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

Following the establishment of the end goal of this work, we started researching the field to find out if there were attempts to solve the outlined problem, what issues they focused on, as well as the extent of success of every attempt that occurred. To keep the research thorough and reliable, we focused on tapping into exclusively academic sources.

Moreover, due to a literal explosion of amounts of available information in recent years, we had to find a way to limit our research so that we fit into deadlines of our thesis work, while also keeping the review academically valuable. This issue was the reason why we chose to perform a systematic literature review, which need not be comprehensive but also provides valuable statistical information that proved to be vital in our attempt to determine best approaches. While SLR is said to be way more effort-consuming to perform than a classical literature review, it also provides a handy framework for structurization of information, that allows for an extent of consistent control over the quality of information we are obtaining.

2.2 SLR protocol development

2.2.1 Research questions

In this literature review, we attempted to answer such research questions that they include maximum relevant information which we need to know to implement the system that suffices the requirements put on the proposed system for it to fit the cause that drove us towards this field. Here are those we pitched upon:

- **RQ1:** What technologies, tools and methods are suitable for implementing a smart parking system?
- **RQ2:** Which efficiency parameters were used to evaluate described systems?
- **RQ3:** What results were obtained in the course of evaluation?.

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. We mostly used the following sources and tools:

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

This list is far from exhaustive, as there were some papers which we have found over the course of the unstructured search. It is, furthermore, quite senseless to try to track each paper to some exact source, since even listed sources are often crosslinked or contain different versions of the same paper.

Search queries

For the search, we first needed to identify the key topics we needed to perform our research on. The first one was, apparently, smart parking. Also, due to us having to implement a standalone software system, software design was another one that came naturally. The other topics were trickier. Over the course of pre-search, we saw that all similar systems employ either sensors of cameras and since sensors are relatively expensive while cameras are everywhere, we decided to tap into camera-based systems. This decision would mean that another topic was computer vision. The next one was machine learning, due to being the most widespread group of methods used for the task accomplishment. This approach always requires plenty of training data to perform fine-tuning of predictive models; thus another topic came to be datasets and data collection. So overall, here is how the final list of key topics looks:

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

For each topic, key concepts were written down, then synonyms found. As a 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. We performed the search across titles as well as abstracts.

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 at least two of the five specified key topics
- 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 did not fit at least one of inclusion criteria were excluded from the review, as well as duplicates.

2.2.4 Quality assessment

We have developed a set of questions to assess the quality of included papers. 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?
    - 1 point if research questions are clear explicitly stated, or if the proposed system concept is described comprehensively
    - 0.5 points if there are some objectives of research that are related to the field
    - 0 points if no objectives are stated, or objectives are hard to determine
2. Does the research satisfy the objectives appropriately?
    - 1 point if research answers precisely the question that was stated or implements the concept that was proposed
    - 0.5 points if research is related to objectives but went somewhat off track
    - 0 points if research is unrelated to objectives
3. Is research process clear and reproducible?
    - 1 point if paper specifies clear steps, methods, technologies, and data, along with all necessary references and sources, needed to reimplement the research
    - 0.5 points if minor details are lacking, such as a dataset that is not readily obtainable, that can be inferred or replaced, or obtained via a few questions in an email
    - 0 points if it is impossible to restore the sequence of actions, or critical details missing, such as an algorithm or technologies used
4. Were the results assessed properly in a paper?

   - 1 point if provided results allow to fully reestablish the structure of the dataset in terms of its labels (i.e., results in a form equivalent to providing a full confusion matrix)
   - 0.5 points if published results can be used to somewhat evaluate the solution, but the full dataset structure cannot be restored
   - 0 points if provided results are not sufficient to give an independent evaluation of the solution

5. Were the results summarized to provide a clear conclusion?
   - 1 points if results were summarized in an appropriate and meaningful way
   - 0.5 points if results were over-summarized or under-summarized, or there was an apparently more meaningful way to summarize the data
   - 0 points if no attempt to summarize data was present, or if conclusion seems unrelated to the data

6. Are there any comparisons with alternatives?
   - 1 point if comparison with other solutions is performed, with advantages and disadvantages clearly stated
   - 0.5 points if only comparison with previous iterations of the same work was performed, or if the comparison was not comprehensive
   - 0 points if no comparison provided.

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

Apart from evaluation, we needed to retrieve some amount of meaningful information required to decide on our future track in this project, so for this reason, we have compiled a paper questionnaire, which then was used as a structured tool to retrieve the data by answering pre-specified questions. For the questionnaire, we have created the following questions:

- What is the dataset used in this study?
- In what ways was this dataset preprocessed before feeding it to the predictive model?
- What methods were used for prediction?
- What was predicted? (the range of output values for the predictor)
- What were the results of prediction?
- Is there any information on the processing speed?
- Is there information on any additional features of the system?

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, as well as to questionnaires, by both researchers. In case of disagreement, we discussed the application of criteria and agreed on conclusions. The synthesis was performed both qualitatively and quantitatively. Statistics about the papers were gathered, such as key topics, years of publication and quality assessment. Also, we classified the found information by methods of data processing, prediction, and evaluation.

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, to aid assessment of the work done. In Table 2.1, we discuss advantages and disadvantages of the sources.

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	Temporal delays of registration, no confidence in reliability of studies
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

2.3.2 Excluded Papers

There were thousands of paper found with our queries. Nevertheless, the time constraints did not allow us to look up them all, so we opted to only look through first 100 papers in every search. We threw out everything that did not fit inclusion criteria, that was, fortunately, possible to do due to the nature of criteria, which are generally possible to determine by metadata. As a result, we have gathered 111 papers, which were reviewed one more time during the quality assessment. After review, 25 more papers were discarded, leaving us with 86 acceptable papers. In Table 2.2, we present information on reasons the papers were excluded. So, hereinafter we are only going to work with 86 remaining papers in this review.

Table 2.2: Reasons for exclusion

Reason to exclude	Quantity	Percentage (of 111 papers)
Not a primary study	4	3.6
It wasn't peer-reviewed/Draft	5	4.5
Duplicate	5	4.5
Not in English	2	1.8
Not conference proceedings or journal articles	3	2.7
Other reasons	6	5.4
Total	25	22.5

2.3.3 Studies Classification

Now, let us first present some statistics on quality and content of studied papers, and then the overview of the 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	50	58.1
Neural networks	2014-2017 (one paper 2003)	21	24.4
Machine learning	2006-2017	48	55.8
Scene reconstruction	2009-2017	10	11.6
3D	2001-2016	8	9.3

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.

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.

The good news is latest research papers have significantly improved in quality, so QA scores are mostly 4 or above (Table 2.6). Nevertheless, even papers

Table 2.4: Distribution by year

Years	Quantity	Percentage
2000-2005	7 [1–7]	8.1
2006-2010	16 [8–23]	18.6
2011-2015	32 [24–55]	37.2
2016-now	31 [56–86]	36

Table 2.5: Quality assessment statistics

QA score	Quantity	Percentage
0-2.5	37	43
3-3.5	19	22.1
4-6	30	34.9

with poor QA score can be useful in providing valuable information, even though we would not fully trust the results regarding the efficiency of applied technologies.

Table 2.6: Quality averages by year

Years	Quantity	Average Quality
2000-2005	7	2.2
2006-2010	16	2.7
2011-2015	32	2.9
2016-now	31	3.9

Let us now look at the content of studies we gathered. In Table 2.7, there are some statistics regarding methods of feature extraction employed.

Table 2.7: Features extraction methods usage

Feature used	Number of papers	Percentage	Years
Background subtraction	9	10.5	2006-2016
SIFT	8	9.3	2015-2016
HOG	7	8.1	2012-2017
BoF	4	4.7	2015-2017
GIST	2	2.3	2015-2016
SURF	2	2.3	2012-2016
Other	18	20.9	2003-2017

As we can see, all these methods keep being used after their introduction, and overall, a wide variety of features are employed. The most widely used are background subtraction, SIFT, and HOG, and what is important, HOG keeps being used even in 2017. Also, though it cannot be seen from the table, it is worth noting that HOG was mostly used in cases similar to our problem.

The Table 2.8 is dedicated to machine learning methods and statistics of their usage so that it is clear which of them are the most widely used.

Table 2.8: Machine learning methods usage

Feature used	Number of papers	Percentage	Years
SVM	24	27.9	2007-2017
Neural network	21	24.4	2014-2017 (one from 2003)
k-(something)	13	15.1	2008-2017
FCM	9	10.5	2009-2016
Bayes-based classifier	5	5.8	2006-2017
Logistic regression	5	5.8	2014-2017
Other	10	11.6	2006-2017

Here it is easy to see that some methods are older than others, but most of them keep being used up to 2017. Fuzzy clustering, logistic regression, and Bayes-based classifiers do not seem to be very popular here, while SVM and neural networks are more widespread.

One could notice that information in the table dedicated to features does not add up to the total number of papers. This issue can be explained by the fact that not all methods require usage of features, so, for instance, papers dedicated to neural networks did not usually describe any feature extraction.

Speaking of methods for evaluation, the Table 2.9 displays statistics on efficiency measures used in papers. These are used to evaluate the quality of prediction in a meaningful way, thus allowing to compare different methods.

Table 2.9: Measures of efficiency usage

Evaluation measure	Number of papers	Percentage	Years
Accuracy	54	62.8	2005-2017
Recall and Precision	8	9.3	2012-2017
ROC curve	8	9.3	2010-2017
Confusion matrix	7	8.1	2013-2017
mAP	4	4.7	2015-2017
F1 score	2	2.3	2014-2016

Here one can even see, along with conventional measures, some more exotic, such as mAP (mean average precision). Also, it is evident that accuracy is overall prevailing.

Note that numbers here are not expected to adapt. The reason for that is that some papers did not contain proper evaluation, and other opted to use several methods at once to show different aspects of prediction.

Note also that there were papers related to work with sensors, not cameras, which were of general interest, but did not fit our purpose due to relatively high expenses on sensors and vast amounts of extra work required to mount them correctly. Nevertheless, a few of them employed machine learning as part of the approach and were quite amusing and beneficial for our cause.

Now with all the information we have gathered, and summaries we obtained and presented in this section, it seems feasible to infer more general and high-level facts from the information we have.

2.4 Discussion

In this section, let us discuss the overall picture we got while gathering information in existing studies. First of all, most of the papers did not look very trustworthy after quality evaluation. Two of the most common flaws were lack of reproducibility, which cast doubts at the validity of all the work, and underreporting of results, which also made checks on whether results are actually that good, problematic at best. While we understand that those experiments might well have been valid and thorough, and recognize efforts of fellow researchers, the way such studies are reported makes them only a barely reliable source for reference. Another problem seems to be systematic underreporting of negative results, which means that while good practices are well-known, the research community is forced to make all the same mistakes all the time, which apparently makes our work way harder.

Let us now answer questions from the paper questionnaire we designed, which were not answered as tables yet.

There were numerous datasets used in studies we explored, including the following:

- PETS
- PASCAL VOC [87]
- Microsoft COCO: Common Objects in Context (COCO) [74]
- VIVA
- Caltech 101 [88]
- ILSVRC [89]
- PKLot [54]

Some of these proved to be useful on some stages of our work later on.

Considering the prediction outputs, multiple cases are possible:

- The binary classification was among the most widely used, especially in tasks aimed to detect parking occupancy. In this case, values were EMPTY or OCCUPIED.
- Other studies used commonly exploited picture datasets, where the task was to classify an object into one of many categories (multiclass classification).

- There also were those aimed to detect an object, in which case, multiple labels could be assigned to a sample which contains multiple objects

- Finally, studies related to license plate detection had models that were to construct a value from the picture, and in this case, the task did not amount to a simple classification, and value range could have been quite broad.

Results of prediction, in their turn, also vary massively, with accuracy ranging from 62.5% in cloudy weather [15] to 99.4% [76]. The range of processing speed was even more widespread, being between 0.5 fps for VGG-based Fast-RCNN Ren2017 and 59 fps for the SSD system [56]. This range is, however, not exhaustive, as many papers did not contain any information on performance, so it only was reported in cases where results are scientifically valuable in the sense that they allow the scientific community to get closer to real-time object recognition. It is well-known that in fact, old neural networks could process a picture for up to 7 minutes.

Also, there are some papers worth noting for special features of the systems described. These include 3D-based parking space detection [35], unbelievably high prediction speed [56] and radar-based prediction [45].

As can be seen from the Table 2.4, this area becomes gradually more and more of interest for researchers, so we can interpolate that the interest is going to keep growing, attracting more people into the process of finding new, better solutions, especially since there are numerous problems in the field.

As for quality, from the Table 2.6 it is clear that there is a certain tendency in the fact that reporting quality improves with time, so one may hope that in the future it is going to be easier to rely on the background work.

The most popular methods of feature extraction seem to be background subtraction (supposedly due to simplicity), SIFT (which may be explained by amounts of information this feature vector preserves) and HOG (which may have something to do with the fact that it is best suited to represent shape). However, background subtraction is believed to be rather time-consuming due to the need to operate on a large number of pixels.

As for machine learning realm, the most widely used solution seems to be the application of artificial neural networks, especially CNNs, due to them being excellently suitable for work with pictures, without the need for actual feature extraction. SVM is almost just as widespread, probably due to the simplicity of the model, as well as the fact that this class of predictors aims for the optimal interclass border. Of methods for evaluation, accuracy beats all records, which is explainable by the fact that it is a simple measure that can be easily applied to multiclass classification.

From the statistics we gathered, it is easy to see that once a given method comes in use, it stays in use for most of the cases. The dates for early use are, however, nonrepresentative as they do not seem to correlate with dates those methods were invented.

2.5 Conclusion

As mentioned in the introductory part, our goal is to develop a system that is fast enough to work relatively real-time, undemanding in terms of computational resources in the process of use (and also in the process of training, as we do not have many computational resources), and reasonably accurate. The idea is to employ a simple solution not yet found in papers, and compare it to existing solutions in terms of both accuracy and performance for evaluation. Based on these goals, the research question for the rest of the work is the following:

RQ: How the proposed system compares to state-of-the-art works covering smart parking in terms of accuracy and performance?

Chapter 3

Methodology

3.1 Motivation

When one considers the idea of this thesis, it is becoming clear that the system is going to have quite a complicated structure in order to do everything it is supposed to do efficiently, 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 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 the public good with only cosmetic changes and overall broad adoption. Another reason is that we need to give evidence persuasive enough that usage of the proposed system can be expanded to the multitude of other use cases, and in the long run, serve as another step in development and usage of IoT.

Here is a brief plan of what we are going to cover in this chapter, for readers convenience:

- General strategies for data collection
- Discussion of issues of data preparation
- Suitable computer vision methods for image segmentation and feature extraction
- Suitable machine learning models
- Overview of software development techniques

3.2 Data collection

3.2.1 Rationale for data collection

One cornerstone of the development of systems that involve machine learning is data collection, since it includes training, and training needs data. Still, just having terabytes of data will not necessarily save an aspiring researcher, since one will also need to have it structured and be able to make sense of it to make predictive models work.

One of the most significant issues with data is that all datasets are flawed, that makes data preparation a crucial step of the process. Over the course of data preparation, one should make data more suitable for machine learning, and in broader terms, data collection is also a part of data preparation process, and often the most time-consuming step, where one might spend months on preparing the data before building the first algorithm.

There are three main variants of where to obtain data:

- Downloading an existing dataset, or combine several downloaded datasets.
- Performing field data collection with subsequent processing
- Combining the first two variants

3.2.2 Existing data sets

The first approach is quite straightforward. There are plenty of datasets out there on the internet — for pictures there are PASCAL VOC [87], COCO [74], and many others — and once one has downloaded one, they have a general purpose dataset, which is more than enough if they only want to compete in accuracy measures.

This method is quite quick and easy, as there is no need to spend time and resources to collect data from real life. At most, some reformatting is required, but the data is there, and it is already labeled. On the other hand, there are multiple drawbacks. There might be no data for the given study, or the data might be non-representative, some important information might be lacking, or there can be plenty of outright unnecessary samples. In the end, one might end up spending not less effort solving these issue than they would with manually collecting the data.

3.2.3 Collecting data sets

So this is basically why the second approach exists. A general purpose dataset might not always fit the specific purpose, and often there is just no dataset that would be sufficient. This issue makes a case for manual data collection, and this is, to put it nicely, not the most straightforward task. Big companies frequently hire some people specifically for that matter.

In other words, it is a long, expensive and resource-consuming process. On the other hand, it yields dataset which, if assembled right, will ideally fit the purpose.

So what is needed to collect a decent dataset, for it to be representative enough?

1. First of all, it is crucial to determine the goal of data collection. The more thoroughly the problem to solve is formulated, the easier it will be to determine and lead data collection. There are plenty of possible goal options, and all of them are good if the goal is well-formulated:

   - *Classification.* One needs to answer either a binary yes-no question (empty/occupied, ill/healthy, red/blue), or a question that has a finite predetermined number of possible answers, or categories (5 breeds of cats, ten genres of music, and so on). The right answers will need to be provided as labels.
   - *Clustering.* One needs to classify data into some classes they do not yet know, with rules that are currently unknown. The main difference from the previous variant is that one does not know in advance principles of division into groups. This situation might occur, for example, while trying to group customers by their behavior.
   - *Regression.* One needs some numeric value to be yielded. Examples can be the estimation of reasonable price, or of the number of items to prepare for a given day.
   - *Ranking.* One needs to rank objects using features. This approach is used in systems that recommend movies, music, or goods to a user.

   Most likely the problem falls into one of these categories, but even in other cases, the key is to avoid overcomplication and state a simple goal.

2. Establish data collection mechanisms, or rules on how to collect, store and process data.

3. Collect data based on predetermined rules.

4. After the data collection, formatting is required to make it consistent. The formatting here is about format consistency of data recording. It is especially true for data gathered from multiple sources by many people that it is essential to ensure consistency of date formats, money formats and so on. Another aspect of data consistency is all values conforming to the set constraints.

