bother doing things at all? Here we are going to respecify the overall key points of work that is to be done in order for the project to be finished.

1. First of all, the high-level architecture of the system is needed, or, in other words, the abstract (yet relatively detailed notion of how the system is supposed to work at all, what parts it consists of, what they do, and how in the world they are supposed to interact, and with what results).

2. Since the final version is extremely likely to contain at least some machine learning methods in implementation, and chances are it is going to be a major part of the product, it is necessary to find a dataset that is good enough to properly train the model which is to be used in the project. The dataset should be big enough (and if it is too big, we can at least split it), as well as it has to be representative. The representativeness is to be manifested in realism of the dataset, that should include a range of situations we are expected to run into in real life, a good range of lighting and weather conditions, appropriate quality (that is, not perfect quality) and so on.

3. Also, when the model is trained and ready, it should be also possible to convert raw input data into data appropriate for the trained model, in terms of format, so that it is possible to further properly analyze these data, so that based in the obtained data it is possible to gain meaningful results that would be possible to process further and use in the next parts of the system.

4. All the listed steps are to be repeated multiple times, iteratively, so that the optimal combination of methods and algorithms is found, such that would fit best of all the needs of our project. Furthermore, the whole system is expected to be actually able to work in real time with several video stream sources, that means we are to aim for really fast-performing algorithms in order for the task to be done in sufficient quality.

4.2 The first try

For our first attempt, we decided we would simply construct the most primitive prototype that would determine if a slot is occupied by a car or empty with some elementary machine learning model.

Let us start with architecture. At first we did not really have any clear ideas, or for that matter, any remotely nice ideas at all, so the first actual thought was to divide the system into several major parts. One of the parts was supposed to directly work with the image. It is to connect to all available parking cameras, eject pictures from resulting video streams at some specified rate, manipulating those pictures in a way as to make the data suitable to a model used for prediction of what is going on in a picture[1].

Now, considering the predictive part, whis happened to be the machine learning part in all the cases, here we found several, namely 2, main approaches to data input. The first one is to put the whole picture into the ”predictor” and desperately hope for the best. Or the second one, the one we started with, is doing preprocessing first, extracting features, processing them if needed, and then use the ”predictor” on the preprocessed data.

Both approaches surely have their own flaws. For the first one, it is the fact that it requires more computational resources, and therefore, theoretically, more time to do the prediction, and especially training. The second approach, that uses preprocessing, on the other hand, requires much more efforts in coding, that you would have to debug and then possibly maintain, while it is not a sure fact that the quality will necessarily improve.

Another part of the system is going to be responsible for result processing in order to further save them in a human-readable format, such as logs, in real-time, and ensure that this format allows the data to be used by third-party modules for their own purposes. This way, we are going to be able to have logs of the parking state at any given moment of time, and if needed, create an overlay that would be put on the picture in real time so that the current situation is visually understandable.

Now that we have mentioned architecture a bit, let us move to the representative dataset search. When we were just starting working on this project, and were on the stage of reading literature, we thought that finding a dataset

[1]For all the further explanation, consider everything below to be said about a single picture due to conservation of detail, since the process of choosing frames from the video is going to be preserved to be the same in every concept we are going to describe. It has not been yet conceptually changed across the versions.

would be literally the last problem we could run into, as we were suffering trying to make sense of dozens of terms we were seeing for the first time while constantly wondering why people cannot write their papers in simple language and instead obfuscate information to such an extent that you would have to be a very patient immortal guru with infinite memory to decipher the resulting papers. Nevertheless, once we started considering which dataset we might need, it came out it was not such an easy task as we previously thought. In fact, it came out that finding an appropriate dataset for some exact task is pretty much overly difficult, and most of the time it still will be not what you need or not representative enough. We basically verified it in practice, by wasting some fortnight on extensive search for a dataset that would satisfy our needs and more or less failing.

Now you may become interested, what the possible difficulties can be, right? Well, first of all, it becomes more sophisticated task, the more specific the problem you are trying to solve is. As you could guess, our problem came out to be way more specific than it seemed. You could presume we would just need pictures of cars, preferably on parking lots, but in reality, we need something way more complex than that, namely pictures of parking lots with cars, people, motorcycles, and all other kinds of land vehicles. These special needs are the very reason why popular image datasets and databases out there in the internet do not generally fit our purposes, since they come out to be not representative at all for the needs of our project.

We put a lot of effort into finding a data set that would meet the needs of our project and on which we could train the model. We found such dataset [19].