5. Reduce data to what is necessary. While it can be tempting to include as much data as possible, common sense should guide a data scientist into reducing data to amounts that are sufficient for the task but at the same time not overwhelming.

- One of the approaches is attribute sampling. One can determine which values in a record are critical, and which do not contribute much, thus reducing the size of the record by throwing out unnecessary values. Note however that domain expertise plays a prominent role here since one might need to be an expert in the domain the task is related to, to determine whether a given value is meaningful or not.
- Another approach is called record sampling. Mostly, this means removing records with missing, erroneous or less representative values, so that prediction becomes more accurate. One might also need to randomly shuffle the dataset and retrieve only a fraction of it if there is too much data.
- Aggregation of values might also help to reduce the number of records. For instance, instead of using daily values, one can add or average them over weeks or months and thus drastically reduce the amount of data to work with.

6. Clean data. At this stage, it is time to look at remaining missing values and do something to them. Concerning machine learning, it is better to have assumed or approximated values instead of just missing ones. The right way to fill the gaps, however, is heavily dependent on the situation:
   - Substitute missing values with dummy values such as 0 for numeric data, or other default values.
   - For numeric data, substitute missing data with dataset mean or median values.
   - For categorical values, use the most frequent categories.
   - Predict missing values based on other features.
7. Decompose data. In some way the opposite to data reduction, this step means that some values are too complex and decomposing them might capture relationships better. An example can be the retrieval of a day of the week from the date.
8. Rescale data. Often there is a need to rescale numeric values in a dataset, so that they do not overweight other features, or so that features are weighted in a right way. Several ways to rescale are the following:
   - Normalize to zero mean, unit variance (so-called standard scaling)
   - Normalize to a range (min–max normalization), often 0 to 1 (0–1 normalization)
   - Decimal scaling, or multiplying to some power of 10
   - Normalize the whole sample to the unit norm
   - Logarithmic normalization, used for data with skewed distributions

9. Discretize data. Some numeric data can work more efficiently once grouped into categories instead. Say, there is not much difference in heights of 167 cm and 169 cm, so these can be grouped. Another good example is binarization of the image, where all pixels are either made black or white.

3.2.4 Combine methods

Finally, the third approach goes as follows: several pre-collected datasets are taken to retrieve the necessary information, the manually collected data is added, and all the data is processed so that the format is uniform. It is sometimes faster than the fully manual data collection, provided that sources of data be known. If they are unknown, on the other hand, there is a risk that the process will consume just as much time as manual data collection.

3.2.5 Data generation

One more approach to finding a dataset is data generation, which can be automated or semi-automated, random, with given distribution, or conforming to specific rules. Bright sides of this method are that one does not need where to find data and only needs to know some rules that hold. Also, it is often easier and cheaper than field data collection so might be a reasonable option in some cases, especially if the suitable dataset does not exist. However, there are some areas in which data generation is quite tricky, like computer vision where data generation often boils down to 3D modeling. Many kinds of data, such as pictures, music, or texts, can not yet be generated in acceptable quality. Furthermore, this approach can quickly lead to dataset being not representative or too artificial.

3.3 Data preparation

3.3.1 Rationale

As mentioned above, it is vital to prepare data properly before they are feed to a model. The reason for that is this way we can decrease the training time, as well as resources needed for the process, or even increase the accuracy.

3.3.2 Part of picture vs. whole picture

When working with a picture, there can be two fundamental approaches to processing it.

The first one is working with the picture as a whole. Here we might have an incredible area for creativity, with all approaches and regions of interest. The best thing about this way is the possibility to work with context, taking into account not only an object, but also its surroundings, that might help us to detect vehicles parked inappropriately, or, occasionally, elephants randomly walking in the parking. On the other hand, this is a way of increased complexity, which might affect performance. Also, this implies that more time is going to be spent on training.

The second approach is cutting the picture into areas of interest and then working on each one separately. This method allows to concentrate on every separate area, yet there can be problems if regions of interest are identified incorrectly, this way missing an important object or counting it twice. This approach is easier than the previous one, both in terms of preprocessing and training. Also, it is suggested that performance is going to increase. The drawbacks are reduced ability to take context into account and dependence on the correct choice of regions of interest.

3.3.3 Raw pictures vs. feature vectors

Now, considering the predictive part, which 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. Alternatively, 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 flaws. For the first one, it is the fact that it requires more computational resources, and therefore, theoretically, more time to make the prediction, and especially training. The second approach, which uses preprocessing, on the other hand, requires much more efforts in coding, that one would have to debug and then possibly maintain, while it is not a sure fact that the quality will necessarily improve.

3.3.4 Picture properties: color, size, format

Even if we decide to use raw pictures as predictive model input, it will still require some transformations we need to perform.

The color, for example, is mostly not necessary for our case, and moreover might hurt the prediction by making the model distinguish between, say, a red car and a blue car. If we only want to capture the form, the grayscale image is perfect.

Scaling also matters a lot. Predictive models usually expect all objects to have the same number of feature, which calls for rescaling all inputs to the same dimensions. This issue applies both to the case when the whole picture is fed to the model, and to the fragment case. With fragments, it is even more complicated, since they are not necessarily even rectangular. Specific transformations (like perspective transformation) are needed to make them all fit the requirements.

3.4 Computer vision

Computer vision is a specialized approach to the creation of machines that aim to detect, track and classify objects on pictures or videos to understand the picture. Computer vision technologies can be represented as a mix of the following:

- Image processing methods, based on common ideas of possible transformation of imagery, as well as on complex mathematical models
- Pattern recognition, mostly with the use of experience-based methods like machine learning
- Attempts to represent the semantics of imagery, with approaches listed above, combined with the latest research in cognitive psychology.

Overall, computer vision technologies seek to approach (and in the future, beat) the human ability to recognize visual patterns, thus being a part of artificial intelligence research. Typical tasks in this area include:

- Detection, segmentation, localization, and recognition of specific objects in images, along with the evaluation of results
- Dynamic tasks such as tracking a moving object or recognizing the different views of the same setting
- Reconstruction of 3D space and objects based on images
- Content-based image retrieval

Many tasks in computer vision are accomplished via operations of so-called regions of interest (ROI), which contain features. For images, features are abstractions of what is pictured on the image, and level of these abstractions can vary widely. They may be curves, points or probably characteristics of the points. They can represent any remotely interesting fragments of the image, while also throwing away redundant information. This process is typically done on the low level. Examples of such features are edges, interest points, blobs, and ridges.

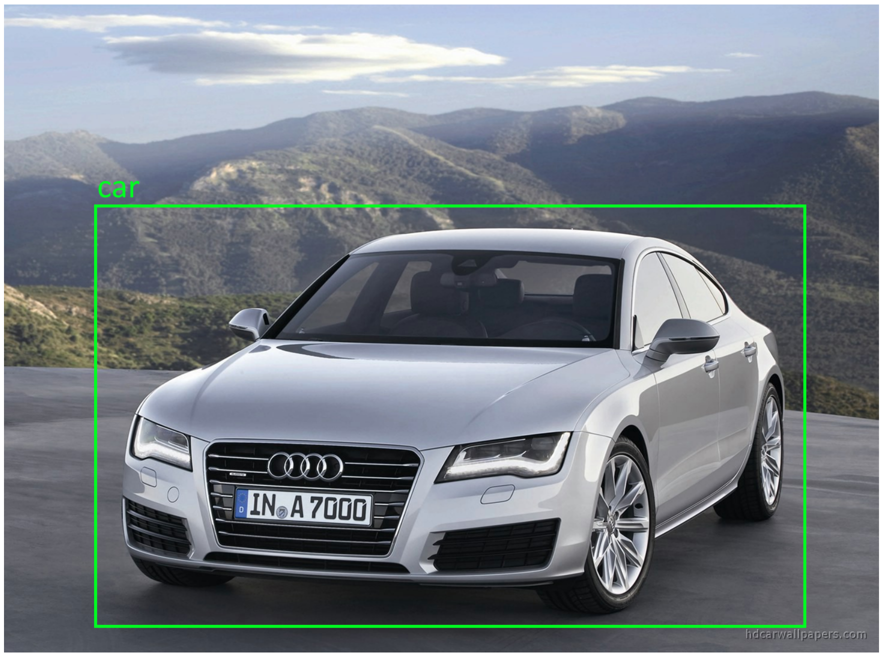
These features are customarily processed through conversion into some numeric representation, namely into vectors of numbers called feature descriptors and feature vectors.

3.4.1 A few notions on computer vision

Let us now look at specific important notions used in computer vision.

Bounding boxes (Ex. on Fig 3.1) are boxes of a predetermined shape (most often these are rectangles, though circles sometimes also occur) that fully enclose objects that are being checked for collision, and ideally are minimal of all that hold this property. They might be both 2D and 3D and often are denoted by coordinates of two corners.

Figure 3.1: Example of Bounding box



A **contour** is merely the boundary of an object in an image. Various representations of contours (e.g., chain code, Fourier descriptors, shape context) are used to recognize or categorize objects. This notion assumes that one has a way to segment out an object and find its boundary, which itself is not a trivial problem.

Let us now look at image features in more detail. As mentioned above, primary kinds of features in computer vision are **edges**, **interest points**, **blobs**, and **ridges**.

Edges are usually one-dimensional and represent the boundaries between objects or just image regions. They can 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.

Edges are fundamental in determining the shape of an object and thus are used incredibly widely. However, edge detection is by itself a non-trivial and complicated task.

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.

Interest points detectors are in a sense universal, as they overlap in applications with all the other kinds of feature detectors. Nevertheless, interest points are almost never used alone, due to them being not expressive enough for the most of tasks.

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.

Blobs are needed to provide auxiliary information which can not be obtained through edges or interest points, and, for example, to locate possible ROIs. A typical application of this kind of features is texture detection and analysis.

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.

Ridges are useful in the tasks of detection or retrieval of elongated objects, as well as image segmentation. Their typical usage is road detection in aerial images, as well as, for example, blood vessel detection.

So overall, all the information extracted from images is more or less based on these four kinds of the image feature, which form a basis for computer vision feature detection.

3.4.2 The structural similarity

The structural similarity (SSIM) index is a method used to measure the similarity between two images, created for prediction of perceived quality of images on television. It is a full reference metric, or to be clear, the measurement was initially based on the initial full-quality image as a reference. SSIM became a big step from peak signal-to-noise ratio (PSNR) and mean square error (MSE). The index is calculated on square windows of an image, that can be multiple. The measure for windows X and Y of the same size is described in 3.1:

$$\mathrm{SSIM}(x,y) = \frac{(2\,\mu_x\,\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.1}$$

The SSIM index satisfies the condition of symmetry: SSIM(x,y) = SSIM(y,x).

The main difference of SSIM from other techniques is that it is a perception-based model which both considers image degradation as a perceived change in structural information and incorporates important perceptual phenomena, such as contrast masking and luminance masking terms, while other approaches only measure absolute errors. The method is based on the idea that the more spatially close the pixels are, the more inter-dependency exists between them. These dependencies encapsulate valuable information about objects and their structure on the visual scene. Contrast masking is a phenomenon where significant activity or texture in the image makes distortions less visible, while luminance masking is a phenomenon where it is brighter areas that help to hide those distortions better.

3.4.3 Feature extraction

Previously, we used the notion of a feature in a way it is used in computer vision. Now let us consider what exactly we are going to do with these features — and the answer is, we are going to extract feature descriptors from the picture, to have a compact numeric representation at hand. All the future work is going to happen to feature descriptors, not just feature. Because of that, from now on we decided to switch to machine learning notion of features, which is numeric vectors used to describe some properties of entry for prediction, that is, exactly what was previously called feature descriptors. Also, features and feature vectors are sometimes going to be used interchangeably.

Features are blocks of numerical information that, 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 fact can be explained by the fact that usually, images contain plenty of relatively redundant data that can be thrown away without much loss, which helps considerably considering the fact features allow us drastically reduce image processing overhead along with noise and other undesirable effects.

3.4.4 Local vs. global features

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, while global features perform better in the tasks of object classification, detection and image retrieval [1].

Local features are used to describe critical points of the image, while global features aim to generalize the entire object. One 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 like HOG, Co-HOG, Invariant Moments or Shape Matrices, on the one hand, or, on the other hand, texture features, shape descriptors and contour representation, they are used more to recognize shapes and objects. Combining local and global features improves the accuracy of prediction for the price of more significant 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 package of features, that needs to be fine-tuned carefully since the 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

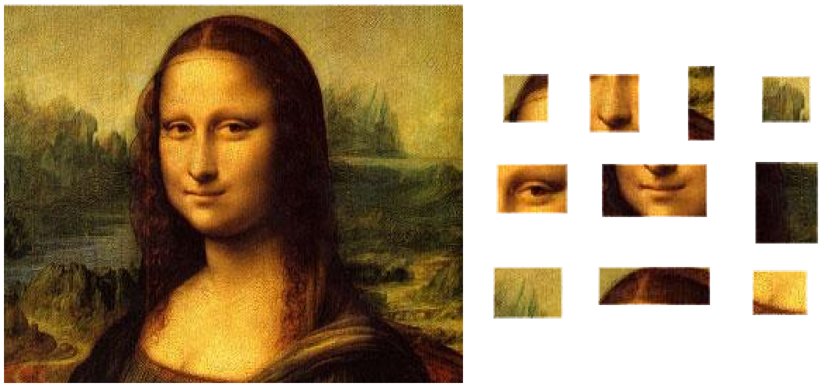
[1]It is crucial to understand differences between detection and identification, that is manifested in the fact that recognition means finding the identity of an object (e.g., voice recognition, face recognition), and detection means finding the existence of an object

them are going to be described below, along with their advantages and disadvantages as a whole and regarding the problem of this research.

3.4.5 Bag of features

One of the methods of feature extraction is Bag of Features or BoF. While working with D-dimensional descriptors, it uses k-means clustering to group local descriptors into k categories, then, using a histogram of the 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 to the scheme have been proposed, including abolishing k-means and using soft quantization techniques instead.

Figure 3.2: Example of BoF



(a) Image **(b)** Features

The nature of Bag of features is essentially Bag of Words (BoW) translated onto image feature extraction. In document classification, Bag of words is a sparse vector of the number of occurrences of each word of a document set dictionary in a document, so it primarily represents frequencies of singled features in an unordered way.

Even though the bag of words is an efficient model that is widely used for feature extraction, it also has its drawbacks. The most significant one is BoW ignoring the 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 the regions of decreasing size 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 thoroughly and adequately tested with regard to performance and various kinds of invariance, nor has it been understood what precisely 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 the 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 the combination of second-order statistics, Sparse Coding and appropriate pooling methods that seems to be soon to start approaching the results that typically were only possible with the 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 useful when it comes to classification of images according to objects pictured of them, while also outputting a constant length vector regardless of the number of detections. It is robust to translation and rotation, but still, the problems arise if one ever needs to localize objects within the image, as it does not explicitly use configuration of positions of the features. Thus, best applications for feature vectors generated with BoF approaches stem into object classification.

3.4.6 Fischer vectors

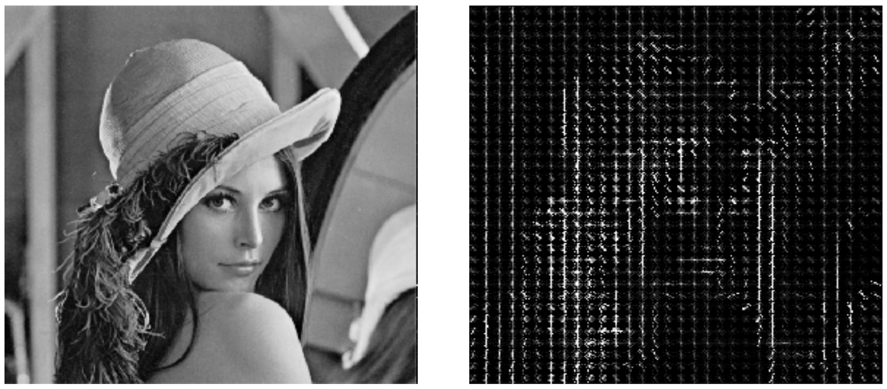
Another widely used technique for getting a fixed size vector representation is the use of the Fisher vector, which is an improved version of Fisher kernel method and is pooling of image features into a vector representation. Fisher kernel assumes estimation of a parametric generative model followed by samples on a training set. The description vector represents parameter space direction for modification of a learned distribution to achieve better fitting of a model, being a gradient of a sample likelihood given its parameters. The inverse square root of Fisher information matrix is used for scaling afterward. It has been shown that after such transformation, classifiers achieve much more impressive performance on the resulting feature vector.

Another good point regarding Fisher kernel is that it has been already used in image classification, showing quite decent results. The resulting vector in this example comes out to be dxk-dimensional, being a diagonal approximation of the Fisher matrix derived from Gaussian mixture model used to obtain incoming vectors. Although this representation is quite sophisticated, it requires less dimensionality of the incoming vectors compared to different approaches.

3.4.7 Histogram of oriented gradients

Another standard feature descriptor is the 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.

Figure 3.3: Example of HOG features



(a) Image **(b)** Features

The idea behind HOG is that one can use edge directions or gradient distributions to describe the local shape of an object efficiently. The image is divided into so-called cells, that are essentially small connected regions, and afterward, the histogram of gradient directions is compiled in order to concatenate these histograms further 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 to lighting conditions such as shadowing and illumination.

The HOG descriptor is a powerful feature, as it has many specific advantages, which make it especially suitable for the task of human detection. For instance, it is invariant to photometric and geometric transformations due to the relative locality of the feature, since this kind of changes only affects regions starting from a certain size. Likewise, it was shown that with coarse spatial sampling, orientation sampling, and strong local photometric normalization, individual movements could be safely ignored provided that the 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 utilizing binary classification. What is even better, it still works well for even for small resolutions, while also requiring very few additional actions to ensure adequate learning. Furthermore, if we want to make 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.

3.4.8 Background subtraction

Background subtraction, also known as foreground detection, is a technique designed to only leave foreground objects for future processing. It is widely used for moving object detection, through comparison of the current frame to the reference frame where the reference frame is the assigned background (even though semantically, it might be a non-background-only picture).

The main problem of background subtraction is determining of what background is. The method assumes that background is static, which is not always the case. However, even if it semantically is, pixelwise it can and will change. Illumination levels might change, and change drastically (e.g., due to clouds), the camera might oscillate, and background trees (or similar object) might also move a bit at high frequencies. Sometimes even background geometry can change, and all these changes need to be taken into account to create robust background subtraction algorithms.

The most basic method of background subtraction is frame difference shown on Figure 3.4. The estimated background here is assumed just to be the previous frame, and despite high sensitivity to a threshold, it is applicable in particular conditions where short-term background oscillations are unlikely.

A background can also be estimated as the average or median value of N previous frames. This method is simple, but memory consuming, due to the need to store those frames in memory. A more efficient alternative is running average:

$$B_{i+1} = \alpha F_i + (1-\alpha) B_i \tag{3.2}$$

where alpha is called learning rate (which is typically equal to 0.05), F is a frame, and B is estimated background. This simple formula allows similar result without memory requirements.

When the need for more complex method arises, another approach to try is selectivity. For a frame, all pixels are classified as either background or foreground, for foreground pixels to be ignored while computing the background estimation. For background pixels, however, background estimation is computed with one of the methods described above.

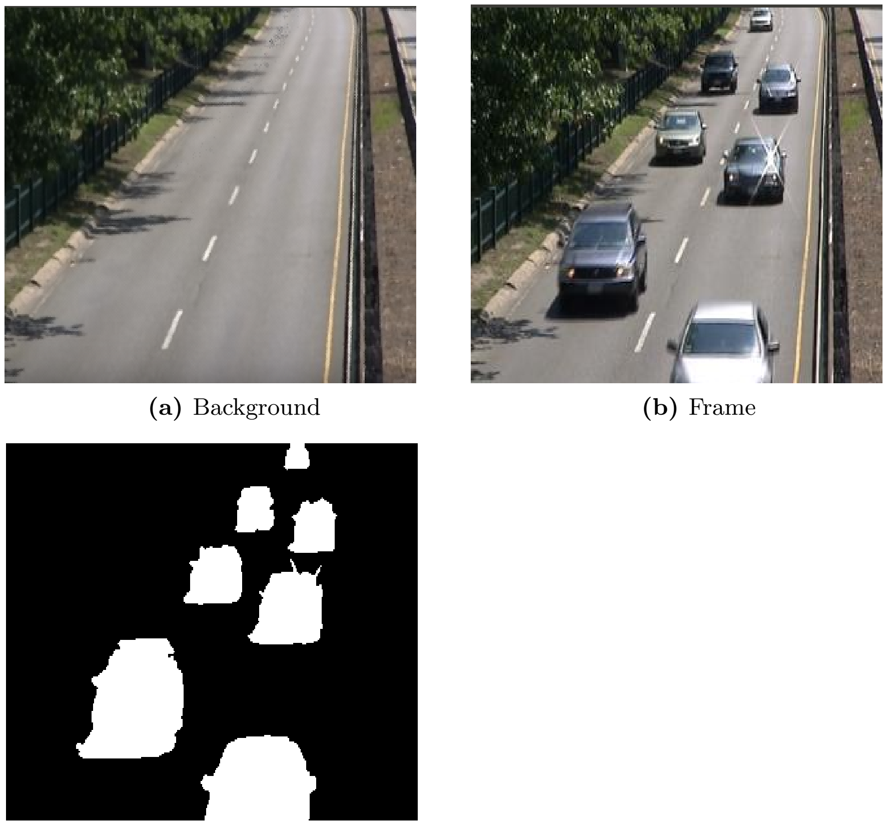
Background subtraction methods covered by now all have the same flaws, namely the lack of explicit strategy of threshold choice, and inability to cope with multimodal background distributions.

Better methods are created to address these problems, such as Running Gaussian Average, which is fitting Gaussian distribution over a histogram thus estimating probability density function (PDF) for the background:

$$\mu_{t+1} = \alpha F_t + (1-\alpha)\,\mu_t \tag{3.3}$$

$$\sigma^2_{(t+1)} = \alpha (F_t - \mu_t)^2 + (1-\alpha)\sigma^2_t \tag{3.4}$$

Figure 3.4: Example of background subtraction


(a) Background **(b)** Frame

(c) Frame difference

The Gaussian approach covers most of the problems with thresholds, yet it is still unable to work with multimodal distributions. The solution here is to mix K Gaussians, where K is usually a predetermined number between 3 and 5. This solution is by itself the problem because a correct arbitrary choice requires experimentations. Furthermore, the question now is on how to initialize and update Gaussians on each iteration.

There are also other methods, growing in complexity, such as Kernel Density Estimators (KDE), Mean-shift based estimation, combined estimation and propagation and eigenbackgrounds, which will not be covered here.

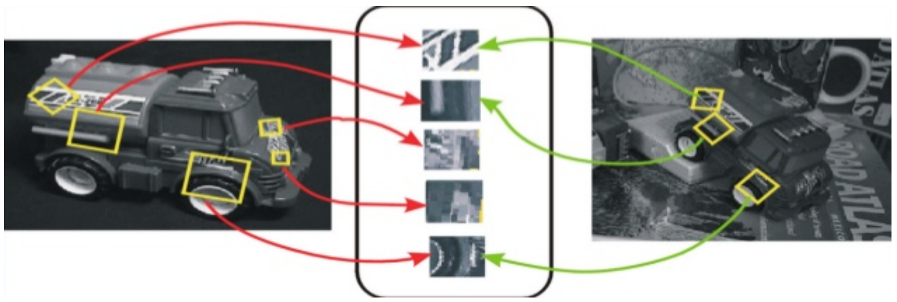
To sum up, the fastest methods are average, median and running average, while the slowest is standard mean-shift. Concerning memory requirements, average, median, KDE, and mean-shift consume the most memory, while the most economical is running average. Since applicability of any of these methods may vary depending on applications, it is safe to conclude that in general, the running average approach is the best first choice, unless there are good reasons to use something else.

3.4.9 Scale-invariant feature transform

More features to go, and now let us consider a classical algorithm for feature extraction called scale-invariant feature transform (SIFT), 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 (Ex. on Figure 3.5). This fact 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.

Figure 3.5: Example of SIFT



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. The 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, which allows for better stability of results by reduction of the contribution of errors occurring because of the 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, which has become classical but is now overwhelmed with drawbacks. To start with, it would not work efficiently on low-powered devices, as it is quite demanding in terms of resources. This fact is easily explainable considering the fact that the method requires a lot of CPU time since we need gradients of each pixel to be computed 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 to be one of the most accurate and reliable ways to describe the pictures even today, regardless of scaling and rotation.

SIFT is widely used in tasks of individual identification of wildlife, video tracking, gesture recognition, 3D modeling and object recognition. 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.

3.4.10 Shape context

Shape context is a feature descriptor proposed mainly 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. To achieve this, n points are picked on the contour of the shape, to construct n-1 vectors for each point by connecting it pairwise to all remaining points. The set of all these descriptions gives the possibility to measure shape similarity and find point correspondences, but still, it is way too detailed at the moment.

The shape context of the point p is the histogram:

$$h_i(k) = \#\{q \neq p_i : (q - p_i) \in \text{bin}(k)\} \tag{3.5}$$

Now we need to ensure some invariances of a descriptor for it to be useful, namely small perturbations, scale, and translation, as well as rotation in some instances. All these distortions are 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 one to provide full rotational invariants, in case it is necessary. It is not always necessary and in some cases even forbidden when it stops one from recognition of different objects as different, since they come out to be similar with respect to rotation, so some local features, if not measured relative to the same frame. Still, if it is needed, one can make it an entirely rotationally invariant descriptor by measuring angles of vectors at each point relative to the direction of the tangent to the contour on that point.

Shape context has been successfully used in tasks like trademark retrieval, 3D object recognition, silhouette similarity-based retrieval, and digit recognition.

3.4.11 GIST descriptor

Now, let us take a look at the GIST descriptor, which was initially proposed in 2001 [90]. 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 eight scales and four orientations, giving us 32 equally sized feature maps, each of them is then divided into a grid, dimensioned 4x4, to further average feature values inside each grid cell. All averages are then concatenated across every parameter, given a vector of 512 values for each parameter. Since this is a lot, different compression strategies are used.

GIST descriptor is notable for being incredibly compact and fast to compute while holding plenty of valuable information. On the other hand, invariance is quite low, strong geometric constraints often fail to hold, and occlusions raise severe difficulties in pattern recognition.

The GIST descriptor has recently been shown to be efficient for duplicate detection, scene alignment and scene categorization in databases containing millions of images. Location recognition is another recent accomplishment.

3.4.12 Haar-like feature

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, which 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, to further subtract 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 frequent almost-always-true observations regarding some areas of the picture having a specific Haar pattern, and that might help to recognize an object provided a bounding box be given.

However, only one Haar-like feature is a weak classifier, so many features need to be extracted to have a prediction at least half-true. They are usually calculated across a bounding box of target size, that is moving across the image until all positions are gone through, or an object is found. During the process, collected Haar features are arranged into a strong learner that is also sometimes called a cascade.

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.

3.4.13 CV methods conclusion

There is a notably significant amount of methods for extracting features from images for different purposes, and in this chapter, we only covered a tiny fraction of them. The reason for it is 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 to some kinds of transformations is presented in the 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 different 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	More or less
Background subtraction	If scaling applied	Depends on applied method	Sensitive
SIFT	Robust	Robust	Robust
Shape context	Robust	Robust	Rotation — on demand, Translation — robust
GIST	Robust	Robust	More or less
Haar-like	Robust	Robust	Rotation – sensitive, Translation – robust

3.5 Machine learning

Machine learning is a class of artificial intelligence methods that are notable because instead of straightforward problem solving, learning by observing solutions of a 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 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 appropriately 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, or a principle of work of the algorithm. Say, there are clustering algorithms for unsupervised learning, when we do not have labels for datasets, or prediction algorithms for supervised or semi-supervised learning, where at least part of a dataset available is adequately labeled. On the other hand, there is the regression, which 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), polynomes (polynomial predictors), trees (such as decision tree or boosting algorithms), or perceptrons (neural networks).

Now let us consider all the notions listed above in ordered fashion and dive deeper into details.

3.5.1 Unsupervised vs. supervised

In practice, most of the machine learning one would run into is supervised learning. The point of this broad category of approaches is mapping a set of input variables (x) to an output variable (Y). One uses the algorithm to approximate the mapping function between this variables to provide an appropriate output to every input. Knowing correct answers, we can adjust the result iteratively and stop once acceptable level of performance is achieved.

There are two kinds of supervised learning:

- *Classification* — output is categorical. Typical use cases include mail sorting, and medical diagnostics, as well as object, pattern, speech and handwriting recognition.

- *Regression* — output is a real value. Examples are the prediction of prices or sizes.

Unsupervised learning, on the other hand, means that one only has input data (X) which is not labeled by output values in any way. The goal of this approach is to learn distribution or structure of data to extract more information and get useful insights, so, since there is no output data present, algorithms are left to themselves to discover patterns in datasets.

Just like with supervised learning, unsupervised learning can be further grouped into two kinds:

- *Clustering* — aims to discover natural groupings of data. Used virtually everywhere, for example in ecology to describe groupings of individuals inside a population, in physical geography to cluster physical properties, or for image segmentation.

- *Association* — aims to discover rules that describe large portions of the given data, in the form of relations between variables. Typical use cases are market basket analysis, malware detection and e-learning.

3.5.2 Classification vs. regression

Classification, as was said above, is the task of mapping a function (f) from input variables (X) to output variables (Y) with a discrete finite domain. Output variables, in this case, are often called labels, or categories.

Examples of applications of classification models:

- Medical image analysis
- Optical character recognition
- Video tracking
- Speech, handwriting and pattern recognition
- Geostatistics
- Credit scoring

By definition, classification is the 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 particular sort of ordering, based on an extent of similarity. Alternatively, on the other hand, a set of objects ordered based on some principle, where objects have some classificational properties that are chosen to determine how similar or different they are.

Regression, on the other hand, is the task of mapping a function (f) from input variables (X) to continuous output variables (Y). Y can be integers or floating-point values, which often represent quantities. Because of that, the quality of prediction is a function of the error in prediction.

Some algorithms have the word regression in their names, but not all of them are regression algorithms. For instance, while linear regression is a regression algorithm indeed, logistic regression as in fact a classification algorithm.

Some algorithms can be used exclusively either for regression (e.g., linear regression) or classification (e.g., logistic regression), while other algorithms, such as decision trees or artificial neural networks, can be applied to both cases with minor modifications.

3.5.3 Binary classification

Binary or binomial classification is the task of classifying elements into precisely two groups, examples being:

- Medical testing whether a patient has a given disease
- Pass or Fail tests
- Deciding whether a page or an article should appear in search results

Most often, sizes of categories are not equal, and errors are skewed - that is, sometimes False positives are more critical, and sometimes False negatives are. This kind of classification is quite easy and needs simplest models. In fact, many machine learning methods are initially created for binary classification and then adapted to multiclass classification through one-versus-one or one-versus-all strategies. On the other hand, one can often need more labels to work with than just YES or NO.

3.5.4 Multiclass/multilabel classification

Multiclass classification as classification where there are more than two classes to sort items into. One item can belong to one class. This variation of classification is more flexible, yet some classifiers are not naturally tailored for anything except binary classification, so clever workarounds are needed.

Let us now make one step forward in complexity and consider multilabel classification. Now, there are many categories, but any object can have multiple labels — even all at once. This feature is usually modeled with binary classifiers for each category that answer the question whether an object belongs to a given category or not. An example is an answer to the question What is in the picture?.

3.5.5 Probabilistic approach

Often, classification models do not just yield labels but instead predict continuous values that serve as probabilities of a given example belonging to each of the classes. These probabilities can be interpreted as confidence or likelihood of prediction. A predicted class is then assigned the class label with the highest probability. With binary classification, it is common to set another arbitrary threshold between classes. For example, if the prediction is whether a patient has cancer or not, the threshold can be as low as 10% which means that patient having more than 10% probability of cancer will not be classified as healthy. From this example, it is clear that such manipulations are done when the cost of type I error is different from the cost of type II error, and the threshold is chosen to minimize this cost.

The idea of probabilities yielded instead of bare labels leads us to another two important notions — discriminative and generative models.

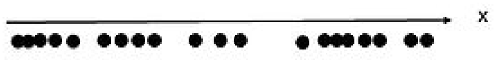
Discriminative models, or conditional models, are those that model the dependence of target variables (Y) on observed variables (x). From the probabilistic point of view, the probability function P(y—x) is what is being modeled. This function can be used to predict Y given X. Unlike generative models, that are considered more thoroughly below, discriminative models do not model enough information to generate more samples. However, for tasks that do not require the joint distribution, such as classification and regression, discriminative models are often more preferable, due to superior performance explained mostly by the lesser amount of variables to compute. On the other hand, they lack the flexibility required for more complex tasks, and also only support supervised learning. Ultimately, the choice of generative versus discriminative model is dictated by needs in every specific application.

Now, let us talk about generative models — these are models that generate all values for a phenomenon, both target variables computed from observed data and values that can be observed. In other words, they generate both outputs and distributions of inputs, often given some hidden parameters. Generative models thus specify joint probability distributions over observations and for target values, given the possibility to yield all probability combinations that arise in a data set. These combinations are used either as an intermediate step for prediction or direct data modeling.

3.5.6 Clustering

As was mentioned above, clustering pretty much consists of dividing values into classes which are not predetermined, where the number of classes is often unknown in advance. There are two kinds of clustering — hard clustering and soft (fuzzy) clustering. Hard clustering — each object belongs to exactly one cluster. Soft clustering — each object belongs to a cluster to a certain degree (in other words, we have probabilities of belonging to a cluster) For more details, let us consider an example of monodimensional data (Figure 3.6):

Figure 3.6: Monodimensional data



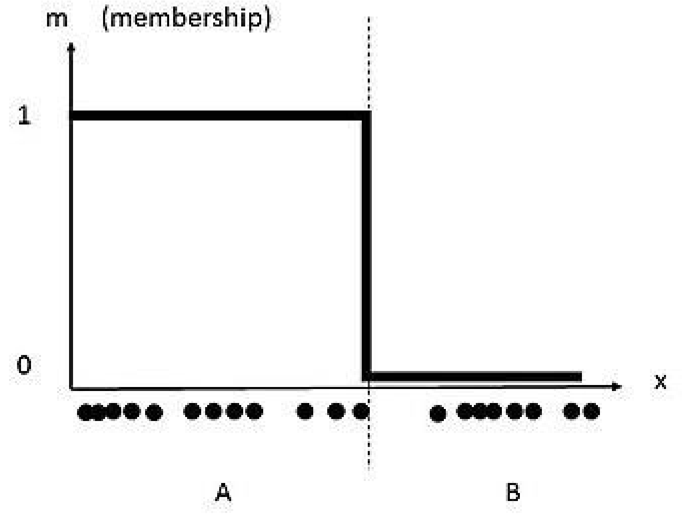
Let us now split this dataset into two clusters. For that, a threshold is selected, and resulting clusters are labeled A and B. This way every data sample will either have a membership coefficient of 1 or 0. The graph of this coefficient is shown on Figure 3.7.

In fuzzy clustering, on the other hand, each sample can be a member of multiple clusters, to an extent. The membership coefficient is thus a value in the range between 0 and 1. Now, on the Figure 3.8, a threshold is redefined, using c-means fuzzy clustering. New membership coefficients are generated based on cluster centroids.

The point highlighted in the image, for example, has a membership coefficient of 0.3 with respect to cluster A. Thus fuzzy clustering 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.