This database contains 12,417 images (1280X720) captured from two different parking lots (parking1 and parking2) in sunny, cloudy and rainy days (Example on Figure 4.1). The first parking lot has two different capture angles (parking 1a and parking 1b).

The images are organised into three directories (parking1a, parking1b and parking2). Each directory contains three subdirectories for different weather conditions (cloudy, rainy and sunny). Inside of each subdirectory the images are organised by acquisition date.

Each image of the database has a XML file (Example on Figure 4.2) associated including the coordinates of all the parking spaces and its label (occupied/vacant).

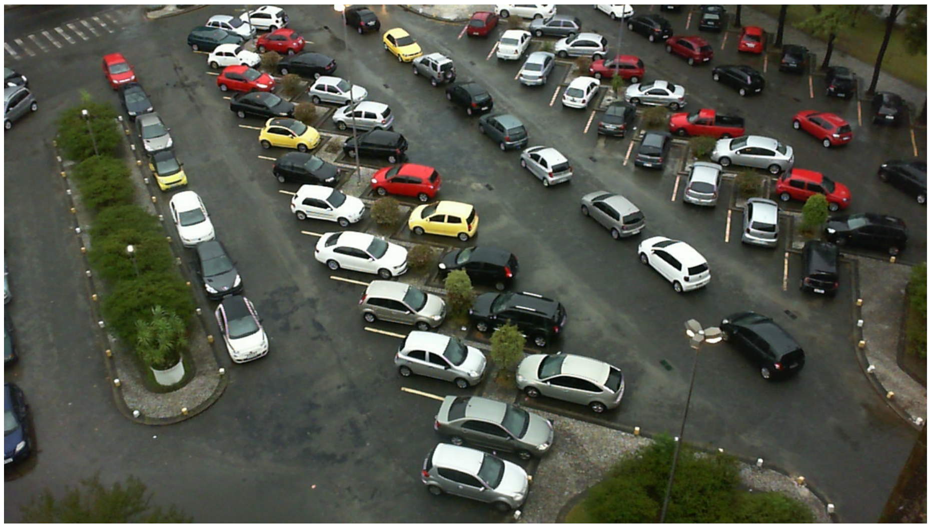
By using the XML files to segment the parking space, we are able to get around 695,900 images of parking spaces (Example on Figure 4.3).

So now, the task was - how to choose a subset to train the model on.

Its obvious that for some early prototype we dont really need to use the whole dataset for training, since all we have is a laptop which is not exactly powerful, so we decided to take a subset of this dataset just to try things out. The way we chose that subset? Well, at first we just chose one folder (corresponding to one parking lot, one weather type, one day) at random for a basic check, it contained about 5k images, and performed a check that everything works at all.

So consider we have at least some dataset now, so we can move further, and we decided to work on the predictive model and engage in some machine

Figure 4.1: Example of image from dataset



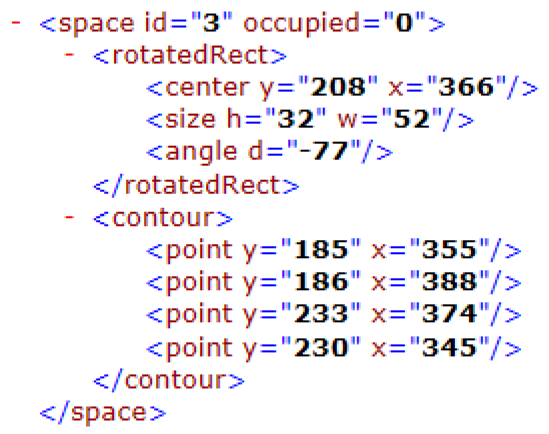
learning in order to do that. This would necessarily start to be the first thing to do immediately after enough data is found, due to that fact that machine learning is overall an incredibly resource-consuming area, while we do not really have much time, or any extraordinary computational capacities.

The first predictor we decided to use was SVM (support vector machine) making predictions based on HOG (histogram of oriented gradients). This combination was chosen for the first try due to the fact that SVM is one of the most frequently and successfully used techniques in smart parkings, besides artificial neural networks, and because HOG is representative of objects shape, which is the most important characteristic in the task of detection of the situation on parking lots. The overall principles of how this combination works is the following. HOG is calculated on picture fragments as a direction of a gradient from light to dark, thus outlining an object; it is a vector that has different length on different picture sizes, thus for usability, scaling all pictures to the same size would be required. SVM, on the other hand, will build two hyperplanes such that they are the farthest from class border, and in optimization, will only take into account elements with margin below 1.