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

Figure 3.7: Hard clustering



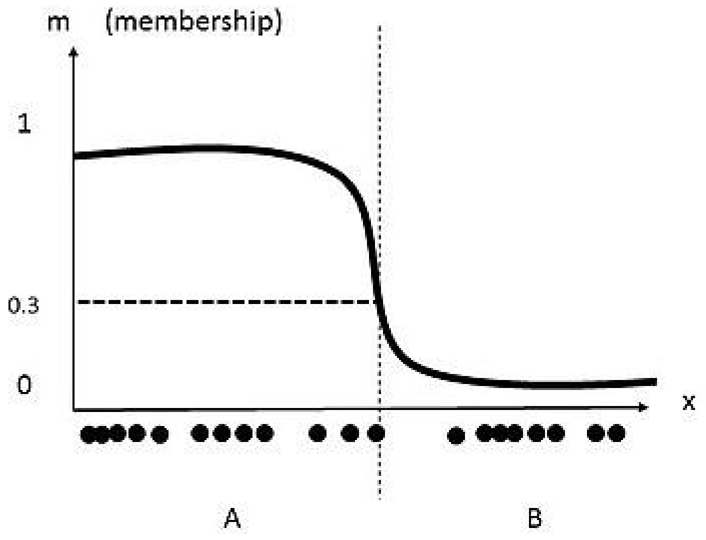
The k-means is the most popular of them, created as early as the 1950s. The algorithm aims to minimize the sum of quadratic deviations of elements from cluster centroids:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2 \tag{3.6}$$

In fact, k-means, k-median, and k-center are all very similar and follow the same pattern, to the 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 center, and then all other points are assigned to the closest of the clusters. Then, centers are rearranged to minimize a particular function, that will be considered later. After that, a new check performed on all object to reassign them to clusters they are closest to. The process is then continued until no object is assigned a different cluster compared to the 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 a sum of squares of distances between a center and objects of a corresponding cluster, or L2 loss function. For k-medians, it is a sum of distances between a center and objects of a corresponding cluster, or L1 loss function. So 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.

Figure 3.8: Fuzzy clustering



What is good or bad about these methods? Let us first start with disadvantages. The first issue is that Euclidean distance measures or any similar metrics will not most probably weight underlying factors by their importance, so at the very least, it is useful to use weighted versions of these metrics. Then, one will need to go through significantly more iterations if you want to achieve a slightly better result, that is sometimes even disappointing. Lastly, as with every clustering method described here, one needs 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).

3.5.7 Distance-based classifiers

For any two N-dimensional vectors, numeric or categorical, it is possible to define the distance. It is some function of two vectors(or points), which is numeric, non-negative, symmetric, defined in such a way that d(x, x)=0, and conforms to triangle inequality. In this case, the distance will be a measure of similarity, or rather, the dissimilarity between vectors. Common numeric distance formulas shown in the Table 3.2.

For categorical vectors the computation of similarity is more uniform, being

$$sim(x, z) = \sum_{d=1}^{D} sim(x^d, z^d) \tag{3.13}$$

with component similarity functions being as easy as 3.14

Table 3.2: Common numeric distance formulas

Name	Formula
Euclidean	$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{\sum_{i=1}^{D}(q_i - p_i)^2} \quad (3.7)$
L_p	$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt[p]{\sum_{i=1}^{D}(q_i - p_i)^p} \quad (3.8)$
L_∞	$\max_i =_{1,2,\ldots D} \mid p^i - q^i \mid \quad (3.9)$
L_1	$\sum_{i=1}^{D} \mid p^i - q^i \mid \quad (3.10)$
Canberra	$\frac{1}{D}\sum_{i=1}^{D}\frac{\mid p^i - q^i \mid}{\mid p^i + q^i \mid} \quad (3.11)$
Lance-Williams	$\frac{\sum_{i=1}^{D} \mid p^i - q^i \mid}{\sum_{i=1}^{D} \mid p^i + q^i \mid} \quad (3.12)$

$$sim(x^d, z^d) = \|[x^d = z^d] \tag{3.14}$$

or as complex as 3.15

$$sim(x^d, z^d) = \begin{cases} 0, & x^d \neq z^d \\ K(p(x^d)) & x^d = z^d \text{ for some} \downarrow K(u) \end{cases} \tag{3.15}$$

where

$$K(p(x^d)) = \frac{1}{p(x^d)^2} \tag{3.16}$$

or

$$K(p(x^d)) = 1 - p(x^d)^2 \tag{3.17}$$

The link between similarity and distance can be easily defined, for example, as 3.18

$$sim(x_{num}, z_{num}) = F(\rho(x_{num}, z_{num})) \text{ for some} \downarrow F(u) \tag{3.18}$$

e.g.

$$sim(x_{num}, z_{num}) = \frac{1}{1 + \rho(x_{num}, z_{num})} \tag{3.19}$$

With distance being a measure of dissimilarity, it is natural to try to use distance-based algorithms for machine learning, using values for vectors close enough to determine the output for a given vector.

One of such algorithms is called K-Nearest Neighbors (K-NN). In this algorithm, for a given instance, one determines K training instances closest to one we are predicting. Then, voting or weighted voting among them is employed to determine the output.

KNN is simple to implement, fast to train, flexible due to the multitude of variations of both distances and weights, handles multiclass classification naturally, as well as regression, and with enough data, applicable quite well in practice. On the other hand, this algorithm is incredibly slow in prediction, being linear both in terms of training set size and vector length, consumes much memory since all training instances must be stored, and only works when it is possible to apply a meaningful distance function.

KNN is widely applied in the following areas:

- Concept search, i.e., searching for documents with similar topics
- Recommender systems
- Face recognition systems

3.5.8 Linear classifiers

In the field of machine learning, a common task to solve is to find a mapping between inputs and outputs. Linear machine learning models, thus, employ linear functions 3.20 as approximations of these mappings:

$$y = F(\overrightarrow{w}\,\overrightarrow{x}) = F(\sum_j w_j x_j) \tag{3.20}$$

Typically, linear classifiers can only solve linearly separable problems, i.e., such that there exists a hyperplane dividing the classes. However, even in non-linear cases, it is possible to make linear models work for the cause by introducing kernel functions to work instead of a conventional scalar product.

Linear classifiers are often used when prediction needs to be fast, due to them being faster than most other models while also having acceptable scores, especially with large-dimensional and sparse vectors.

3.5.9 Support Vector Machine

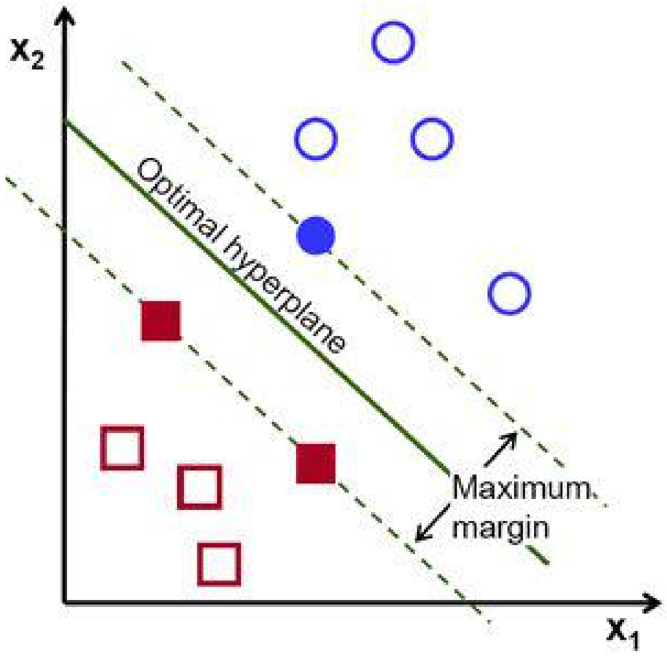
The first model for us to consider is Support Vector Machine (SVM). Unlike distance-based methods, it aims for the most optimal class boundaries and thus achieves better accuracy, while also being more complex, due to more information being taken into account. It is essentially a linear model with hinge loss and L2 regularization. This method aims 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 in case of nonseparable classes, less than one minus specific constant that is also optimized. In addition to linear classification, SVM can also perform non-linearly with use of alternative kernel functions instead of standard scalar product, implicitly mapping input features to more dimensions. On 3.21 one can see how optimization task for SVM looks like.

$$\frac{1}{2C} \mid w \mid^2 + \sum_{i=1}^{N} [1 - M_i(w, w_0)]_+ \rightarrow \min_{w,\xi} \tag{3.21}$$

SVM is pretty widely used everywhere where machine learning is in use at all, and for some good reasons. First and foremost, it is the fastest existing method to find discriminative functions. Also, the method is fundamentally based on solving a quadratic programming problem in a convex area that implies there is always the only solution. So what is best, the method performs the search for the broadest possible band, which 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 the 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 one faces such a situation, they need to choose a kernel manually and hope for the best. Finally, this method proves to be the slowest across all the linear models.

The most successful applications of SVM include:

Figure 3.9: Support Vector Machine

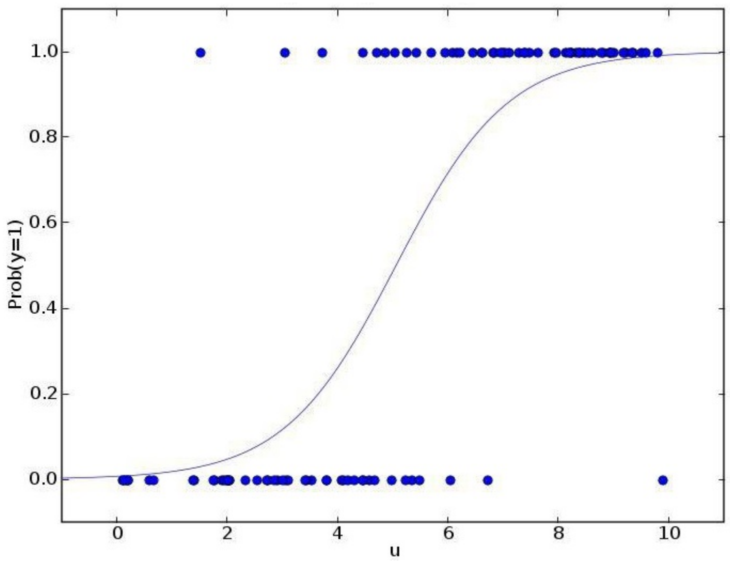


- Face detection
- Text and hypertext classification
- Image classification
- Bioinformatics — protein classification, cancer classification
- Handwriting recognition

3.5.10 Logistic regression

Let us now consider another classification method, called logistic regression. It is especially suitable when the 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 some particular advantages over discriminant analysis. In fact, there are so many advantages that they deserve a separate paragraph. To start with, feature values can freely be bounded or non-interval. No assumptions about features are made. 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 possible to add power terms or explicit interaction, and, as it could have already become apparent, 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, that does not have to be assumed.

Figure 3.10: Logistic regression



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 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. On 3.22 one can see how optimization task for logistic regression looks like.

$$\sum_{i}^{n} \ln(1 + e^{-\langle w, x_i \rangle y_i}) + \to \min_{w} \tag{3.22}$$

Most successful applications of logistic regression include:

- Image segmentation and categorization
- Geographic image processing
- Handwriting recognition
- Prediction whether a person is depressed or not based on the words they use

3.5.11 Naive bayes

Naive Bayes classifiers are the whole family of probabilistic classifiers based on Bayes theorem with strong independence assumptions made for the features. This kind of classifiers is around since as early as the 1950s and is extensively used for text categorization. With appropriate processing, it can beat or at least compete with SVMs and even more advanced methods in certain fields, including among all others medical diagnosis. This family of classifiers is highly scalable, with appropriate training performed in linear time, in one iteration. On the other hand, when features are not strongly independent, the method will fail miserably.

The classification function for Naive Bayes shown on 3.23.

$$\widehat{y}(x) = \arg\max_{w} p(y)p(x^1 \mid y)p(x^2 \mid y)...p(x^D \mid y) \tag{3.23}$$

3.5.12 Decision tree

A decision tree is a decision algorithm using a tree-like graph or model of decisions and possible consequences. For splitting the nodes, various decision rules are employed, such as categories, ranges or conditions, sometimes even multivariate conditions.

Decision trees are simple, interpretable, able to implicitly perform feature selection and handle all kinds of features naturally well, even being able to cope with missing data, which makes the algorithm quite valuable. On the other hand, these models are not entirely accurate, prone to overfitting and do not support reinforcement learning.

Use cases of decision trees are numerous:

- Star galaxy classification
- Nonlinear dynamic systems
- Medical diagnostics
- Amino acid sequences analysis
- Estimation of development effort for a given software module
- Medical text classification

3.5.13 Combinations

Sometimes, one may decide to combine multiple models into an ensemble, to either reduce the risk of overfitting or increase resultativity. There can be similar or different models, working consequently or in parallel, or with any other arrangement. Two popular methods of combining models are boosting and bagging.

Boosting is an algorithm for primarily reducing bias, as well as a family of algorithms to convert weak learners (slightly better than random) to strong ones. It has grown from a big question: is it possible to produce a strong learner from many weak learners?

The procedure is sequentially building an ensemble in such a way that the next algorithm corrects mistakes of already existing ensembles. This algorithm is thus greedy. For the last decade, boosting remains one of the most popular machine learning methods, along with neural networks and SVMs, due to simplicity, universality, flexibility and high generalizability.

Decision tree boosting, for example, proved to be one of the most efficient classification methods. Many experiments showed that error rate on the training set was declining almost infinitely, with test set error rate continuing to decline long after the training set error rate fell to zero. It was a shocking discovery, overthrowing the common belief that too complex models always reduce the quality of prediction. The phenomenon was later theoretically explained.

Bagging, also called bootstrap aggregating, on the other hand, is an ensemble meta-algorithm used to prevent overfitting by reducing variance, as well as to achieve stability in results. It is a special case of model averaging approach — all models are trained independently, with the result being determined by voting. The idea is that classifiers do not correct each other, but instead compensate each other in voting. The base classifiers must either be based on different methods or be trained on different sets, to be independent.

3.5.14 Random forest

Let us now talk about a specific example of an ensemble, called a random forest. The catch is to construct a multitude of decision trees on training randomly and output the mode of classes for classification, or an average for regression. Randomness here exists to correct decision trees proneness for overfitting.

Random forest helps to overcome many problems that exist with decision trees. The overfitting is reduced due to averaging, the variance is decreased, and thus random forests almost always perform better than single decision trees.

On the other hand, disadvantages are indeed numerous. Random forest is more complex, less interpretable, more challenging to implement, and more computationally expensive. Sometimes, reduced risk of overfitting compensates for all these drawbacks. Applications for random forests in computer vision include:

- body part classification
- head pose estimation
- pedestrian detection
- body pose estimation

3.5.15 Neural networks

Artificial neural networks (ANNs) are systems inspired by biological neural networks animal brains consist of. They are an advanced method to avoid task-specific programming. An ANN consists of connected nodes, also called artificial neurons. Each connection can transfer information between nodes. A node can then process the signal and send it further across the network through connections.

Most commonly, a signal is a real number, and the output of each neuron is a function of its inputs (or commonly, a function of a sum of its inputs). Typically, nodes and connections also have weights that adjust during the learning process, which are responsible for the strength of the signal. Nodes can also have a threshold that cuts all inputs lower than a particular value. Typically nodes are arranged into layers, where each layer is dedicated to one kind of transformation.

Initially, the goal of the ANN approach was to solve problems in a way similar to how a human would solve them. However, soon the focus shifted to more specific tasks, thus forging the drift away from biological similarity. Now, ANNs are used in a wide variety of ways, including medical diagnosis, games, machine translation, speech recognition, and computer vision.

3.5.16 Neural network building blocks

For ANNs, three key components matter:

- Network Topology
- Adjustment of weights or Learning
- Activation functions

Let us now look at each of these components in detail.

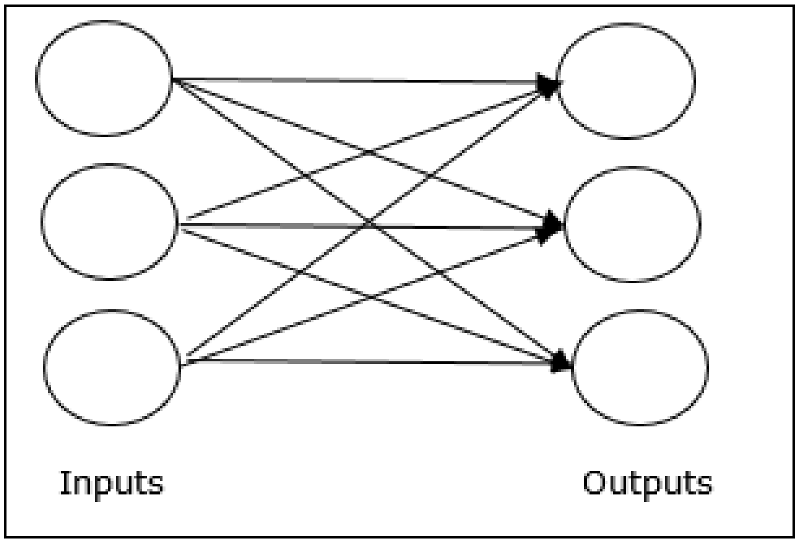
A network topology is the arrangement of nodes of a network, with layers and connecting lines. By topology, several kinds of ANNs can be outlined, such as

- Feedforward networks
- Feedback networks

Let us first look at feedforward networks — this is a kind of non-recurrent network that can be divided into layers, with nodes in a layer being connected to nodes of the previous layers, with different weights upon them. The signal here only flows in one direction, from input to output. Simplest feed-forward networks are single-layer, which means they only have the input layer fully connected to the output layer, with nothing else in between (Ex. on Figure 3.11).

Multilayer networks (Ex. on Figure 3.12), however, have some number of hidden layers in between, differently weighted, so are able to process more complex data.

Figure 3.11: Example of single-layer feed-forward network



Now, let us look at feedback networks, which have some feedback paths, allowing the signal flow in both directions using loops, which makes a network non-linear dynamic system, which continues forcing the flow until equilibrium is reached.

For example, recurrent networks are feedback networks with closed loops (Ex. on Figure 3.13). Two well-known types of recurrent networks are fully recurrent networks and Jordan networks. Fully recurrent networks are quite simple in architecture, as every node is connected to every node, with all nodes working both as input and as output.

With Jordan networks, the matters are even simpler than that, with outputs going back to inputs as feedback (Ex. on Figure 3.14).

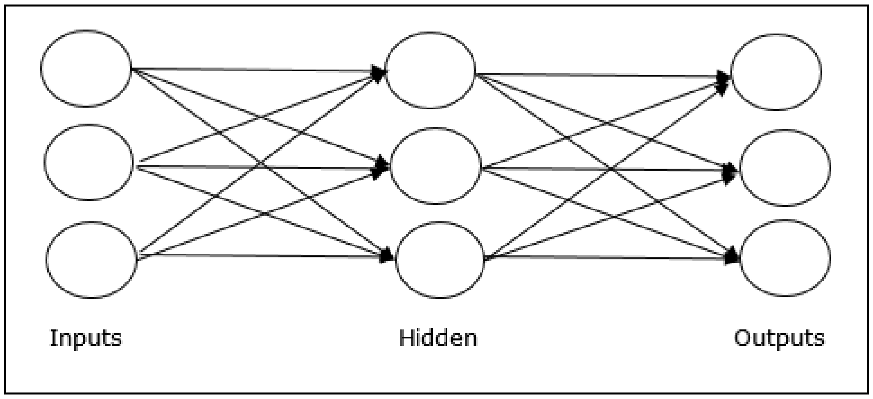
After the network architecture is finalized, it is time to think about adjustments of weights, also called learning. It is the method of modifying the weights of connections between the network neurons. One can distinguish supervised learning, unsupervised learning and reinforcement learning.

With supervised and unsupervised learning, the situation is just as straightforward as for any other predictor. With reinforcement learning, however, it all becomes more interesting (Ex. on Figure 3.15). It is the learning used on a pre-trained network to strengthen it over some critical information. It is similar to supervised learning but requires a pre-trained model, yet training sets are way less in this case.

The network receives some feedback from the environment, yet the feedback is evaluative, not instructive. After that, weights are adjusted further.

Let us now move on to activation functions — the last critical component of neural networks. They can be defined as extra efforts required to convert the

Figure 3.12: Example of multilayer feed-forward network



input to output, or functions performed on input, and are applied to nodes, not a network as a whole. The simplest of them is a linear activation function or identity function 3.24

$$F(x) = x \tag{3.24}$$

Other popular activation functions include sigmoids, which can be binary or bipolar. Binary sigmoidal function renders the output to be between 0 and 1, and is strictly increasing 3.25

$$F(x) = sigm(x) = \frac{1}{1 + exp(-x)} \tag{3.25}$$

Bipolar sigmoidal function, on the other hand, fits the output into (-1; 1) range, while also being strictly increasing 3.26

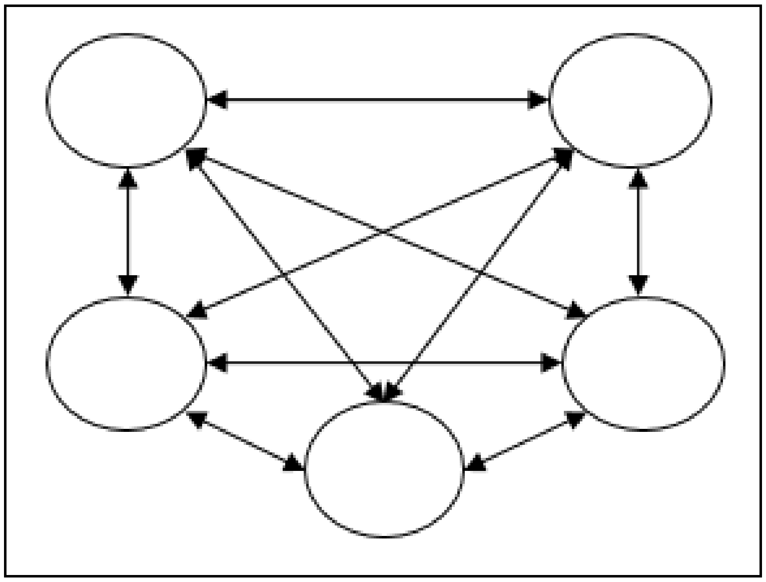
$$F(x) = sigm(x) = \frac{1 - exp(x)}{1 + exp(x)} \tag{3.26}$$

3.5.17 ANN applications

Here is a longer (but in no way exhaustive) list of successful ANN applications:

- black-box models in geoscience
- cancer diagnostics
- email spam filtering
- machine translation

Figure 3.13: Example of feedback recurrent network



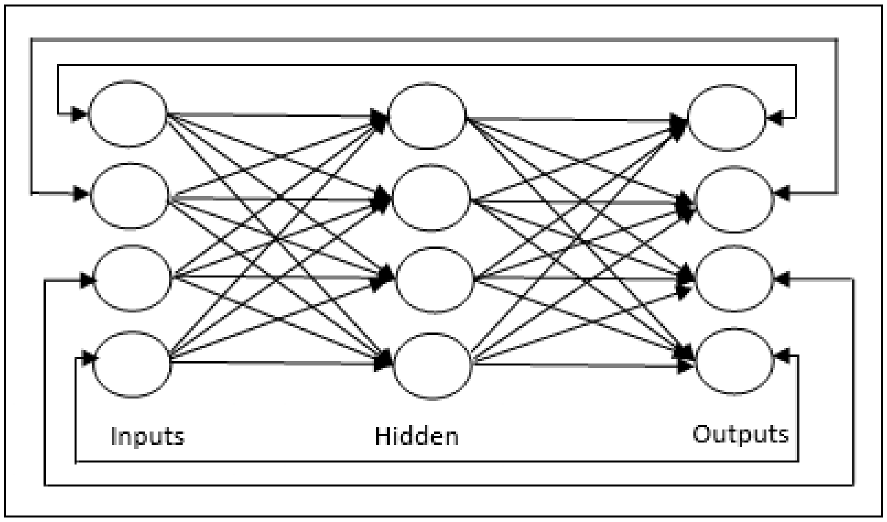
- data mining
- automated trading systems
- gesture, speech, handwritten and printed text recognition
- object recognition
- face identification
- radar systems
- game-playing and decision-making
- quantum chemistry
- trajectory prediction

With current rate of progress, the actual areas of use seem to be going to start expanding almost exponentially.

3.5.18 Convolutional neural networks

Let us now talk about one particular kind of ANNs. Convolutional neural networks (CNNs) are deep, feed-forward ANNs, which have been successfully applied to the field of computer vision. CNNs use a particular kind of multilayer perceptrons designed in a way that would require minimal data preprocessing, and make a network naturally invariant to translation. CNNs were inspired

Figure 3.14: Example of Jordan networks



by principles of information processing in the animal visual cortex. Individual neurons only react to stimuli in a restricted area, but such areas for different neurons overlap, together constituting the whole visual field. A good advantage of CNNs in computer vision is that these networks learn the filters that were explicitly programmed in previous algorithms, making them independent of a lot of efforts and prior knowledge. Aside from visual recognition tasks, CNNs proved to be applicable to natural language processing and recommender systems.

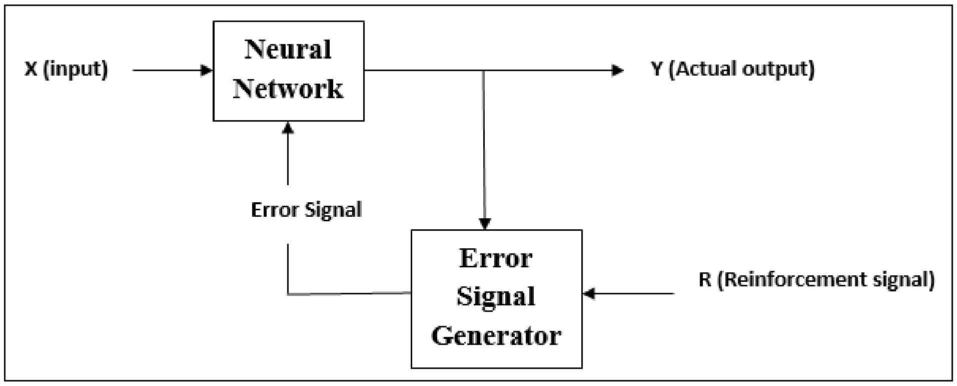
3.5.19 Region-based CNNs

Now, we are going to discuss a well-known family of CNNs, namely R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN, all recently created by a group of researchers over the course of several years.

The R-CNN takes an image as input, correctly outputting bounding boxes and labels for all principal objects in the image. The way it makes the prediction is putting plenty of boxes into an image and seeing if any of them contain an object. This process is called region proposal, which is done through the method called Selective Search — taking windows of different sizes, traversing the image through them, and then trying to identify objects by grouping pixels by some parameters.

After proposal creation, the region shape is reformed into a square of a standard size, to pass through a modified variation of AlexNet. In the final layer, an SVM starts working to classify an object. After the object is found, the final task is to tighten the bounding box to reproduce actual dimensions of an object. For that, a simple linear regression is run on the region proposal

Figure 3.15: Example of reinforcement learning network



to produce coordinates. While R-CNN achieves significant accuracy, it is also painfully slow, both since every single region proposal has to be passed through AlexNet, making up several thousand passes per picture, and also the need to train all models separately and then arrange them into a working pipeline.

In 2015, the problem of speed was finally solved, yielding a new model called Fast R-CNN. The catch is, most of the proposed regions overlap significantly, so computations are repeated for hundreds of times. Thus there should be a way to speed up the computations by only making one pass of the whole picture to the CNN. This way was found soon, and called Region of Interest Pooling (RoIPool) – now for each region proposal, only specific features needed to be taken out of the CNN output.

Moreover, the Fast R-CNN was trained jointly, as if it was a single solid model, or to be more exact, started using the single network for all three tasks – feature extraction, classification, and bounding box regression.

Still, even with these improvements, the region proposal has become a bottleneck, due to Selective Search being quite slow. Soon, a team from Microsoft Research found a way to make this step nearly cost-free, in the next generation of the system called Faster R-CNN. Because region proposals depended on image features output by the CNN, researchers decided to use these result for region proposals instead of employing a separate algorithm.

The next step was performed just recently by a team of researchers from Facebook AI that included the original author. The new architecture, named Mask R-CNN, allowed solving the problem of image segmentation instead of just extracting bounding boxes. The Faster R-CNN got modified with an auxiliary branch which outputs a mask showing whether a pixel belongs to an object or not. The branch was just a Fully Convolutional Network on the top of the feature map output by the CNN. For now, this family of algorithms is a huge breakthrough in the field of object recognition.

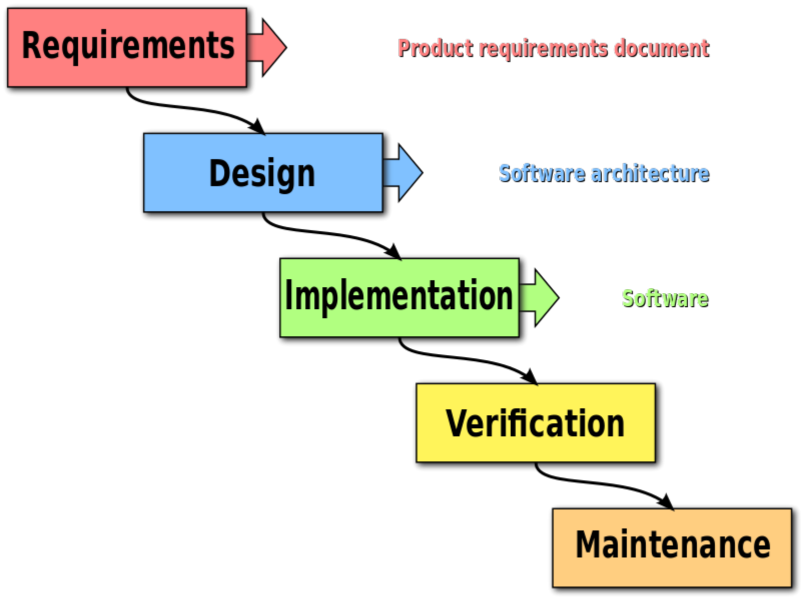
3.6 Software development techniques

Since the task for this thesis project is to develop a system with some functionality, there is an urgent need to consider different software development methodologies to be confident regarding what techniques we are going to use and why.

3.6.1 Waterfall model

The first one, and the earliest methodology stemmed directly from engineering (namely in manufacturing and construction), is the waterfall shown on Figure 3.16. The model is linear, meaning that all stages are passed consequently and not revisited.

Figure 3.16: Waterfall model



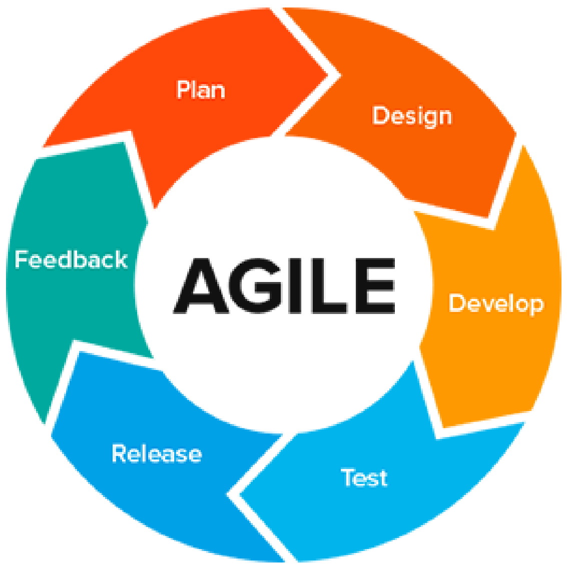
The logic is that time spent early on specification and planning can reduce costs at the later stages, along with the time spent coding. Thus the approach is about going through each stage thoroughly, with even large projects having very detailed specifications. Another argument for the use of the waterfall model is a heavy emphasis on documentation, as well as source code, which allows for easy developer interchangeability. It is also easily understandable, highly structured and provides identifiable milestones. This fact makes it perfect for cases where requirements are fixed, the end product is stable, and employed technologies are well-known.

However, it is not very practical when it comes to most of the products developed in the industry. Most of the time, clients tend to dynamically change the requirements after seeing some version of the product, leading to substantial redesign and so on, which implies a drastic cost increase. Furthermore, some details of constraints and requirements can only be discovered in the process of coding or testing, making the team to redesign the system again. These facts led to further modifications of the approach, as well as to the rise of entirely new methodologies.

3.6.2 Agile software development

A relatively newer software development methodology, Agile is nowadays one of the most popular approaches to software engineering. Within this technique, requirement and solutions are expected to continuously evolve through collaboration, with such features as adaptive planning, evolutionary development, early delivery, and continual improvement, so that all changes are addressed urgently.

Figure 3.17: Agile software development



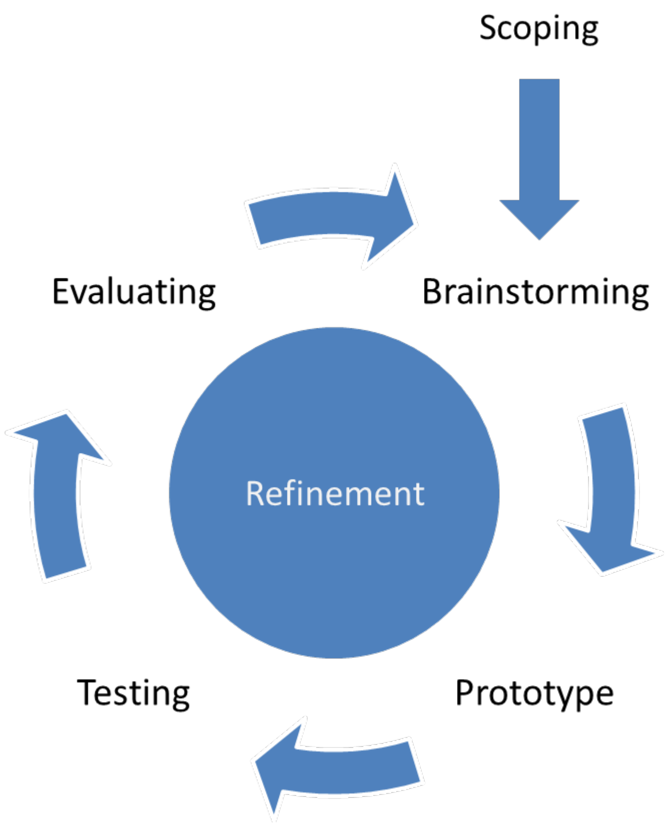
The best parts of agile software development are the acceptance of uncertainty existing in requirements and issues, faster review cycles leading to the better product, greater flexibility in releasing features, and less upfront work implying no more need to produce lengthy documentation while developers are doing nothing. However, everything comes at a price, and the price here is that agile principles are often misunderstood leading to pure chaos. Examples are dysfunctional choices made with no reference to check against, the need for specific organizational structure allowing for deep interaction, lack of predictability which might lead to the loss of strategic vision, the inherent difficulty of the creation of entirely cross-functional teams, as well as reduced scalability of the approach.

While anecdotal evidence strongly suggests that agile indeed helps to produce better products, some empirical studies have not found such connection.

3.6.3 Software prototyping

Software prototyping is about creating prototypes of software applications, or incomplete versions of software to be produced, as the name would suggest. It is similar to prototyping in such fields as manufacturing and mechanical engineering. A prototype need not resemble the final product, as its purpose is to simulate only certain aspects of the software being developed.

Figure 3.18: Software prototyping



Prototyping as a valuable practice in a sense that it allows to obtain useful feedback on the early stages of product development. The prototyped parts can be compared to specification so that changes are rapidly made if deemed necessary. It also allows for insights regarding project planning and time estimates, so that milestones are set realistically. Nevertheless, the convenience of the method is dimmed by increased development time and costs unbearable by tight budget teams, and reduced freedom for software designed stemming from the architecture of the prototype.

3.6.4 Iterative development

Iterative development is about breaking down the development of a system into small chunks so that the project is designed, developed and tested cyclically. In every iteration, new features can be added, until the fully functional software is ready to be released.