Now we have dataset, chosen approach and we started tuning parameters with GridSearchCV (parameters like C parameter, which is responsible for trade off between error correction and regularization, i.e model complexity/simplicity, kernel, which determines shape of decision boundaries, gamma, coef0, parameters used inside kernel functions, tolerance, which determines when to stop optimization). As the result, we almost always got just the same results, well over 99% of accuracy (with roughly equal sizes of classes).

That made us think the dataset is not representative, and thus we need a larger dataset. We decided to add more images to our subset, and added them

Figure 4.2: Example of XML file from dataset (part)



```
- <space id="3" occupied="0">
  - <rotatedRect>
       <center y="208" x="366"/>
       <size h="32" w="52"/>
       <angle d="-77"/>
    </rotatedRect>
  - <contour>
       <point y="185" x="355"/>
       <point y="186" x="388"/>
       <point y="233" x="374"/>
       <point y="230" x="345"/>
    </contour>
  </space>
```

in such a way that in the end we had a dataset with both parking lots and all three types of weather, but other than that, the choice for folders with images was manual and random. As a result, the subset became 28990 pictures.

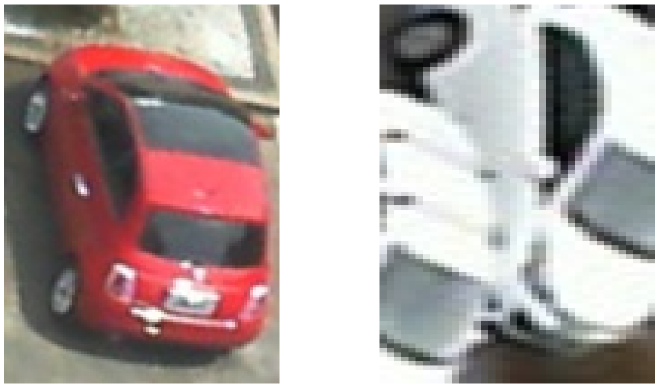
The way the machine learning code works? At first, all pictures of a subset are read from a directory we placed them into. Then, labels for these pictures are produced from filepaths (since files are sorted into free space and occupied space categories). Then HOG features are extracted from these segments, then the dataset is shuffled randomly, and then sent for training to SVM wrapped into GridSearchCV for parameter tuning. GridSearchCV uses 3-fold cross-validation for evaluation, that means that a given dataset Is randomly divided into 3 parts, then one of parts is used for validation and other two for training, where validation is done for each part, totaling 3 iterations.

The main problem with this thing was that it overfittes easily. We tried to fight it, but were not successful initially. The overfitting happened because we only took a small part of a dataset due to scarcity of computational resources, and the dataset was sorted.

We roughly prototyped the system structure in order to test the model in action, on pictures from the same dataset. As a result, after some manipulations we got a prediction regarding parking slots and their status in a human-readable format. The aforementioned format was chosen to be json, due to its easiness, popularity and relative universality. Also for the sake of convenience of debugging and testing we decided to draw some overlay above the picture, so that results are observable visually. So when we really tried to test it on a picture and look at it with our own eyes, we saw that actually this model wasnt nearly as good as it was supposed to be after the gridsearch was performed on it, so this made us think that the model overfits to that little chunk of data we used. So for a while, we decided to put this model off, and to try to employ other models for the task.

So as the result, we needed a working prototype and we got it. Well, it was

Figure 4.3: Example of segments from dataset



(a) Good example **(b)** Bad example

not that good, but at least it existed and worked. In fact, we got a working prototype with ridiculously low accuracy (¡number¿) and just as low speed. The next question is, why it was the case.

The ML model apparently did not show much accuracy since we basically trained it on so-called perfect data, small and neat. These were, to be more specific, pictures of slots that were right in the dataset we obtained, that were already cut, rotated, more or less scaled to similar (but not equal) sizes and made rectangular. Yet the prediction was basically on our own data, that is, slots we cut and preprocessed with our own tooling. These pictures were drastically different from the training set.

As for the speed, it is explainable at least with the fact that the whole field was analyzed every time, while SVM is not what we routinely call a fast predictor. Still, it was only the first prototype, so it is basically acceptable.