Iterative development is most suitable for cases when clear requirements are present, major requirements are known from the beginning while minor ones can evolve with time, the time-to-market is limited, an innovative technology not yet learned by the team is involved, and goals can be changed over time.

Experience shows that iterative development is quite a useful technique. It allows spotting potential defects early, to have functional working versions on early development stages. It implies more time on design and less on documentation, easy progress measurement, more straightforward implementation of changes, facilitation of testing, easy identification of risks. Iterations can be mapped to milestones, a new version of a product delivered on any iteration, operating time is reduced. Still, drawbacks are also numerous. More resources are required. Successive iterations cannot be overlapped, so each one is a miniature waterfall; project management can become more intensive, and the end date is often hard to predict. Risk analysis here becomes complex enough to require a highly skilled professional.

3.6.5 Pair programming

While not being a software development methodology in and of itself, pair programming is an agile software development technique where two programmers work at the same code at the same time at the same workstation. One programmer, called the driver, is writing the code, while another performs the line-by-line review, both switching roles frequently. The observers role is also to consider the strategic view on the work, proposing improvements and highlighting future problems to solve. The intention is to let the driver free to concentrate on more tactical issues for the current task.

Advantages of the method include two points of view, natural catching of small accidental mistakes, better concentration, the possibility for knowledge combination, and development of soft skills in the process. On the other hand, it is not the best method in a case of skill disparity, can quickly turn into socializing sessions and also problems can arise when both developers are experienced and are pushing their visions of how the work should be done.

3.7 Conclusion

In the end, we decided to try and make a convolutional neural network as simple as possible, and look whether it suits our needs since CNNs are used most widely for our cause. Nevertheless, but most of the conventional architectures are incredibly complicated, yet we need to produce something that will work fast and not consume many resources.

For comparison, we decided to take HOG + SVM combination due to it being prevalent in cases most similar to our problem. Also, we decided to try some existing architectures of CNNs if there is time left.

Also, we decided to work with picture fragments, to improve performance.

Finally, considering the software development methodology, the decision was to combine prototyping and iterative development, since this combination is the most convenient for this work. This way, we split the work into tiny parts, share them, implement them, and after that — combine them into something whole, which is a prototype of a system we are trying to build. Then it is evaluated, all advantages and disadvantages found and noted, all errors and bug are fixed, and the new cycle is started.

Chapter 4

Implementation

With what we found out during the preliminary research, and extensive knowledge obtained while digging deeper into concepts, now we were ready to start implementation, and here we are going to explain how the process went, what considerations were there, and what problems occurred.

We are going to cover the system requirements first, then move on to high-level system architecture and related considerations, including development plan and programming language choice. After that, we are going to dive deep into details, including data collection, data transformation, prediction, I/O and other parts. Finally, we are going to discuss some strategies for system evaluation, that were used to obtain data presented in the next chapter.

4.1 System requirements

Extensive discussion on the subject of system requirements, along with practical considerations and issues related to lack of resources, led us to the following set of requirements:

- The system should be able to capture video from files
- The system should be able to capture live video streams
- The system should capture frames from videos
- The system should split a frame into slots according to a markdown configuration file
- The system should predict the status of the given slot with an acceptable degree of correctness
- The system should output predictions in a format both human-readable and computer-readable
- The system should save the output in log files

- There should be one log file for each frame
- The log file should contain enough information to reproduce the logged state on the picture, given markdown configuration.
- The system should be multiplatform
- The time to process one frame must not exceed 2 seconds
- The system should be lightweight in terms of consumed memory and computation resources, where lightweight means that an average modern computer should be able to also handle a couple of typical applications like a web-browser, while running the system, without lags in performance.
- The system should be easy to install, launch and run
- The system should be able to show the output visually in real time
- The system should be easily extendible, with additional tools, and modifiable with other predictive models or similar improvements

As one could notice, nothing is said about the origin of configs here. This issue is also going to be discussed, along with an auxiliary tool produced specifically to make parking lot markdown configs.

There were also other requirements that were considered but thrown out:

- The system should determine whether the situation is typical or somehow strange
- The system should have convenient UI for interaction
- The system should be able to improve prediction based on the manual sorting of an unrecognized situation and relearning

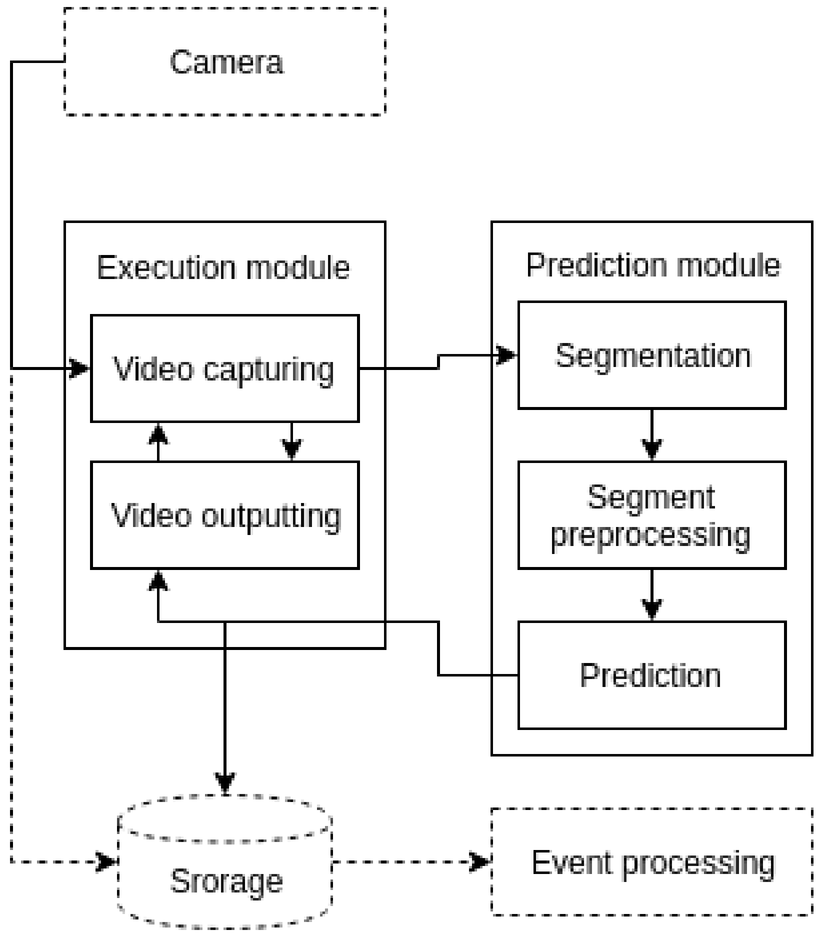
All three were thrown out because of time constraints and drastic increase in complexity of implementation.

4.2 High-level system architecture

Out of requirements listed above, a notion of the high-level architecture of our future tool emerged (Figure 4.1). It is inherently modular, consisting of an execution module and a prediction module, which should communicate with each other.

The execution module would capture a video stream from a camera and send it to a prediction module, while also outputting it on the screen with current labels visualized and continuing video capturing. After the prediction is made, it would update the labels and save them into a log, and then the cycle would start again.

Figure 4.1: High-level system architecture



The prediction module cycle has, on the other hand, much simpler work cycle. It accepts a frame from the execution module, retrieves parking slots from this frame, preprocesses the slots, predicts their status, and outputs prediction back to the execution module, waiting for the next frame to predict on.

4.2.1 Justification of modular architecture

Why did we make the structure modular? This architecture has plenty of advantages, to the extent that it is the recommended option nowadays. The good sides of this approach include:

- Fewer bugs and easier testing, since parts are decoupled
- More natural understanding of internal structure and algorithms, due to

semantically appropriate structurization

- The work can be divided more efficiently, with every developer implementing their part (we did not, though)
- Allows interchangeability of components and easy extensibility

All these features allow saving of time to develop at a price of overhead on more thorough documentation and integration testing.

4.2.2 Essential and additional components

As one could notice, some lines on the diagram are dotted. They are to represent the components we are not responsible for. We cannot guarantee an appropriate camera and storage — this needs to be done by a user. Other developers might also ensure saving of the video stream to storage. Another useful feature would be an event processing module that would parse logs from storage and use them to determine the behavior of other arbitrary devices, thus opening up more space for improvement of the smart parking.

4.2.3 The issue of parallelism

As could be already clear from the diagram and the explanation of it, to work in real time, and to handle video streams from a camera at all, the system should do two actions in parallel, the first being prediction, and the second — drawing and skipping frames while awaiting the prediction. These activities should happen at the same time, which makes a one-thread implementation unviable. Possible ways to handle the issue include:

- Use of asynchronous primitives. This is a solution commonly used in industry. However, some of the libraries that are possible to use to accomplish our goal can be incompatible with the approach.
- Implement modules as separate programs that interact with each other using pipes or queues. An older approach, which is also harder to handle, but which allows circumventing the library restrictions on asynchronous programming.

We tried to use both over the course of development.

4.3 Development plan

This work is in fact quite complex, consisting both of development part, where we need to create a new system from scratch, and research part, where we need to compare it to other existing methods to verify its practical applicability and overall quality. Thus, our plan consists of both research and development parts.

So here is the plan itself:

1. Train SVM model with use of HOG for the sake of further comparison
   (a) Train the network on existing data, get the result
   (b) Train the network on our generated data, get the result
   (c) Choose the model which works better and decide on the size of an appropriate dataset to retrain on
   (d) Retrain the chosen variation on a more prominent segment of the respective dataset
2. Try to use YOLO and (if time allows) other existing computer vision solutions
   (a) Try to use a pre-trained solution and look how it performs on our data
   (b) Retrain the network on a dataset customized for our needs and check on existing parking data we have
   (c) If it works, find more data and train the model on a better dataset
3. Develop the target system
   (a) Make and train a simple CNN
   (b) Implement functionality to cut segments out of frames
   (c) Implement the architectural (integrational) part
   (d) Add optimizations
   (e) Add support for video
   (f) Parallelize
   (g) Repeatedly refactor
4. Test all the models we have, compare them in terms of performance and predictive quality, and put the best model into the system

At the same time, we would have to develop an image labeling tool to produce config, which we are going to cover later. This tool is the reason why we have a notion of our generated data in this plan.

4.4 Justification for programming language

For the system we were creating, there were three choices of programming language:

- C
- C++
- Python

A choice that small is explainable by the fact that these are the languages for which the libraries we would need for the project exist to the fullest. Most other languages either do not have appropriate frameworks for computer vision and machine learning, or only have restricted APIs or poor documentation for these frameworks.

Thus, we decided to stop on Python, due to the following reasons:

- As developers, we have the most experience with this language.
- It is well-suited for prototyping.
- Has an incredible amount of data science frameworks
- More human-readable
- Cross-platform interpretable
- Safer than C/C++ due to lack of direct work with memory
- Extensible in C/C++

These features are especially valuable due to rigid time constraints put on us, which means we would not be able to test every pitfall we might otherwise have run into thoroughly.

However, we are also aware of the language disadvantages:

- Slow, which makes optimization an important consideration
- Harder to test and debug than most of the statically typed languages
- Issues with memory usage, which are manually unsolvable

Due to these facts, it is proposed that possible future development teams that have more time at hand rewrite the system in C++ or similar, thus potentially increasing the processing speed.

4.5 Data collection

Now that we have gone through basic high-level considerations let us concentrate at specific parts of the work, starting with data collection, as it is something to be performed first since details of system implementation might well depend on the kind of data we end up with. Let us go through a couple of things we are going to discuss in detail in this section.We are going to get through requirements for data, then consider existing datasets, and finally consider the work with video datasets.

4.5.1 Overview of existing picture databases

As the primary data collection strategy to employ was chosen to be existing dataset usage, let us look at the most notable picture datasets we ran into over the course of the search.

PASCAL VOC 2012

A general-purpose image dataset for the prediction of 20 object classes, divided into four categories:

- Person: *person*
- Animal: *bird, cat, cow, dog, horse, sheep*
- Vehicle: *airplane, bicycle, boat, bus, car, motorbike, train*
- Indoor: *bottle, chair, dining table, potted plant, sofa, tv/monitor*

The dataset contains:

- 11530 images
- 27450 regions of interest
- 6929 segmentations

The dataset was updated annually over the course of many years and primarily exists as a dataset for computer vision challenges. The goal of the challenge is mostly to recognize objects from twenty preselected classes and classify them appropriately.

This dataset, although employed at some stage, ended up being not suitable to our needs due to its general nature.

Microsoft COCO: Common Objects in Context

A large general purpose image dataset created for the wider variety of tasks as compared to Pascal VOC, including object detection, segmentation and captioning. Includes such special features as:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation

The dataset contains:

- 330 thousand images
- Over 200 thousand of labeled images
- 1.5 million objects

Figure 4.2: Example of image from PASCAL VOC 2012 dataset

- 80 object categories
- 91 stuff categories for superpixel segmentation
- 5 captions per image

This dataset has undoubtedly been much work and was also briefly used at one stage of our research, yet in the end, it turns out it is not suitable for our work due to its generality, while we have a more specific purpose.

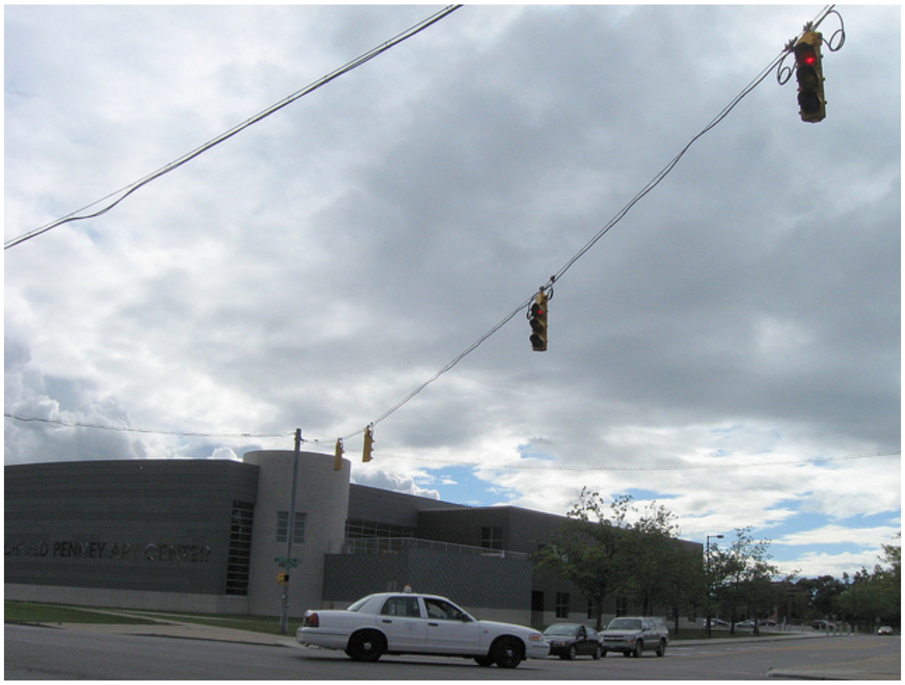
The PKLot database

An open-source database which containing pictures captured from three cameras on two different parking lots over an extended period of sunny, cloudy and rainy days. The dataset contains:

- 12417 images of parking lots (1280X720)
- 695900 images of parking spaces cut from the images of parking lots
- XML files providing markdown and labels for slots on all pictures

Parking lot images are organized into three directories (parking 1a, parking 1b, and parking2). Each directory contains three subdirectories for different weather conditions (cloudy, rainy and sunny). Inside of each subdirectory, the images are organized by acquisition date. It is the dataset that proved to be the

Figure 4.3: Example of image Microsoft COCO dataset



most useful for our research. Both frames and segments were extensively used for the development — spaces were our first training dataset.

Nevertheless, frames further proved to be more useful than segments, since we were not able to find the segmentation tool used to produce this dataset, so we needed to implement our own. The pictures from our output were very different and thus incompatible to those present in the dataset. So later we used frames for checks and testing while generating our segments from frames to train models.

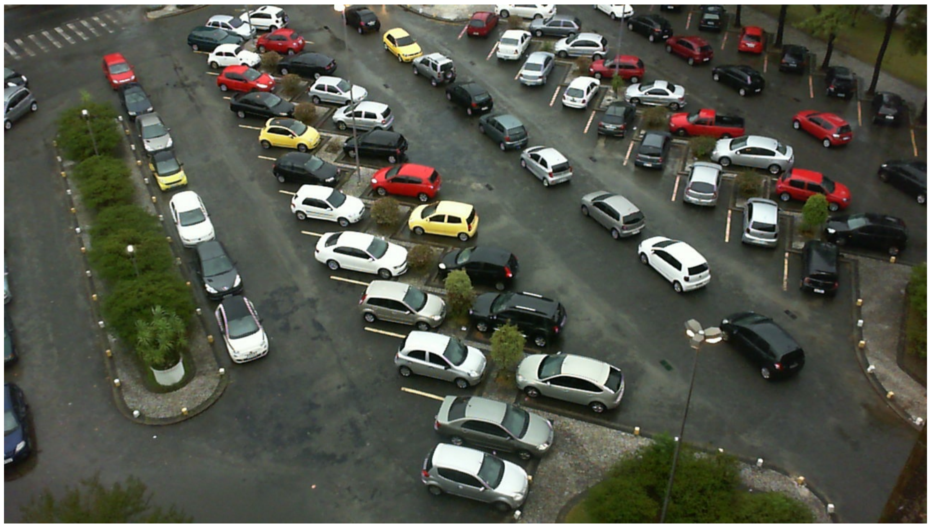
Another issue with this dataset is, it is Brazilian and thus does not contain snowy weather entries which would help improve the prediction in many countries, including Russia.

4.5.2 Video datasets

Unfortunately, we were not able to find any comprehensive video datasets which suite our needs. There were some videos concentrated on pedestrians that were not labeled for cars at all.

Still, we managed to obtain one useful video, 2 minutes long, which we used only for testing purposes. In fact, there are plenty of similar videos out there on the internet, yet they cannot precisely be named datasets since mostly they are demonstrations of how existing tools work, and do not have any usable labels

Figure 4.4: Example of image from PKLot database



but instead have labels as an overlay onto a video which is impossible to get rid of. Thus, we tried to filter those videos so that only those that dont prevent the correct work of our system are used.

4.5.3 Pictures vs. Videos

Work with video is pretty much similar to work with pictures. With videos, it is enough to capture some frames to work with them just like we would with pictures. With video streams, that would mean skipping frames until the prediction is made. Otherwise, it is pretty straightforward.

However, it is not like that when it comes to search for a dataset. It is notable hard to find a useful video dataset on the internet, which makes manual data collection a more suitable option. Furthermore, there is no such thing as video stream dataset, so one only can test work with streams during real-life test runs.

4.6 Data preparations

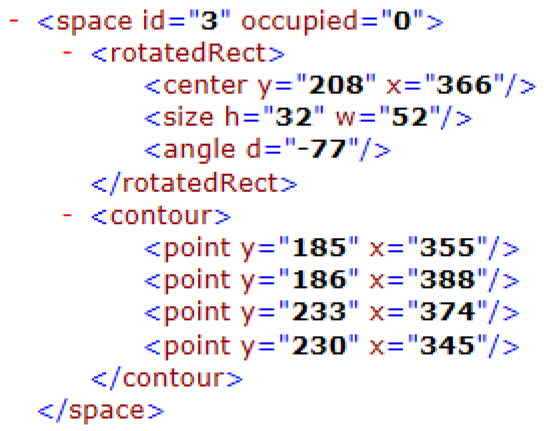
4.6.1 Config format considerations

XML (Extensible Markup Language) is a markup language that defines the rules for document markup in a format that is both human-readable and machine-readable. It is a textual data format which supports a multitude of human languages, used both for document markup and for object representation in programming.

XML is a basis for many other data formats, including office documents, pictures, and formats for web data transfer. The main reason why XML is an excellent choice for markup and configuration is that it is universal and extensible, thus allowing to possibly store everything in one place and create even insanely complex hierarchies of data. On the other hand, it is quite dense and wordy and requires traversing the tree to obtain its elements, which makes coding harder and slower.

XML was used as markup for PKLot (Ex. on Figure 4.5), as well as for PASCAL VOC annotation, and this was the reason we started with this format for configs. On the first iteration, we parsed parking lot coordinates for PKLot from its XML markup, which made for quite a lot of code in terms of line count, which is to be further debugged, tested and possibly maintained, and possibly makes the implementation slower, which made us think of alternatives quite soon.