4.3 The second try

Some time after that we were thinking on the paradigm of processing and learning workflow and came to new conclusions. Previously, the idea was that we mark the space the camera captures into areas of interest (such as parking spaces) and then take each of such areas separately and classify with multi label classification (has cars or not, has people or not, and so on). The drawback of this approach is, for instance, that we wont be able to recognize when a car is not parked properly (such as into 2 parking spaces at once). So we had another idea, that probably we should take a picture, outline areas of interest, but then check a picture as a whole, and find all cars with their location (coordinates of bounding boxes), all people with their location, and so on. Then with bounding boxes check which areas of interest a given object falls into (that would be defined as object having more than a certain amount of bounding box falling into a given area. The amount could be, for instance, 40% of bounding box for incorrectly parked car and 80% for a correctly parked one) and assign the respective label to that area of interest. All areas of interest not containing objects of interest (or objects at all?) would then be labeled empty. For this approach, we had no idea how to use linear models and if feature extraction would help at all, so we decided to search for convolutional neural network based solutions.

That was how we started searching for neural networks out there in the internet. The thing we ran into and at first, liked, is called YOLO [20]. This it the neural network based system for object detection, written in C. Also there are short but clear guidelines on the website (what is good, they are really clear; what is bad, they are short, while we would prefer comprehensive documentation). In general, and on general datasets, it seemed to perform good, so we took a pretrained model, and tested it on parking data. Results were not so good, so we decided we need to take a more specific dataset and retrain the model.

For easy training of this model, two datasets are proposed on their website, namely VOC [21] and COCO. However, these are general purpose datasets, so we couldnt just simply take and use them. For simplicity, we took VOC 2012, and left there only images with vehicles and people, that are relevant. For that, we wrote several python scripts that deleted unneeded images and mentions of them in labeling and marking files. Overall, we got 4 categories and around 5.5k images, not much, but we needed something to test the thing. Then we modified for our needs and used another script, which generated marking data in correct format for a network. Then we launched the training, but in a couple of hours we got a failure, after ONLY 256 IMAGES having already gone into the network. We were disappointed. Well, it is quite obvious that we will need a better dataset for our purpose (like mentioned above), and most probably it will need to be manually labeled and marked.

4.4 The third (last) try

The architecture remained the same, as we think it is well suitable for our needs, and we also had the dataset of more or less acceptable quality, so we could finally put it into practice. This is where we started actually thinking on how to preprocess pictures so that we could only swap models, as it would not be convenient to rewrite the system every time we need a new model, due to time and effort constraints. After all, we are all people and get lazy at times almost all the time, so we could not possibly handle it.

In order to preprocess pictures it is required that every region of interest had its coordinates well-defined. In our case, regions of interest are parking slots, which are represented by quadrilaterals of arbitrary form on the picture. We actually got extremely lucky that XML markdown attached to the initial dataset contained these coordinates, and also that, due to specifics of our task, they are going to remain static, that means that the coordinates will also be static because it is quite obvious you absolutely would not move the location of a parking slot.

The coordinates then need to be parsed in a correct way so that the model is fed with appropriate data, not just some random stuff. In the first version we worked with coordinated parsed from XML, which was a more complicated option due to the need to parse an XML tree and retrieve coordinates from there which is not very convenient from the coders point of view, as it makes the code more sophisticated, as it is way more code in terms of lines count, which is to be further debugged, tested and possibly maintained, and also possibly makes the implementation slower, so we decided that in general we are going to store coordinates in JSON files instead, since it is a simple and reliable format that is easy to use.

Also that means that we would need our own tool for parking slot markdown, which we have actually made specifically for this purpose. Later on over the course of the paper we are going to describe this tool in detail, while now let us describe the process of cutting slots based on the coordinates we have. So the way it works is basically that there are coordinates of vertices of quadrilaterals bounding parking spaces, and a picture we need to process. First of all, we make the picture grayscale, as the color is not needed for our cause, and after that, we work with it black and white. Using these coordinates, the fragments are cut out of the source image, in order for them to further withstand a number of modifications, most notable of which are coordinate alignment and perspective transformation.

Now speaking about the reasons for these modifications. In short, we were thinking on an issue that we will have to extract fragments of a picture when making a prediction, since we will have an image of a whole parking as input, not separate slots. Then we found out that the data we cut looks quite different to slots we had in original dataset, and it started becoming a problem once we saw it drastically degraded the quality of prediction. Thus another idea came naturally what if we modify the script a bit, extract fragments from all the parking pics of an original dataset, and get a standardized dataset for training and validation, that would correctly reflect patterns in data we are going to work with.

The whole procedure of fragment transformation emerged due to the fact that from the camera point of view, parking slots dont look like rectangles, but rather like quadrilaterals, for example, trapeziums or parallelograms, but also completely arbitrary quadrilaterals, just like we mentioned above, and the fragments cut come out to be of different sizes and proportions, to worsen all things. This doesnt make a good material to feed a predictive model with, so a number of certain procedures is required.