Figure 4.5: 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>
```

JSON (JavaScript Object Notation) is a format to serialize objects, in both machine-readable and human-readable way. Created first as conforming to JavaScript type system, now the format is mostly language-independent since most of the modern languages include the functionality for JSON serialization and deserialization.

Although JSON supports only a few data types, and the format is not entirely standardized, it also has a much simple syntax as compared to XML and is natively supported by a multitude of languages including Python. Thus it is possible to deserialize data and work with it as with native Python objects.

These considerations made the format our choice for both configuration (markup) files and prediction log files, as a reliable and straightforward format that is easy to use.

Of course, there were thoughts on other options for configuration files and logs. For instance, what if it was **TOML (Tom's Obvious, Minimal Language)**? It is even more human-readable and straightforward. However, on the other hand, it is not ingrained into programming languages as heavily as JSON, and furthermore, the kind of information we are including would not make TOML files any more understandable than JSON. We are bound to work with very long arrays of values, and one would not make them any more readable than they are. Also, this language only supports strings and ints, which may be enough in the beginning but makes for lots of config restrictions, especially if we store the values that are to be deserialized into objects — and we do.

Another thought was to make up our format, yet it would require writing more logic for serialization and deserialization, which we cannot afford at the moment. Why bother if we already have formats well-applicable to our task?

4.6.2 What info to include

The following information was included in XML markdown used in PKLot:

- The parking ID
- Parking space ID and status
- Rotated rectangle (parking space) coordinates in 2 formats:
  - As center, width, height, and angle of rotation
  - As four points denoting vertices

The first drawback of such structure is that there is plenty of information which does not change between frames, namely locations of slots. On the other hand, labeling is easily changed between frames. This issue is what led us to separate these two kinds of data. The unchanging info goes to config, making it one config per parking, while labels are written into logs.

Another consideration is that rectangle markdown would be practically inconvenient for the user, so it is better to make it with arbitrary quadrilaterals, which are a natural representation of how slots look like at a picture, with all perspective warps objects undergo. Thus the config should be a list of slots, where a slot is a list of 4 points, where each point is two coordinates. On the other hand, a log is a map where the slot ID is a key, and a value is either Empty or Occupied. Slot IDs are 1-based and determined from the ordering in the config.

4.6.3 Metadata and labeling issues

Over the course of the discussion, we, as many aspiring developers before us, first set a goal to create an ultimate God tool that can do everything. Thus, we considered many labels for classification, just for everything to be detected. Among them, the sanest set was:

- Person
- Animal

- Truck
- Car
- Empty parking lot

However, over the course of dataset search, as well as architectural a practical considerations, we decided to simplify the set. The first question to ask was — will the possible user care whether there is a truck, or a car, or a bus? The answer was no for most cases, so we ended up with the following binary classification labels:

- Occupied parking space
- Empty parking space

Among all other things, this choice eased or lives in a sense that there was no urgent need to perform manual data collection anymore. Instead, we could find a suitable pre-existing dataset and focus on the system implementation.

There were also other issues we ran into over the course of our research. While with SVM and our CNN everything went smoothly with markdown, during the work with YOLO we needed to transform the annotations we had into an appropriate format (PASCAL VOC annotation format). It would take time to implement necessary scripts. We decided to not start with it yet there still arose a need for plenty of scripts for selective training, so that only data relevant for us is used. The PASCAL VOC markdown is an XML which includes:

- folder name
- filename
- description of an image source
- image size
- segmented or not
- information on objects to detect
  - object category
  - object pose
  - truncated or not
  - occluded or not
  - bounding box as diagonal coordinates
  - difficult object or not

In our data, we did not have most of this information, which would have to be generated have we gone far enough to train YOLO on a representative dataset.

4.7 Image labeling tool

While working on the markdown, we ran into a need to be able to markdown parking spaces manually. It would also be useful for future users of the system. So why would we need to provide markdown for pictures? The answer is since, by definition of a problem we are trying to solve our system must not consume many resources, we decided that we will make predictions slot-by slot. Thus, we need to cut those slots down, using the markdown, which should be done for every new parking-camera pair. One could consider it would be easier to take an existing tool, and there are some, which are used for general-purpose datasets and contain plenty of irrelevant data in obscure formats. Our task is straightforward in terms of classification, that is, we only distinguish between empty and occupied spaces, so we do not need full annotation. We do not need to write down that it is a bike here, it is occluded, and has this set of cartesian coordinates and also those sets of other kinds of coordinates, and such and such mask. We only need to mark shapes of fields — but also we need something more than just rectangles similar to those used as bounding boxes. This issue is why this tool exists — it does precisely what we need. The tool is called, once again, image labeling tool (ILT), and let us talk a bit more about it. The idea is fundamental— 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, one would need to work with some random parking lot that is not yet labeled. This fact explains the need for this tool, along with the fact that we found no tool that would be both allowing for arbitrary shapes and convenient output formats. Then usual programmers thing rolled in, that if there is no tool one is comfortable enough to work with, they probably should code one.

4.7.1 Requirements

- Requirements for the ILT are the following:
- The tool should allow to markdown quadrilateral slots on a picture
- The markdown should be saved in JSON in the form of an array of coordinate pairs
- One should be able to save the work done and continue it later
- One should be able to cancel wrong points or even wrong slots
- The actions above should be done using keyboard and mouse

4.7.2 Architecture

The program consists of two loops, one inside the other. In the innermost loop, the quadrilateral is drawn, while the outer loop ensures there can be many quadrilaterals. Everything else is functionality aimed at program responsiveness, such as mouse click monitoring or picture rendering, as well as ability to save or cancel actions.

4.7.3 Implementation

The whole application was written in Python, with extensive use of OpenCV framework. We first decided to look at how similar applications are made. First of all, there was a part that draws a quadrilateral on a canvas, using a mouse. Then we added an outer loop to draw many quadrilaterals and replaced an empty canvas with a target picture file which can be specified. Then, JSON dumping was added, and finally, other functionality improvements implemented.

4.7.4 Functionality

The way the ILT works is quite straightforward. We are 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 particular array on the single left mouse click, and then a point appears on the screen. Points then make line segments. Then, to finish drawing a shape, one needs to double-click. Also, if one is mistaken, the right button click is to the rescue as it cancels the last point.

After this basic functionality, we added such possibilities as:

- Cancel previous shapes (z)
- Save current shapes into a JSON file (s)
- Open a file dump and continue work (at the start, the tool asks the user whether they want to start a new work or continue some work which has been started previously.)

Thus we can conclude that the tool can do as much as intended.

4.7.5 Usability

Overall, the tool implements the required functionality, yet its usability could be way better. The main disadvantage is that one would confirm their actions all the time, both in a case when they finish drawing slots or cancel some slots (though on the other hand, it helps to avoid misclicks). That issue stemmed from the fact that OpenCV poorly supports parallelizing, which makes background keystroke monitoring impossible. Another disadvantage is that there is no help in UI, so either one knows how to use the tool in advance, or they cannot use it.

These issues undoubtedly should be improved in production, yet our role is more to show a proof of concept. The user guide for this tool can be found in Appendix C.

4.8 Image pre-transformation

There are specific actions we would need to do with the picture anyhow.

The first action is converting to grayscale, which is needed to simplify things, as well as since color does not play a prominent role in our prediction (vehicles can have different colors).

The next one is blurring, which is used to make a comparison with previous frames less susceptible to noise. For the sake of that, we used two kinds of blur, namely median blur and Gaussian blur, with different radii. It is an optimizational need.

Then, one would need to find SSIM to find out whether anything has changed — this is also guided by optimization concerns.

Now, we also need binarization to locate those changes — and the functionality to find contours, for the same reason.

We would also need perspective transformation since when we cut the slots out, we need to transform those fragments into standalone pictures, which implies rectangular.

4.8.1 Library capabilities

We considered several libraries to work with pictures and tried to compare them in terms of functionality we need. Scikit-image and OpenCV came out to be the most potent libraries here (Table 4.1), nearly similar in capabilities. We decided to stick to OpenCV for most of the tasks, since we also use its API in ILT, but use Scikit-image to find SSIM. We also considered other libraries, such as Matplotlib (which is designed for other use cases and thus it is tricky to use it for our purpose) and ImageMagick (which does not have proper documentation for API).

Table 4.1: CV library capabilities

Method	Skimage	OpenCV
rgb2gray	+	+
blur	+	+
compare SSIM	+	C++ only
tresholding	+	+
find contours	+	+
get videoframes as numpy array	+	+
getPerspectiveTransform	similar functionality	+
warpPerspective	+	+

4.8.2 Segment extraction

Now, let us move to an issue of segment extraction. We needed to cut an arbitrary quadrilateral from a picture every time we are cutting slots. That was a trickier part which we solved drawing shapes and applying masks.

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 rectangles, there are also trapezoids and parallelograms, and arbitrary quadrilaterals, too. This issue should be taken into account in any case for the final solution to work better. Finally, coordinates in XML files we got with PKLot dataset we used in the baseline referred to not just rectangles but rotated rectangles, so we were forced to implement it this or some similar 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 sub-image, with the first point being in the bottom-left corner.

After that, we use perspective transformation to change the quadrilateral into the 40x60 rectangle. It is also useful since all cameras come with perspective distortions. Mathematically, it is done via mapping vertices of the quadrilateral to a rectangle of a particular shape and doing the respective affine transformations. The resulting rectangles are the final sub-images that are to be then fed to the model. The reason why everything is transformed into 40x60 pictures is that we need a constant picture size as model input in any case.

4.8.3 The case for optimization — here comes the bounding box

We have been implementing our system iteratively, adding something new on each iteration. The first one did not contain optimizations at all, being a minimal proof of concept. After that, we decided that we need to decrease the number of predictions to make to make the system run faster. Thus we decided to find image difference between two consecutive frames we are predicting for (i.e., different by about a minute). The first try was catastrophic, due to insane amounts of unnecessary noise caught, so optimization failed.

The second try led us to significant improvements, as we added medianBlur, and after that GaussianBlur to make up for small fluctuations. This step made differences actual differences, and not random noise. After our solution started working, we noticed that execution takes too long, that means that for now, our software will not be able to perform in real time. The problem seemed to be that our algorithm for slot search performs a brute-force 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 in the picture, there are multiple overlapping areas with changes, which could be merged so that there are fewer changes.

For the third 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.

4.8.4 Other optimization

todo about parallelism

4.8.5 Benchmarking strategy

In order to determine which optimizations really work, and which do not look like optimizations, we decided to do the following. For each version of preprocessing, we would test it repeatedly (three times) on the same video or picture sequence, while writing down the execution time, then compare average execution time for each version.

4.9 Feature extraction

We have compared two popular libraries for work with imagery which we ended up using in our project regarding well-known feature descriptors being implemented. While they are nearly similar in capabilities, OpenCV proved to be a bit wider (Table 4.2). Nevertheless, for SVM training, we decided to use HOG from Scikit-image due to better convenience at the time and more extensive documentation.

Table 4.2: CV library capabilities

Method	Skimage	OpenCV
BoF	The DAISY local image descriptor based on BoF and SIFT	C++ only
HOG	+	+
background subtraction	possible to implement using existing functionality	+
SIFT	+	+

4.9.1 HOG

Now let us look at details of HOG usage, which we ran into while training our SVMs. There are not many issues, but the most important ones include the question of normalization, the issue of feature vector size and resource usage considerations.

We could have used some normalization on resulting feature vectors, as it is often recommended to do to prepare data. Nevertheless, we believe that this is the case where there is no need for normalization since to do it properly, we would need to work on an incredible number of samples to count the required constants such as range, mean, or variance. Furthermore, while normalization is useful to combine features with different ranges and even orders of magnitude, in our case we have one feature descriptor, which is the same kind of data across all columns. As long as we do not add any other feature vectors, normalization here is believed to do more harm than good.

There exists an inevitable tradeoff between many features and few features. On the one hand, more features mean more information thus potentially leading to better prediction. On the other hand, some features might be irrelevant and thus decrease prediction quality, not even talking about the fact that high dimensionality drastically affects performance. Striving for the balance is imperative. We cannot claim the size of the feature vector that we obtain is perfect, yet there were specific considerations. HOG was calculated on pictures 40x60. We do not consider it practical to make those fragments any smaller since it would make it difficult for a human to recognize the content, yet many predictors have not yet reached the human level. On the other hand, we are still getting several hundred features this way, and having more would likely be impractical.

Had this study been concentrated on other aspects, we would probably consider some feature transformations such as PCA, yet here the SVM + HOG combination is only made for the sake of comparison, so it is not needed to try.

It turns out the calculation of hog is quite resource consuming. The reason is not the time — it took less than 20 minutes to calculate HOG for the whole fragment dataset. It took around 14 GB of memory to do that, so unless we had a lucky chance to use a server with 24GB RAM for feature extraction and model training, it might have taken an eternity.

4.10 Prediction part

4.10.1 Library capabilities

Table 4.3: ML library capabilities

Method	Keras	Scikit-learn	OpenCV
SVM	+	+	+
Neural network constructor (+ convolutional layer)	+	+	+
Grid Search or similar	-	+	-

4.10.2 SVM

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 since SVM is one of the most frequently and successfully used techniques in smart parking, besides artificial neural networks. Another reason was that HOG is representative of objects shape, which is an essential characteristic in the task of detection of the situation on parking lots.

The overall principles of how this combination works are 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 the class border, and in optimization, will only take into account elements with margin below 1.

So we started implementing the plan, and immediately discovered that it is tied to numerous difficulties, some of which got us stuck for really long. Let us now describe in details how it was going on. As previously mentioned, we have found a more or less acceptable dataset (PKLot) and kept working with it for a while.

Let us now remember some facts. Each image of the database has an XML file associated including the coordinates of all the parking spaces and its label (occupied/vacant). By using the XML files to segment the parking space, one will be able to get around 695,900 images of parking spaces.

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

It was evident that for some early prototype we do not 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. Then we started tuning parameters with GridSearchCV (parameters like C parameter, which is responsible for a 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 a result, we almost always got just the same results, well over 99

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 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 main problem with this thing was that it overfitted 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 the scarcity of computational resources, and the dataset was sorted. Further on, when we got more resources for training models, we tried to fix main flaws in the initial experiment.

Further on, we chose another approach. We have calculated HOG for ALL the fragments, then randomly shuffled the dataset, then took 0.1 of the dataset as a training set. By then, we already had the server, so this time the training went well. After that, we repeated the procedure with fragments we have cut by ourselves.

The way the 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 file paths (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 three parts, then one of parts is used for validation and other two for training, where validation is done for each part, totaling three iterations.

The ML model apparently did not show much accuracy since we 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. The prediction was mainly in our data, that is, slots we cut and preprocessed with our 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 acceptable.

Overall, It took about 24 minutes to train each of two last SVMs. Accuracy on validation was 0.98 for PKLot segment data and 0.95 for our segment data.

4.10.3 Yolo - a story of failure

Sometime after that, we were thinking about 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 will not be able to recognize when a car is not parked correctly (such as into two 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 the given object falls into (that would be defined as an object having more than a certain amount of bounding box falling into a given area. The amount could be, for instance, 40% of a bounding box for an 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 on the internet. The thing we ran into and at first, liked, is called YOLO. It is 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 clear; what is bad, they are short, while we would prefer comprehensive documentation). In general, and on general-purpose datasets, it seemed to perform right, so we took a pre-trained 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 and COCO. However, these are general purpose datasets, so we could not just only take and use them. For simplicity, we took VOC 2012 and left there only images of vehicles and people, which 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 four 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 the 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. If not for that fact, we would consider adapting PKLot markdown to YOLO and tried, but with our failure, we moved on to other parts of the plan fast.

4.10.4 A simple CNN (made with keras tutorial)

Now, let us talk about neural networks. Here we found an excellent Keras tutorial on fast and dirty convolutional neural network suitable for small training set sizes (and this is true since more massive training sets made final results worse). So we have 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 five models, similar in structure but trained on different data. Why so many models, and what does different data mean, one might ask. The answer is, we did a certain amount of experimentation.

The structure of all models is the same, and unchanged from the tutorial version, except for a target picture size. The difference is in training datasets.

To train the first model, we used segments that are an inherent part of the PKLot dataset. We took the least recent date from each weather from each parking, resulting in about 11k samples. For validation, we took two most recent dates from each weather from each parking, resulting in about 16k samples. All pictures were resized to 60*40, with some further preparation with standard means (oversampling and randomization with added distortions for robustness). The model shows accuracy on the validation set about 91%, according to Keras report on the network learning.

The second model was similar to the first one, with the only difference in dataset size. This time, for training, we took four least recent dates from each weather from one parking, resulting in about 17k images, and for validation, four most recent dates from each weather from another parking, resulting in about 26k images. As a result of training, the model showed around 96% accuracy on validation.

The third was done a bit differently from previous two, as it came out that training the network on parking space pictures led to poor results of prediction using parking space pictures we extracted in our prototype since they were pretty much different. Thus this model was trained on our data, that was previously extracted into two folders(for empty and occupied spaces) and ordered alphabetically. Then, for the training set, we chose (moved) first 10k images from each category, using a simple bash command. By the same principle, we chose validation images, with 70k images in each category. The model trained showed almost 99% accuracy on validation, according to Keras report.

The fourth model was similar to the third one, with the only difference being a size of training dataset, which was drastically increased. From the remaining pool of images, we took 20k of occupied space images, and 40k of empty space images, which were the first alphabetically. The validation set remained the same as in the previous model. This model also showed almost 99% of accuracy on validation, but during prediction, it worked much worse than the previous model, that became the reason this one was discarded. The reason for such change is supposed to be the size of the training dataset, which seemingly was too much for this kind of network.

The fifth model was born since we found out that for one of the cameras on the parking which had two cameras we did not cut slots for a dataset for some reason. So we found it out and decided to cut it, and got the second folder of our dataset. Thus, this time, for training set we got 5k alphabetically first images from the first part of our dataset remaining, and 5k first images from the second part, for both empty and occupied slots, totaling 20k samples. For validation, we then took 30k empty spaces and 30k occupied spaces from the first part, and 15k empty spaces and 30k occupied spaces from the second part, totaling 105k images. In practice, this is the best performing model we got so far, so this is the one that is currently used in our prototype. Another reason for that might be that previous models were trained with 1 to 3 epochs, while this one was trained with five epochs, even though on the validation set the quality deteriorates after the second epoch.

4.11 Outputting results

We chose to have two ways of outputting results: logs — to store them to be further used by other software systems, and visualization — made in real time for humans.

As it was mentioned before, the file format choice for the output (or log) is now JSON. The filename consists of the timestamp and file extension. The file itself is a dictionary (object), where keys are 1-based parking slot IDs, and values are labels — either empty or occupied. It is the most straightforward possible format that contains all the necessary information.

We output the video we get as an input so that people can also see what is going on in a parking lot by their own eyes. We also output labels and slot numbers as an overlay, — the green label meaning empty, and red label meaning occupied. Thus, we show slot borders as specified in markdown, either in red or green. This way was chosen because it is easy and intuitive so that humans can double check the system.

4.12 System evaluation strategies

4.12.1 Methods of testing

todo how to tes system

4.12.2 Methods of whole-system benchmarking

The last part of work would be to evaluate the whole system. For this, we decided to make a test set of pictures, large enough, such that they are all labeled. For these pictures, we could measure how long the system works to process them all, as well as calculate an overall confusion matrix, and from it, all the necessary measures. Also, we considered it necessary to calculate the average time to process one frame. The process described above is what is necessary to be done for each variation (version) of the system.

Chapter 5

Evaluation and Discussion

...

Chapter 6

Conclusion

...

REFERENCES

Bibliography

[1] M. Hilaga and Y. Shinagawa, “Topology matching for fully automatic similarity estimation of 3D shapes,” *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 203–212, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=383282

[2] 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

[3] S. Nath, A. Deshpande, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan, “IrisNet: An Architecture for Internet-scale Sensing Services,” *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, vol. Berlin, Ge, pp. 1137–1140, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315568

[4] C. H. Zhu, K. Hirahara, and K. Ikeuchi, “Street-parking vehicle detection using line scan camera,” *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 575–580, 2003.

[5] L. Bo, “Using Object Classification To Improve Urban Traffic,” *Science And Technology*, no. 1, 2003.

[6] C.-H. L. C.-H. Lee, M.-G. W. M.-G. Wen, C.-C. H. C.-C. Han, and D.-C. K. D.-C. Kou, “An automatic monitoring approach for unsupervised parking lots in outdoors,” *Proceedings 39th Annual 2005 International Carnahan Conference on Security Technology*, 2005.

[7] S. Midrange and P. Cars, “Vehicle Detection with a Mobile Camera,” *Entropy*, no. March, pp. 37–43, 2005.

[8] 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.

[9] G. Toulminet, M. Bertozzi, S. Mousset, A. Bensrhair, and A. Broggi, “Vehicle detection by means of stereo vision-based obstacles features extraction

and monocular pattern analysis," *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2364–2375, 2006.

[10] R. J. López Sastre, P. Gil Jiménez, F. J. Acevedo, and S. Maldonado Bascón, "Computer algebra algorithms applied to computer vision in a parking management system," *IEEE International Symposium on Industrial Electronics*, no. 3, pp. 1675–1680, 2007.

[11] S. F. Lin, Y. Y. Chen, and S. C. Liu, "A vision-based parking lot management system," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 2897–2902, 2007.

[12] G. Cormode and A. Mcgregor, "Approximation Algorithms for Clustering Uncertain Data Categories and Subject Descriptors," *roceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 191–200, 2008.

[13] A. Chianese, V. Moscato, and A. Picariello, "Detecting abnormal activities in video sequences," *Proceedings of the First International Conference on Ambient Media and Systems*, 2008. [Online]. Available: http://eudl.eu/doi/10.4108/ICST.AMBISYS2008.2827

[14] T. Fabián, "An algorithm for parking lot occupation detection," *Proceedings - 7th Computer Information Systems and Industrial Management Applications, CISIM 2008*, pp. 165–170, 2008.

[15] D. Bong, K. Ting, and K. Lai, "Integrated approach in the design of car park occupancy information system (COINS)," *IAENG International Journal of Computer . . .* , no. February, 2008. [Online]. Available: http://www.doaj.org/doaj?func=fulltext{&}aId=380949

[16] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1565–1575, 2008.

[17] M. Chitnis, Y. Liang, and J. Zheng, "Wireless line sensor network for distributed visual surveillance," *. . . Wireless Ad Hoc, Sensor, . . .* , no. January, pp. 71–78, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1641890

[18] 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

[19] 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.

[20] O. David-tabibi, N. S. Netanyahu, and M. Shimoni, "Genetic Algorithms for Automatic Classification of Moving Objects," vol. 20742, pp. 2069–2070, 2010.

[21] J. Salvador, X. Suau, and J. R. Casas, "From silhouettes to 3D points to mesh," *Proceedings of the 1st international workshop on 3D video processing - 3DVP '10*, p. 19, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1877791.1877797

[22] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrashekharan, W. Xue, M. Gruteser, and W. Trappe, "ParkNet : Drive-by Sensing of Road-Side Parking Statistics," *Challenge*, pp. 123–136, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1814433.1814448

[23] H. Ichihashi, T. Katada, M. Fujiyoshi, A. Notsu, and K. Honda, "Improvement in the performance of camera based vehicle detector for parking lot," *International Conference on Fuzzy Systems*, vol. 1, no. 5, pp. 1–7, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5584554

[24] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C. C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, and M. Desai, "AVSS 2011 demo session: A large-scale benchmark dataset for event recognition in surveillance video," *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2011*, no. 2, pp. 527–528, 2011.

[25] I. Kamal, "Car recognition for multiple data sets based on histogram of oriented gradients and support vector machines," *2012 International Conference on Multimedia Computing and Systems*, no. 3, pp. 328–332, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6320284

[26] K. Choeychuen, "Available car parking space detection from webcam by using adaptive mixing features," *JCSSE 2012 - 9th International Joint Conference on Computer Science and Software Engineering*, pp. 12–16, 2012.

[27] K. Raman, K. M. Svore, R. Gilad-bachrach, and C. J. C. Burges, "Learning from Mistakes : Towards a Correctable Learning Algorithm Categories and Subject Descriptors," pp. 1930–1934, 2012.

[28] S. Gokul, G. S. Kumar, and M. Sreeraj, "Real time recognition of pedestrian and vehicles from videos," *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology - CCSEIT '12*, pp. 517–523, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2393216.2393303

[29] Z. Zhang, A. Mistry, W. Yin, and P. L. Venetianer, "Embedded smart sensor for outdoor parking lot lighting control," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 54–59, 2012.

[30] C. C. Huang, Y. S. Dai, and S. J. Wang, "A surface-based vacant space detection for an intelligent parking lot," *2012 12th International Conference on ITS Telecommunications, ITST 2012*, pp. 284–288, 2012.

[31] N. Goyette, "changedetection . net : A New Change Detection Benchmark Dataset," pp. 1–8, 2012.

[32] Q.-Y. Zhou and V. Koltun, "Dense scene reconstruction with points of interest," *ACM Transactions on Graphics*, vol. 32, no. 4, p. 1, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2461912.2461919

[33] B. Jiang and X. Liu, "A divide-and-conquer approach to large scene reconstruction with interactive scene analysis and segmentation," *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry - VRCAI '13*, pp. 283–284, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2534329.2534372

[34] L. Lambrinos and A. Dosis, "Applying mobile and internet of things technologies in managing parking spaces for people with disabilities," *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct*, pp. 219–222, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2494091.2494162

[35] D. Delibaltov, W. Wu, R. P. Loce, and E. A. Bernal, "Parking lot occupancy determination from lamp-post camera images," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, no. Itsc, pp. 2387–2392, 2013.

[36] 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.

[37] A. Chayeb, N. Ouadah, Z. Tobal, M. Lakrouf, and O. Azouaoui, "HOG based multi-object detection for urban navigation," *2014 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2014*, pp. 2962–2967, 2014.

[38] S. Karayev, M. Fritz, and T. Darrell, "Anytime recognition of objects and scenes," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 572–579, 2014.

[39] 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.

[40] A. Vora, M. A. Kumar, and K. G. Srinivasa, "Low Cost Internet of Things Based Vehicle Parking Information System," *Proceedings of the 6th IBM Collaborative Academia Research Exchange Conference (I-CARE) on I-CARE 2014*, pp. 16:1—-16:4, 2014. [Online]. Available: http://doi.acm.org/10.1145/2662117.2662133

[41] E. J. Sen, K. Deepa Merlin Dixon, A. Anto, M. V. Anumary, D. Mieheal, F. Jose, and K. J. Jinesh, "Advanced license plate recognition system for car parking," *International Conference on Embedded Systems, ICES 2014*, no. Ices, pp. 162–165, 2014.

[42] B. R. Payne, J. F. Lay, and M. A. Hitz, "Automatic 3D Object Reconstruction from a Single Image," *Proceedings of the 2014 ACM Southeast Regional Conference*, vol. 1, no. 1, pp. 31:1—-31:5, 2014. [Online]. Available: http://doi.acm.org/10.1145/2638404.2638495

[43] S. S. Mathew, Y. Atif, Q. Z. Sheng, and Z. Maamar, "Building sustainable parking lots with the Web of Things," *Personal and Ubiquitous Computing*, vol. 18, no. 4, pp. 895–907, 2014.

[44] A. K. Gupta, G. Ye, and J. Vendrig, "Hybrid Resolution Based Video Foreground Detection," 2014.

[45] 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.

[46] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[47] Y. Sakai, T. Oda, M. Ikeda, and L. Barolli, "An Object Tracking System Based on SIFT and SURF Feature Extraction Methods," *2015 18th International Conference on Network-Based Information Systems*, pp. 561–565, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7350677/

[48] Jing Shao; Kai Kang; Chen Change Loy; XiaogangWang, "Deeply Learned Attributes for Crowded Scene Understanding," *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4657–4666, 2015.

[49] M. Dixit, S. Chen, D. Gao, N. Rasiwasia, and N. Vasconcelos, "Supplement : Scene Classification with Semantic Fisher Vectors," p. 6964, 2015.

[50] Y. Yuan, L. Mou, and X. Lu, "Scene Recognition by Manifold Regularized Deep Learning Architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2222–2233, 2015.

[51] H. Ye, Z. Wu, R.-w. Zhao, X. Wang, Y.-g. Jiang, and X. Xue, "Evaluating Two-Stream CNN for Video Classification Categories and Subject Descriptors," pp. 435–442, 2015.

[52] H. Rajput, T. Som, and S. Kar, "An automated vehicle license plate recognition system," *Computer*, vol. 48, no. 8, pp. 56–61, 2015.

[53] F. J. Lin and H. Chen, "Improving utilization and customer satisfaction of parking space with M2M communications," *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pp. 465–470, 2015.

[54] 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

[55] J. M. Menéndez, C. G. del Postigo, and J. Torres, "Vacant parking area estimation through background subtraction and transience map analysis," *IET Intelligent Transport Systems*, vol. 9, no. 9, pp. 835–841, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7305851/

[56] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016.

[57] S.-M. Chen and C.-K. Chiang, "Rotation, Translation, and Scale Invariant Bag of Feature Based on Feature Density," *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pp. 163–168, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7877207/

[58] H. Huttunen, F. S. Yancheshmeh, and C. Ke, "Car type recognition with Deep Neural Networks," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2016-Augus, no. Iv, pp. 1115–1120, 2016.

[59] S. M. Alwan, M. A. El-Abed, and R. N. Zantout, "Vehicles recognition from partial images," *2016 3rd International Conference on Advances in Computational Tools for Engineering Applications, ACTEA 2016*, pp. 236–240, 2016.

[60] T. H. N. Le, Y. Zheng, C. Zhu, K. Luu, and M. Savvides, "Multiple Scale Faster-RCNN Approach to Driver's Cell-Phone Usage and Hands on Steering Wheel Detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 46–53, 2016.

[61] G. K. Yadav, P. Shukla, and A. Sethi, “ACTION RECOGNITION USING INTEREST POINTS CAPTURING DIFFERENTIAL MOTION INFORMATION Department of Electronics and Electrical Engineering , IIT Guwahati Department of Mathematics , IIT Guwahati , Guwahati,” *Icassp 2016*, pp. 1881–1885, 2016.

[62] 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.

[63] 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.

[64] 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.

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

[66] 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.

[67] R. Du, S. Bista, and A. Varshney, “Video fields: fusing multiple surveillance videos into a dynamic virtual environment,” *Proceedings of the 21st International Conference on Web3D Technology - Web3D ’16*, pp. 165–172, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2945292.2945299{%}5Cnhttps://www.youtube.com/watch?v=5h9r2ksdJQc

[68] H. Assem, L. Xu, T. S. Buda, and D. O’Sullivan, “Machine learning as a service for enabling Internet of Things and People,” *Personal and Ubiquitous Computing*, vol. 20, no. 6, pp. 899–914, 2016.

[69] V. Jain, Z. Sasindran, A. Rajagopal, S. Biswas, H. S. Bharadwaj, and K. R. Ramakrishnan, “Deep automatic license plate recognition system,” *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing - ICVGIP ’16*, pp. 1–8, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3009977.3010052

[70] J. Cho and J. Park, “Automatic Parking System using Background Substraction with CCTV Environment,” 2016, pp. 1649–1652.

[71] L. Baroffio, L. Bondi, M. Cesana, A. E. Redondi, and M. Tagliasacchi, “A visual sensor network for parking lot occupancy detection in Smart Cities,” *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pp. 745–750, 2016.

[72] 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.

[73] 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/

[74] M.-c. Roh and J.-y. Lee, “Refining Faster-RCNN for Accurate Object Detection,” pp. 3–6, 2017.

[75] I. Kamal and J. Oubaha, “Car recognition using the bag of features method,” *International Conference on Multimedia Computing and Systems -Proceedings*, pp. 99–102, 2017.

[76] S. Ardianto, C.-J. Chen, and H.-M. Hang, “Real-time traffic sign recognition using color segmentation and SVM,” *2017 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1–5, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7965570/

[77] X. Xiang, N. Lv, M. Zhai, and A. El Saddik, “Real-time Parking Occupancy Detection for Gas Stations Based on Haar-AdaBoosting and CNN,” *IEEE Sensors Journal*, vol. 17, no. 19, pp. 6360–6367, 2017.

[78] N. Passalis and A. Tefas, “Learning Neural Bag-of-Features for Large-Scale Image Retrieval,” pp. 1–12, 2017.

[79] X. Zhao, W. Li, Y. Zhang, T. A. Gulliver, S. Chang, and Z. Feng, “A faster RCNN-based pedestrian detection system,” *IEEE Vehicular Technology Conference*, 2017.

[80] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[81] X. Wang, L. Lu, H. C. Shin, L. Kim, M. Bagheri, I. Nogues, J. Yao, and R. M. Summers, “Unsupervised joint mining of deep features & image labels for large-scale radiology image categorization & scene recognition,” *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pp. 998–1007, 2017.

[82] 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

[83] 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.

[84] 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/

[85] O. Bulan, V. Kozitsky, P. Ramesh, and M. Shreve, "Segmentation- and Annotation-Free License Plate Recognition With Deep Localization and Failure Identification," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2017.

[86] 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

[87] 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.

[88] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[89] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[90] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

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.