For each parking lot, there is an xml file, where information on all parking spaces is stored (as we found out later, not all, but only part of them, but it does not really change anything). For each parking, four points are specified, that are coordinates of a bounding quadrilateral.

Of course it could be just a rectangle, and it would be way easier, because preprocessing would take us much less time to code, but the problem is that not all images of parking spaces are actually rectangles, there are also trapezoids and parallelograms, and arbitrary quadrilaterals, too. This should be taken into account anyway in order for the final solution to work better. Finally, coordinates in xml files we got with PKlot dataset we used in the baseline referred to arbitrary quadrilaterals, not rectangles, so we were basically forced to implement it this way from the beginning.

After coordinates extraction, coordinates processing is performed, which in our case means that we search for the longest side of a quadrilateral, to reorder points in such a way that this longest side is ultimately mapped to the left side of a subimage, with first point being in the bottom-left corner.

After that, we use perspective transformation to change the quadrilateral into the 40x60 rectangle. This is also useful due to the fact that all cameras come with perspective distortions. Mathematically, it is done via mapping vertices of the quadrilateral to a rectangle of certain shape and doing the respective affine transformations. The resulting rectangles are the final subimages, that are to be then fed to the model.

Now, let us talk about neural networks. We found a good Keras tutorial on fast and dirty convolutional neural network good for small training set sizes (and this is really true, since larger training sets made final results worse). So we have basically implemented several models using these instructions. Now, the last one of them works well enough to satisfy our requirements regarding prediction accuracy.

Overall, we got 5 models, similar in structure but trained on different data. Why so many models, and what does different data mean, you ask. The answer is, we did a certain amount of experimentation.

This was an outline of what our prototype does. But afterwards, we started trying to optimize its work.

While working on this version of our prototype and test-driving it, we came to an idea that we could seek for differences between the current picture and the previous one, and only check the slots this diff overlays onto. We decided to do it the following way: find IoU (Intersection over Union) of a changed area against a slot, and if the number is more than 0.1 (an arbitrary number subject to arbitrary changes, most likely to the less) then there are significant changes and we should use the predictive model to see what's going on there.

The question whether this is really an optimization remains open and might require benchmarking to be solved, but for now it looks like a good solution.

We even had several versions of this feature here.

The first version was searching for even the smallest differences. By smallest, we actually mean it. Overall, we decided to try finding difference between frames to further use it to analyze the situation, namely, if some changes are found, then find where they are, since chances are something happened there. However, this approach didnt lead us to success, due to its high sensitivity to basically everything. we decided to try finding difference between frames to further use it to analyze the situation, namely, if some changes are found, then find where they are, since chances are something happened there. However, this approach didnt lead us to success, due to its high sensitivity to basically everything.

The second try led us to significant improvements, as we added medianBlur, and after that GaussianBlur in order to make up for small fluctuations. This made differences actual differences, and not just random noise. After our solution started working, we notices that execution takes too long, that means that for now, our software won't be able to perform in real time. The problem seems to be that our algorithm for slot search performs a bruteforce search, being both linear in terms of slots and linear in terms of changes, or O(Slots*Changes), which can take quite a long time. Next we noticed that on the picture, there are multiple overlapping areas with changes, that could be merged so that there are less changes.

For the third and the last version, we merged overlapping areas, so their amount decreased. Results were similar, yet there is a concern that all these optimizations take in fact more time than they win in other parts of the system.

Now let us have a few words about our image labeling tool (ILT). The idea is extremely basic - we felt a need in some tool that would help one to markdown the slots, so that they could be marked whenever the system is being established somewhere or a new camera appears. In other words, in practice you would need to work with some random parking lot that is not yet labeled, especially if you are risky enough to actually employ our system in industry. This basically explains the need for this tool to exist, along with the fact that we found no tool that would be both allowing for arbitrary shapes and for convenient output formats. Then usual programmers thing rolled in, that if there is no tool you are comfortable enough to work with, you probably should code one.

The way the ILT works is quite straightforward. We are basically monitoring input for events like mouse clicks or keys pressed. The left click induces saving coordinates the cursor is on, that is, they are added to a special array on the single left mouse click, and then a point appears at the screen. Points then make line segments. Then, to finish drawing a shape, you need to double click. Also, if you are mistaken, the right button click is to the rescue as it basically cancels the last point. After this basic functionality, we added such features as the possibility to cancel previous shapes, save current shapes into a file, and open a file dump and continue work.

Chapter 5

Evaluation and Discussion

. . .

Chapter 6

Conclusion

...

Appendix A

Extra Stuff

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Appendix B

Even More Extra Stuff

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.